UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 5- Entornos Virtuales e Informática Industrial

TÍTULO: **Herramienta de Generación de Código Mediante Sistema Experto**

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR: **Lázaro Campoalegre Vera**

TUTOR: **Msc.Yuniesky Coca Bergolla**

ASESOR: **Msc. Pedro Carlos Pérez Martinto**

Ciudad de la Habana, 10 de Junio del 2008.
Año 50 de la Revolución

A la memoria de mis abuelos

María Teresa, Enma, Margarito y Arquímedes:

Por la enorme necesidad de entregarles más cariño, por su sacrificio ante la vida y la nobleza con la que crearon mi familia.

A mis padres

…por ayudarme a construir cada uno de mis sueños perpetuos y por su aporte en el largo camino de llegar hasta aquí.

Al Comandante en Jefe Fidel Castro Ruz

…por la monumental idea de crear la UCI.

A la decana Mayra Durán y a todos mis profesores

…por sus conocimientos aportados y ejemplos de profesionalidad que me servirán para servir orgullosamente a nuestra patria.

A mi tutor Yuniesky Coca

…por la seguridad trasmitida en cada paso y por aceptar la idea de esta investigación.

Al Asesor Martinto

…por la ayuda durante el diseño teórico

A mis hermanas Teresita, Anabel y Yuliet

…por el entusiasmo con que apoyan mis metas.

A Maria Esther

…por la alegría con que ha recibido mis resultados docentes desde la primaria.

A Adrian, Ariel, Tony, Felipe , Fonseca y Lianet

…por el apoyo incondicionl en la tesis.

A mis amigos

…por ayudarme a levantar ante las dificultades y por empeñarse en hacerme sentar cuando no he debido andar.

A Marta y David

…por acogerme como un hijo en Venezuela

A todos los que han contribuido al resultado de esta tesis

Muchas Gracias

Resumen

La generación de código ha devenido en una necesidad casi obligatoria para la mayoría de los programadores durante el desarrollo de software. Numerosas herramientas conocidas como CASE (Software Asistido por Computadoras) permiten la generación automática de código a partir de diagramas o modelos. Utilizar o construir un conjunto de librerías que provean funcionalidades que permitan utilizar los recursos que brinda UML y que a la vez los ordene y seleccione automáticamente, constituiría un nuevo paso en la generación de código. La conformación de un sistema experto es la alternativa capaz de hacer más inteligente el problema de la generación automática de código fuente. Este proyecto de tesis presenta los pasos para construir una herramienta capaz de obtener código fuente en el lenguaje de programación especificado por el usuario. Haciendo uso de la biblioteca de clases de código abierto ExpertCoder, se muestra un primer resultado de la construcción de HGCE (Herramienta de Generación de Código mediante Sistema Experto).

Índice de Contenido

Introducción

En la actualidad existen numerosas alternativas para disminuir la carga laboral de los programadores, sobre todo la generación automática de código permite este propósito. Sin embargo, la dificultad de aprendizaje inmediato de las herramientas de generación de código, la sobrecarga de funcionalidades y la diversidad de plataformas de desarrollo; impiden que el nivel de abstracción del programador sea una técnica universal.

Una gran cantidad de herramientas soportan UML (Unified Modeling Language), es un lenguaje, que permite modelar (analizar y diseñar) sistemas orientados a objetos. Muchas de las herramientas que utilizan UML tienen las características que reducen significativamente el trabajo de los programadores, lo que las hace ser consideradas Herramientas CASE. Ninguna de estas herramientas trata la generación de código desde la esfera de la inteligencia artificial, por lo que las decisiones y cambios específicos entre un componente u otro muchas veces tienen que rectificarse por el desarrollador luego de ser generado el código fuente de manera automática.

El factor crítico a la hora de la construcción de un generador de código es la difícil tarea de la automatización de la codificación de las reglas del negocio. Los planteamientos hasta el momento sobre generación de código mediante herramientas de software, tratan el tema para aquellas capas de la arquitectura que no pertenecen a la codificación de las reglas del negocio. Como ejemplo de estas capas tenemos la capa de acceso a datos, los procedimientos de inserción, modificación y actualización. La generación de las clases con sus atributos y la declaración de los métodos. Los métodos solamente se declaran. Inclusive se ha planteado la generación de la capa de presentación, con unos estándares previamente definidos, scripts de creación de la base de datos y sus objetos.

La capa donde se codifican las reglas del negocio, es llamada habitualmente BL (Business Logic) lógica del negocio, es la capa donde se concentra la mayor complejidad de la codificación de software; en ocasiones las reglas del negocio afectan el comportamiento de los eventos en formularios y ventanas. Tradicionalmente para escribir generadores de códigos se crean generalmente programas con una serie de comandos estilo “print”. Sin embargo este tipo de enfoque resulta muy difícil para el autor del generador a la hora de realizar alguna modificación en el diseño del código que se genera, simplemente porque este no se asemeja al código del generador.

Problema

La ineficiente automatización de la codificación de las reglas del negocio en los generadores de código fuente para clases de formularios.

Herramientas CASE

La tecnología avanza de manera sorprendente, surgen nuevas y mejores formas de hacer las cosas, siempre buscando métodos más efectivos, confiables, con mayor calidad y menos riesgos. Las herramientas CASE (Ingeniería Asistida por Computadora) aparecen para auxiliar a los desarrolladores de software, brindando un apoyo computarizado en parte del ciclo de desarrollo de un software. Pero el principal objetivo de estas herramientas es dar solución a varios problemas inherentes al diseño del software, principalmente para solucionar el problema de la mejora de la calidad del desarrollo en sistemas de mediano y gran tamaño.

Los inicios de herramientas informáticas que ayudan a crear nuevos proyectos informáticos se remontan a la década del 1970 , cuando un proyecto llamado ISDOS diseñó un producto que analizaba algún tipo de relación existente entre los requisitos de un determinado problema y las necesidades que estos generaban, a este lenguaje se le denominó Problem Statement Language cuyas siglas eran:(PSL) y la aplicación que permitía a los diseñadores buscar las necesidades se le llamó diseñadores PSA (Problem Statement Analyzer).
Sin embargo, la primera herramienta CASE apareció en el año 1984 con el nombre de Excelerator que trabajaba bajo la plataforma PC.

El término CASE puede ser generalmente aplicado a cualquier sistema o colección de herramientas que ayuda a automatizar el proceso de diseño y desarrollo de software. Compiladores, editores estructurados, sistemas de control de código fuente, y herramientas de modelado, son todas, estrictamente hablando herramientas CASE. Ellas impiden a los programadores tratar directamente con el hardware y les permiten trabajar en un alto nivel de abstracción en la definición de un sistema de software que será construido.

Al llegar la década del 1990, IBM (International Business Machines) había conseguido una alianza con la empresa de software AD/Cycle para trabajar con sus mainframes, estos dos gigantes trabajaban con herramientas CASE que abarcaban todo el ciclo de vida del software. Pero poco a poco los mainframes han ido siendo menos utilizados y actualmente solo lo utilizan las grandes empresas.

Durante algún tiempo se han estado utilizando por los desarrolladores, estas herramientas que se pueden diferenciar tanto por las plataformas que soportan, como por las fases del ciclo de vida del desarrollo de sistemas que cubren, así como por la arquitectura de las aplicaciones que producen. Como ejemplo de estas herramientas tenemos a AllFusion ERWin, ArgoUML, Eclipse, EasyCase, INNOVATOR, Rational Rose, Umbrello, Visual Paradigm para UML, Argo UML y muchísimos otros.

Muchas empresas se han extendido a la adquisición de herramientas CASE, con el fin de automatizar los aspectos claves de todo el proceso de desarrollo de un sistema, desde el principio hasta el final, e incrementar su posición en el mercado competitivo. Pero obteniendo algunas veces elevados costos en la adquisición de la herramienta y costos de entrenamiento de personal así como la falta de adaptación de la herramienta a la arquitectura de la información y a las metodologías de desarrollo utilizadas por la organización. Por otra parte, algunas herramientas CASE no ofrecen o evalúan soluciones potenciales para los problemas relacionados con sistemas o virtualmente no llevan a cabo ningún análisis de los requerimientos de la aplicación, por lo que las posibilidades brindadas por las mismas necesitan rectificación.

Pero de cualquier manera la Ingeniería Asistida por Computadora proporciona un conjunto de herramientas semiautomatizadas y automatizadas que están desarrollando una cultura de ingeniería nueva para muchas empresas. Uno de los objetivos más importante de este tipo de software(a largo plazo) es conseguir la generación automática de programas desde una especificación a nivel de diseño.

UML

La complejidad que ha alcanzado la Ingeniería del Software, impulsó a los programadores a realizar diagramas que describen como hacer las cosas durante el proceso de desarrollo, esto ha ocasionado el surgimiento de metodologías de Análisis y Diseño Orientada a Objetos (OO – Object Oriented) (Orientado a Objetos) y herramientas CASE para facilitar el uso de tales métodos. Sin duda alguna, la existencia de muchos de estos métodos provocó un importante problema de estandarización.

Para eliminar esta dificultad en el mundo del desarrollo de software surge en octubre de 1994 UML de Rational Corporation, cuyo principal propósito era lograr la unificación de los métodos Grady Booch (Booch), James Rumbaugh (OMT) y Jacobson (Objectory). El UML fue aceptado por el OMG (Object Management Group – Grupo Administrador de Objetos) como un estándar en Noviembre de 1997.

Pero la finalidad del UML es ser un lenguaje de modelado independiente de cualquier método. Es un lenguaje gráfico utilizado para el desarrollo de componentes de software, el cual puede ser aplicado en diversas áreas como el comercio electrónico, actividades bancarias, arquitecturas de codificación y otras áreas. Según varios expertos incluyendo a sus creadores James Rumbaugh, Ivar Jacobson y Grady Booch, UML es el sucesor de las notaciones de modelado y elimina las diferencias entre semánticas y notaciones. [1]

Es un lenguaje gráfico para visualizar, especificar, construir y documentar los componentes de un sistema de software, permite tanto la especificación conceptual de un sistema como la especificación de elementos concretos, como pueden ser las clases o un diseño de base de datos. [1]

Generación de Código Fuente

Proveer el código fuente que le interesa al programador ha sido un requisito de cada una de las herramientas que se encargan de automatizar este proceso en el desarrollo de software. Actualmente existen varios productos que permiten abstraer un poco a los desarrolladores, durante el proceso de desarrollo al generar el código de los programas. Por ejemplo CodeDOM (Code Document Object Model) permite generar código a través de una abstracción completa de los lenguajes de programación. Cuando se escribe código a través de estas abstracciones, un provider (proveedor) de .Net para un lenguaje como VB .Net ó C#, ó cualquier otro provider de terceros permite generar código para cualquiera de los lenguajes soportados por el.

En la raíz del modelo de objetos se encuentra el graph CodeDOM. Esto es un árbol de objetos que describe los elementos del código en términos abstractos. El árbol es llamado también CodeCompileUnit, el cual describe el código en partes elementales. Cada árbol o grafo CodeDOM da como resultado un archivo, pero podría contener múltiples clases. Como se mencionaba, usando diferentes providers se podría generar código para múltiples lenguajes.

Con CodeDOM podemos crear generadores de clase en base a una Arquitectura. Una ventaja de CodeDOM es su capacidad de generar código para múltiples lenguajes. El uso de los generadores de código debe ser parte importante en el desarrollo y diseño de una Arquitectura de Aplicaciones y que tiende a acelerar el desarrollo de aplicaciones.

Otra herramienta es GeneXus, se utiliza para el diseño y desarrollo de software multiplataforma. Permite el desarrollo incremental de aplicaciones críticas de negocio de forma independiente de la plataforma. Genera el 100% de la aplicación. Basado en los requerimientos de los usuarios realiza el mantenimiento automático de la base de datos y del código de la aplicación, sin necesidad de programar.

En la herramienta MPO, el usuario define los objetos, sus propiedades y sus relaciones, y la aplicación se encarga de generar todo el código necesario para leer, eliminar, actualizar y listar los datos de la base de datos.

Este proyecto deberá materializar la claridad que aporta el uso de plantillas (documento de texto plano, donde pueden escribirse algunos marcadores especiales) para la generación de código, así como reutilizar y explotar el poder de expresión que brindan las plataformas actuales de programación en fin de lograr una generación de código fuente, basado en sistemas expertos. Al estar basado en estos principios los generadores resultantes deberán ser mucho más extensibles, modulares con posibilidades de estructuras declarativas y no imperativas.

Para lograr un ambiente adecuado de generación en esta herramienta, no interesa la posición ni el tamaño de las cajas, ni tampoco el grosor ni color de las líneas que brinda los diagramas UML; solo se tienen en cuenta los conceptos representados por estas imágenes. Al conjunto de los conceptos representados por los componentes anteriores es lo que se conoce como modelo y es en definitiva diseño del diagrama que quiere representar el autor de algún modelo original.

Prácticamente la" Herramienta de Generación de Código Mediante un Sistema Experto ", que se desarrollará, deberá permitir obtener el código fuente de un modelo UML sin carencia de especificaciones, diferenciación y refinamiento de características entre un componente u otro de una clase.

La teoría de los sistemas expertos concuerda con la idea de software para hacer software. Los sistemas expertos encapsulan el conocimiento de una persona en un campo específico. Con este conocimiento los sistemas expertos pueden solucionar problemas, tomar decisiones y hasta perfeccionar su comportamiento e incrementar su conocimiento. En un acercamiento a un hipotético sistema experto generador de código, que encapsule las habilidades de un programador con experiencia y pueda crear software.

Objeto de estudio y Campo de acción

Teniendo en cuenta lo descrito en la introducción la presente tesis presenta como **Objeto de Estudio** La generación automática de código fuente a partir de modelos UML. Para precisar el objeto de estudio se prioriza como **Campo de Acción**, la generación automática de código de clases para formularios a partir de modelos UML, mediante la utilización de herramientas y librerías que permitan la creación de un Sistema Experto.

Objetivo General

Desarrollar una herramienta de generación de código fuente para clases de formularios que utilice un sistema experto.

Tareas de la investigación

- Estudiar las características de las herramientas CASE
- Estudiar las ventajas y desventajas de UML.
- Seleccionar y aplicar un método adecuado de serialización de modelos.
- Estudiar los diferentes tipos de Sistemas Expertos.
- Estudiar técnicas novedosas para la utilización de Sistemas Expertos.
- Utilizar librerías y herramientas de software para la confección de un Sistema Experto.
- Integrar las tareas anteriores en la puesta en marcha del generador.

Capítulo 1: Fundamentación teórica

En este capítulo, se sustenta el problema científico y el propósito de este trabajo, mediante la fundamentación teórica, formada por un grupo de referencias seleccionadas e interpretaciones, análisis y valoraciones de los temas tratados.

Los sistemas de desarrollo de software cada vez más profundos y complejos son la demanda del mercado, el volumen de información que se maneja, tanto en organizaciones como instituciones han guiado el crecimiento en las capacidades de desarrollo. La Ingeniería de software ha creado métodos y herramientas que ayudan a los programadores a desarrollar aplicaciones de forma más rápidas y de mejor calidad. Existen varias definiciones de Ingeniería del Software; Roger S. Pressman considera que es la ciencia que ayuda a elaborar sistemas con el fin de que sea económico, fiable y que funcione eficientemente sobre las máquinas reales. [2]

Podemos considerar que el uso de la Ingeniería del Software posibilita algunas ventajas como la obtención de un nivel competitivo, mejora de la uniformidad de los métodos, la adaptación de la automatización del analista, los cambios de los métodos de trabajo a lo largo de un proyecto, entre otras. En el año 1990 la IEEE definió a la ingeniería del software como:" La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento de software". [3]

Desde el inicio de la escritura de software, ha existido un conocimiento de la necesidad de herramientas automatizadas para ayudar al diseñador del software. Inicialmente, la concentración estaba en herramientas de apoyo a programas como traductores, recopiladores, ensambladores, procesadores de macros, montadores y cargadores.

Este conjunto de aplicaciones que pueden informatizarse, aumentó dramáticamente en un breve espacio de tiempo, causando una gran demanda por nuevo software a desarrollar. A medida que se escribían nuevos software, habían ya en existencia millones y millones de líneas de código que necesitaban ser mantenidas y actualizadas.

Esto causó a la industria de las computadoras muchos problemas, no podía cubrir el incremento de la demanda con los métodos que se estaban usando. Esto fue reconocido como una crisis de software. Para

superar este problema en el proceso de desarrollo de software, se introdujeron metodologías para intentar crear estándares de desarrollo. Hay también otra manera en la que la industria ha ayudado a superar las dificultades de uso de esta tecnología disponible. La industria de computadoras ha desarrollado un soporte automatizado para el desarrollo y mantenimiento de software. Este es llamado *Computer Aided Software Engineering* (CASE) Ingeniería de Software asistida por computadora.

1.1Herramientas CASE

La Ingeniería de Software asistida por computadora, refina las posibilidades de la Ingeniería de Software, pues tiene como objetivo proporcionar un conjunto de herramientas bien integradas, que simplifiquen el trabajo, utilizando técnicas de unificación y automatizando todas o algunas fases de ciclo de vida del software. CASE es una herramienta que mejora el trabajo de un ingeniero de software para desarrollar sistemas de cómputo.

Se puede definir a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos o algunos los pasos del Ciclo de Vida de desarrollo de un Software. Como es sabido, los estados en el Ciclo de Vida de desarrollo de un Software son: Investigación preliminar, análisis, diseño, implementación e instalación.

Una herramienta Case puede verse también como una innovación en la organización, un concepto avanzado en la evolución de tecnología con un potencial efecto profundo en la organización. Se puede ver al CASE como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales.

Un nuevo software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Las Herramientas CASE fueron desarrolladas para automatizar esos procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo. [4]

1.1.1 Generalidades de herramientas CASE integradas

El CASE permite a las compañías competir más efectivamente usando estos sistemas desarrollados nuevamente para compararlos con sus necesidades de negocio actuales. En un mercado altamente competitivo, esto puede hacer la diferencia entre el éxito y el fracaso. Las herramientas CASE también permiten a los analistas tener más tiempo para el análisis y diseño y minimizar el tiempo para codificar y probar.

La introducción de CASE integradas está comenzando a tener un impacto significativo en los negocios y sistemas de información de las organizaciones. Con un CASE integrado, las organizaciones pueden desarrollar rápidamente sistemas de mejor calidad para soportar procesos críticos del negocio y asistir en el desarrollo y promoción intensiva de la información de productos y servicios. Estas herramientas pueden proveer muchos beneficios en todas las etapas del proceso de desarrollo de software, algunas de ellas son:

- Verificar el uso de todos los elementos en el sistema diseñado.
- Automatizar el dibujo de diagramas.
- Ayudar en la documentación del sistema.
- Ayudar en la creación de relaciones en la Base de Datos.
- Generar estructuras de código.

1.1.2 Ventajas de las herramientas CASE

La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Para conseguir estos dos objetivos es conveniente contar con una organización y una metodología de trabajo, además de la propia herramienta. La mejora de calidad se consigue reduciendo sustancialmente muchos de los problemas de análisis y diseño, inherentes a los proyectos de mediano y gran tamaño (lógica del diseño, coherencia, consolidación, etc.). La mejora de productividad se consigue a través de la automatización de determinadas tareas, como la generación de código y la reutilización de objetos o módulos.

1.1.2.1 Facilidad para la revisión de aplicaciones

Una vez que se implementa una aplicación, se emplea por mucho tiempo o cuesta un poco de trabajo hacer un despliegue inmediato de la nueva versión, pues esta no se concluye con poca diferencia de tiempo a la versión anterior. Cuando se crea un sistema en grupo, el contar con un almacenamiento central de la información agiliza el proceso de revisión ya que éste proporciona consistencia y control de estándares. La capacidad que brindan algunas herramientas para la generación de código contribuye a modificar el sistema por medio de las especificaciones en los diagramas más que por cambios directamente al código fuente.

El desarrollo de sistemas es un proceso interactivo, donde los conocimientos deben tener un repositorio común. Las herramientas CASE soportan pasos interactivos al eliminar el fastidio manual de dibujar

diagramas. Como resultado de esto, los analistas pueden repasar y revisar los detalles del sistema con mayor frecuencia y en forma más segura.

1.1.2.3 Generación de código de las herramientas CASE

La generación de código trae consigo beneficios considerables dentro del desarrollo del software una de ellas es que permite acelerar el proceso de desarrollo al automatizar el tedioso proceso de crear diagramas y posteriormente codificarlos. Otra ventaja es que asegura una estructura consistente (lo que ayuda en la fase de mantenimiento) y disminuye la ocurrencia de varios tipos de errores, mejorando de esta manera la calidad del software desarrollado. Además permite la reutilización, ya que se puede volver a utilizar el software y las estructuras estándares para crear otros sistemas o modificar el actual, así como el cambio de una especificación modular: es decir, volver a generar el código y los enlaces con otros módulos.

Cabe resaltar que ninguna de las herramientas que existen actualmente en el mercado es capaz de generar un código 100% completo, en el sentido de que pueden crear una estructura estática y partes de una estructura dinámica de un sistema, aún no son capaces de generar todo el código que un desarrollador tiene que elaborar a lo largo del ciclo de desarrollo. Cubrir todo este ciclo es precisamente una de las tendencia de las herramientas CASE.

1.1.2.4 Reducción de costos

En ocasiones se desea desarrollar el diseño de pantallas y reportes con el fin de mostrar cómo quedará la interfaz del usuario, es decir, la distribución y colocación de los datos, títulos, mensajes, etc., además es muy común en las etapas de mantenimiento de un sistema, hacer correcciones al prototipo desarrollado. Todo lo anterior trae como consecuencia un costo en tiempo y dinero al tener que utilizar personal para estas tareas.

La reducción de costos se ve reflejada entonces, al utilizar una herramienta CASE para diseño de interfaces, logrando hacer cambios de interfaz con rapidez. Y usando herramientas con generación de código, el código fuente se puede volver a generar después de hacer las modificaciones requeridas rápidamente en los diagramas.

1.1.2.5 Incremento de la productividad

El incremento en la productividad se ve reflejado por la optimización de los recursos con que se cuentan para llevar a cabo un proceso. Al hacer uso de una herramienta CASE se administran los recursos humanos y tecnológicos, y al combinar el conocimiento de los desarrolladores con las prestaciones que brindan estas herramientas es que se obtiene un incremento en la productividad en el desarrollo del software.

1.1.3 Desventajas de las herramientas CASE

Pero todo en las herramientas CASE no es favorable, estas también traen consigo algunos factores de debilidad que van desde el nivel de confiabilidad de los métodos propuestos por estas hasta la poca profundidad en sus límites y alcances.

Como se comenta en la introducción una de las desventajas de las herramientas CASE es **la preparación del personal**. Al adquirir una herramienta CASE es común que se requiera invertir tiempo y en ocasiones dinero para el entrenamiento del personal para obtener un mejor aprovechamiento de la nueva herramienta, esto implica aprender el lenguaje de modelado que la herramienta soporta y la aplicación del proceso de desarrollo a seguir para la construcción de un sistema. A la desventaja anterior se suma que muchas de estas herramientas, están construidas teniendo como base alguna metodología de análisis estructurado y un proceso de desarrollo de sistemas. Esto trae como limitante, que no todos los desarrolladores implicados en el desarrollo de un sistema en grupo están de acuerdo con el empleo de cierta metodología y/o proceso de desarrollo que seleccionó la herramienta. Esto obliga a acordar la metodología a seguir antes de que comiencen a aparecer complicaciones.

La falta de estandarización entre la información que se intercambia, es una de las desventajas más significativas que tienen las herramientas CASE desde que aumentaron en número. Aún no aparece un conjunto compatible de herramientas CASE, es decir, una herramienta puede dar soporte a los diagramas que emplea cierta metodología o bien imponer su propia metodología, reglas y procesos.

A pesar de que las herramientas pueden apoyar muchas fases del ciclo de desarrollo o adaptarse a diferentes métodos de desarrollo, por lo general tienen un enfoque principal hacia una fase o método específico. Esto hace que las herramientas CASE tengan una limitación de sus funciones; por ejemplo este proyecto de tesis no pretende solucionar lo que hemos planteado en el análisis, centra su propósito

en el cumplimiento de los objetivos que respaldan las nuevas ideas para la generación de código automática.

1.1.4 Clasificación de las Herramientas CASE

De manera análoga a como se comenta en la parte introductoria del trabajo, no existe una única clasificación de herramientas CASE y, en ocasiones, es difícil incluirlas en una clase determinada, sin embargo de acuerdo a las posibilidades que brindan podrían clasificarse atendiendo a las plataformas que soportan, las fases del ciclo de vida de desarrollo de sistemas que cubren, la arquitectura de las aplicaciones que producen y su funcionalidad.

Teniendo en cuenta las fases del ciclo de vida abarcada, se pueden agrupar las herramientas CASE de la forma siguiente:

Herramientas integradas

I-CASE (Integrated CASE, CASE integrado), abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench.

El I-CASE se concibe como el conjunto de cuatro herramientas que tocan las disciplinas que van desde la estrategia de la empresa, y la concepción del sistema de información, hasta el análisis, diseño y la generación de los mismos programas.

Tienen la ventaja de integrar el ciclo de vida. Permiten lograr importantes mejoras de productividad a mediano plazo. Este tipo de CASE garantizan un eficiente soporte al mantenimiento de sistemas; a la vez que mantienen la consistencia de sistemas a nivel corporativo.

Como se describe en las desventajas de las herramientas CASE, las I-CASE se basan en una metodología. Además, tienen un repositorio y suman técnicas para todo el ciclo de vida.

En la práctica, estas herramientas no son tan eficientes para soluciones simples, sino para soluciones complejas. Depende del Hardware y del Software, además de ser costosas.

Herramientas de alto nivel

U-CASE (Upper CASE - CASE superior) o front-end, orientadas a la automatización y soporte de las actividades desarrolladas durante las primeras fases del desarrollo: análisis y diseño.

Este tipo de CASE se utiliza en plataforma PC, son aplicables a varios entornos y su costo es menor, sin embargo aunque permite mejorar la calidad de los sistemas, no mejora la productividad. Esta herramienta no logra integrarse al ciclo de vida.

Herramientas de bajo nivel

L-CASE (Lower CASE - CASE inferior) o back-end, dirigidas a las últimas fases del desarrollo: construcción e implantación.

Las L-CASE mejoran la productividad a corto plazo, y el soporte al mantenimiento de sistema es eficiente. Pero no garantizan la consistencia de los resultados a nivel corporativo, no logra la eficiencia en el análisis y el diseño, además de no integrar el ciclo de vida del software.

Juegos de herramientas o Tools-Case

Son el tipo más simple de herramientas CASE. Automatizan una fase dentro del ciclo de vida. Dentro de este grupo se encontrarían las herramientas de reingeniería, orientadas a la fase de mantenimiento.

Las **Tools-Case** las forman un conjunto de herramientas orientadas cada una de ellas a resolver una determinada fase de desarrollo. Las interfaces de estas herramientas logran, el ensamblaje que adaptan las salidas producidas por cada una de forma que sirva como entrada en la siguiente. Algunas de estas herramientas las citaremos a continuación.

Para el **análisis y diseño,** se utilizan las Tools CASE: Prokit Workbench de McDonnell-Douglas, Desing Aid Nastec, Analyst/Designer Tolkit de Yourdon, Excelerator de Index Technology y Pose de Computer System Advisers. En el **diseño de archivos y bases de datos** resaltan IDMS/Architec de Cullinet Software, Autmate Plus de LBMS, Case Designer de Oracle, Synon, Oracle e Informix. **En la programación** es importante mencionar a las siguientes herramientas: APS de Sage Software, Tranforms de Transform Logic, Telon Pansophic System, Decase de DEC COBOL 2/ Worbench de Micro Focus y Snap CASE para AS/400. [4]

1.1.5 Clasificación atendiendo a la funcionalidad

Herramientas de planificación de sistemas de gestión

Sirven para convertir en modelos los requisitos de información estratégica de una organización. Brindan un metamodelo para obtener sistemas de información específico. Tiene como principal objetivo una mejor comprensión de la información que circula entre las distintas unidades organizativas. Proporcionan una ayuda importante cuando se diseñan nuevas estrategias para los sistemas de información y cuando los métodos y sistemas actuales no satisfacen las necesidades de la organización.

1.1.5.1 Herramientas de análisis y diseño

Son las CASE que permiten al desarrollador crear un modelo del sistema que se va a construir y les permite evaluar su consistencia y validez. Esta característica, proporciona un grado de confianza en la representación del análisis y ayudan a eliminar errores con anticipación. Como ejemplo de estas se pueden mencionar:

- Herramientas de análisis y diseño (Modelamiento).
- Herramientas de creación de prototipos y de simulación.
- Herramientas para el diseño y desarrollo de interfaces.
- Máquinas de análisis y diseño (Modelamiento).

1.1.5.2 Herramientas de programación

Se engloban aquí los compiladores, los editores y los depuradores de los lenguajes de programación convencionales.

- Herramientas de codificación convencionales.
- Herramientas de codificación de cuarta generación.
- Herramientas de codificación orientada a objetos.

1.1.5.3 Otras Herramientas

Herramientas de integración y prueba

Sirven de ayuda a la adquisición, medición, simulación y prueba de los equipos lógicos desarrollados.

Herramientas de gestión de prototipos

Los prototipos son utilizados ampliamente en el desarrollo de aplicaciones, para la evaluación de especificaciones de un sistema de información, o para un mejor entendimiento de cómo los requisitos de un sistema de información se ajustan a los objetivos perseguidos.

Herramientas de mantenimiento

Herramientas de ingeniería inversa, herramientas de reestructuración y análisis de código, herramientas de reingeniería

Herramientas de gestión de proyectos

La mayoría de las herramientas CASE de gestión de proyectos se centran en un elemento específico de la gestión del proyecto, en lugar de proporcionar un soporte global para la actividad de gestión. Utilizando un conjunto seleccionado de las mismas se puede: realizar estimaciones de esfuerzo, coste y duración, hacer un seguimiento continuo del proyecto, estimar la productividad y la calidad, etc. Existen también herramientas que permiten al comprador del desarrollo de un sistema, hacer un seguimiento que va desde los requisitos del pliego de prescripciones técnicas inicial, hasta el trabajo de desarrollo que convierte estos requisitos en un producto final.

Herramientas de soporte

Se engloban en esta categoría las herramientas que recogen las actividades aplicables en todo el proceso de desarrollo. Se incluyen dentro de estas herramientas, la de documentación, las herramientas para software de sistemas, las del control de la calidad y las herramientas de Base de Datos.

Reingeniería en las CASE

Los sistemas Case permiten establecer una relación estrecha y fuertemente formal entre los productos generados a lo largo de distintas fases del ciclo de vida, permitiendo actuar en el sentido especificaciones-código (ingeniería "directa") y también en el contrario (ingeniería "inversa"). Ello facilita la realización de modificaciones en la fase más adecuada en cada caso y su traslado a las demás. Al conjunto de

facilidades proporcionadas por la ingeniería «directa» e "inversa" se le denomina "re-ingeniería". Específicamente en este proyecto la reingeniería será una funcionalidad explotada para la materialización del proyecto.

Repositorio en los Sistemas CASE

Funcionan en torno a un repositorio central, siendo éste el núcleo fundamental que contiene todas las definiciones de objeto y sus relaciones. Los objetos pueden ser especificaciones del sistema en forma de diagramas de flujo de datos, diagramas entidad-relación, esquemas de bases de datos, diseños de pantallas, etc. El repositorio es un concepto más amplio que el de diccionario de datos y soporta a los demás grupos de funciones. No es fácil encontrar en el mercado productos CASE con funcionalidades estrictamente a las de repositorio, ya que, a pesar de su innegable importancia, tienen un carácter auxiliar de los demás grupos de funciones. Cualquier sistema Case poseerá un repositorio propio o bien, trabajará sobre un repositorio suministrado por otro fabricante o vendedor.

1.1.6 Sistemas CASE y el ciclo de desarrollo

Tanto para una aplicación, como para un sistema de información, el ciclo de vida abarca desde la planificación de su desarrollo hasta su implementación, mantenimiento y actualización.

Los sistemas CASE pueden cubrir la totalidad de estas fases o bien especializarse en algunas de ellas, teniendo en cuenta las diferentes metodologías que se pueden usar. En el caso en que se especifiquen solo algunas de las fases, se pueden distinguir sistemas de "alto nivel" ("Upper Case"), orientados a la autonomía y soporte de las actividades correspondientes a las dos primeras fases y, sistemas de "bajo nivel" ("Lower Case"), dirigidos hacia las dos últimas. Los sistemas de "alto nivel" pueden soportar un número más o menos amplio de metodologías de desarrollo, como hemos comentado en los epígrafes anteriores.

1.1.6.1 El soporte del proyecto

Constituye el soporte de actividades que se producen durante el desarrollo, derivadas fundamentalmente del trabajo en grupos, como ejemplos tenemos las facilidades de comunicación, soporte a la creación, modificación e intercambio de documentación, actualización, herramientas personales, controles de

seguridad, entre otras. Los sistemas CASE pueden conceder a estas cuestiones una importancia variable por lo cual el soporte de proyecto constituye un factor de diferenciación entre ellas.

1.1.7 Composición de una herramienta CASE

1.1.7.1 Módulos de diagramas y modelos

Algunos de los diagramas y modelos utilizados son:

- Diagrama de flujo de datos.
- Modelo entidad - interrelación.
- Historia de la vida de las entidades.
- Diagrama Estructura de datos.
- Diagrama Estructura de cuadros.
- Técnicas matriciales.

Algunas características referentes a los diagramas son:

- Número máximo de niveles para poder soportar diseños complejos.
- Número máximo de objetos que se pueden incluir para no encontrarse limitado en el diseño de grandes aplicaciones.
- Número de diagramas distintos en pantalla o al mismo tiempo en diferentes ventanas.
- Dibujos en formato libre con la finalidad de añadir comentarios, dibujos, información adicional para aclarar algún punto concreto del diseño.
- Actualización del repositorio por cambios en los diagramas. Siempre resulta más fácil modificar de forma gráfica un diseño y que los cambios queden reflejados en el repositorio.
- Control sobre el tamaño, fuente y emplazamiento de los textos en el diagrama.
- Comparaciones entre gráficos de distintas versiones. De esta forma será más fácil identificar qué diferencias existen entre las versiones.
- Inclusión de pseudocódigo, que servirá de base a los programadores para completar el desarrollo de la aplicación.
- Posibilidad de deshacer el último cambio, facilitando que un error no conlleve perder el trabajo realizado.

1.1.7.2 Repositorio

El repositorio es la base de datos central de una herramienta CASE. Amplía el concepto de diccionario de datos para incluir toda la información que se va generando a lo largo del ciclo de vida del sistema, como por ejemplo: componentes de análisis y diseño (diagramas de flujo de datos, diagramas entidad-relación, esquemas de bases de datos, diseños de pantallas), estructuras de programas, algoritmos, etc. En algunas referencias se le denomina Diccionario de Recursos de Información.

La mayoría de las herramientas CASE poseen un repositorio propio o bien trabajan sobre un repositorio suministrado por otro fabricante o vendedor. Apoyándose en la existencia del repositorio se efectúan comprobaciones de integridad y consistencia para que no existan datos no definidos, no existan datos autodefinidos (datos que se emplean en una definición pero que no han sido definidos previamente). Que todos los alias (referencias a un mismo dato empleando nombres distintos) sean correctos y estén actualizados.

Características del repositorio:

- Tipo de información: Que contiene alguna metodología concreta, datos, gráficos, procesos, informes, modelos o reglas.
- Tipo de controles: Si incorpora algún módulo de gestión de cambios, de mantenimiento de versiones, de acceso por clave, de redundancia de la información. Realizar cambios en la versión en producción y en la de desarrollo, simultáneamente.
- Tipo de actualización: Si los cambios en los elementos de análisis o diseño se ven reflejados en el repositorio en tiempo real o mediante un proceso por lotes (batch). Esto será importante en función a la necesidad de que los cambios sean visibles por todos los usuarios, en el acto.
- Reutilización de módulos para otros diseños: El repositorio es la clave para identificar, localizar y extraer código para su reutilización.
- Posibilidad de exportación e importación para extraer información del repositorio y tratarla con otra herramienta (formateo de documentos, mejora de presentación) o incorporar al repositorio, información generada por otros medios.
- Interfaces automáticas con otros repositorios o bases de datos externos.

1.1.7.3 Herramienta de prototipado

Esta herramienta tiene como objetivo mostrar información al usuario, desde los momentos iniciales del diseño, el aspecto que tendrá la aplicación una vez desarrollada en su totalidad. Ello facilitará la aplicación de los cambios que se consideren necesarios, todavía en la fase de diseño. La herramienta será tanto más útil, cuanto más rápidamente permita la construcción del prototipo y por tanto antes, se consiga la implicación del usuario final en el diseño de la aplicación. Asimismo, es importante poder aprovechar como base el prototipo para la construcción del resto de la aplicación. Actualmente, es imprescindible utilizar productos que incorporen esta funcionalidad por la cambiante tecnología y necesidades de los usuarios. Los prototipos han sido utilizados ampliamente en el desarrollo de sistemas tradicionales, ya que proporcionan una realimentación inmediata, que ayudan a determinar los requisitos del sistema. Las herramientas CASE están bien dotadas, en general, para crear prototipos con rapidez y seguridad.

1.1.7.4 Analizador de sintaxis

El analizador de sintaxis verifica la validez y el correcto uso de la información introducida a la herramienta de acuerdo a la metodología o lenguaje soportado por la herramienta. Este componente es muy importante para ayudar al desarrollador a no cometer errores de sintaxis al momento de crear los diagramas y generar un código más completo y preciso.

1.1.7.5 Generador de código

Para evitar los problemas que puede traer pasar el código generado de un host a otro, la generación se suele utilizar sobre ordenadores personales o una estación de trabajo específica. Esto evita los problemas de compilación en ambos entornos.

Características de los generadores de código

- Lenguaje generado: Si se trata de un lenguaje estándar o un lenguaje propietario.
- Portabilidad del código generado: Capacidad para poder ejecutarlo en diferentes plataformas físicas y/o lógicas.
- Generación del esqueleto del programa o del programa completo: Si únicamente genera el esqueleto será necesario completar el resto mediante programación.

- Posibilidad de modificación del código generado: Suele ser necesario acceder directamente al código generado para optimizarlo o completarlo.
- Generación del código asociado a las pantallas e informes de la aplicación: Mediante esta característica se obtendrá la interfaz de usuario de la aplicación.

1.1.7.6 Módulo generador de documentación

El módulo generador de la documentación se alimenta del repositorio para transcribir las especificaciones allí contenidas.

Los generadores, permiten la generación automática a partir de los datos del repositorio, sin necesidad de un esfuerzo adicional. Permiten la combinación textual y gráfica para facilitar su comprensión. Generan referencias cruzadas. Con ello se podrá localizar fácilmente en qué partes de la aplicación se encuentra un determinado objeto o elemento, con el fin de analizar el impacto de un cambio o identificar los módulos afectados por un determinado error. Facilitan la introducción de textos complementarios para la documentación generada de forma automática. Tienen interfaces con otras herramientas como los procesadores de textos, editores gráficos, etc.

1.1.8 Ejemplos de Herramientas CASE

1.1.8.1 Herramientas basadas en modelos

Rational Rose

Es una herramienta de software para el Modelado Visual mediante UML de sistemas software. Permite especificar, analizar y diseñar el sistema antes de Codificarlo. Es la herramienta CASE de referencia para la especificación gráfica con notación UML. Las herramientas de Rational Corporation (Corporacion Rational), intentan abarcar todo el ciclo de vida del desarrollo, para dar soporte a su metodología RUP (Rational Unified Proces). Sin embargo la generación de código de esta herramienta carece como la mayoría de los casos de especificaciones en la capa de negocio durante la obtención automática de código fuente.

Argo UML

Es una herramienta implementada en java y con licencia de código abierto, permite modelar usando UML. Precisamente esta apertura de código permite la creación de módulos adicionales para la generación de código. La mayoría de las funciones soportan la selección múltiple de los elementos del modelo .Permite además arrastrar y soltar desde el árbol de exploración al diagrama y dentro del árbol de exploración. Es una herramienta construida en diseños críticos suministra una revisión no obstructiva del diseño y sugerencias para mejoras. Presenta un interfaz de módulos extensible. Soporte de Internacionalización para inglés, alemán, francés, español y ruso. Genera ficheros PNG, GIF, JPG, SVG, EPS desde diagramas. Sin embargo, los modelos a veces no pueden ser reabiertos, no se puede aplicar las llamadas reflexivas en los diagramas de secuencia; y para crear un diagrama de secuencia debes seleccionar una clase. Durante la generación de código una limitante es la codificación de las reglas del negocio, la redefinición del código generado en ocasiones se hace molesta para los desarrolladores. [5]

ERwin

La herramienta PLATINUM ERwin está creada para el diseño de base de datos. Brinda productividad en diseño, generación, y mantenimiento de aplicaciones. Desde un modelo lógico de los requerimientos de información, hasta el modelo físico perfeccionado para las características específicas de la base de datos diseñada, ERwin permite visualizar la estructura, los elementos importantes, y optimizar el diseño de la base de datos. Genera automáticamente las tablas y miles de líneas de *stored procedure* y *triggers* para los principales tipos de base de datos. ERwin hace fácil el diseño de una base de datos. Los diseñadores de bases de datos sólo apuntan y pulsan un botón para crear un gráfico del modelo E-R (Entidad-relación) de todos sus requerimientos de datos. Más que una herramienta de dibujo, ERwin automatiza el proceso de diseño de una manera inteligente. Se mantienen las vistas de la base de datos como componentes integrados al modelo, permitiendo que los cambios en las tablas sean reflejados automáticamente en las vistas definidas. La migración automática garantiza la integridad referencial de la base de datos.

Esta herramienta CASE establece una conexión entre una base de datos diseñada y una base de datos, permitiendo transferencia entre ambas y la aplicación de ingeniería reversa. Usando esta conexión, ERwin genera automáticamente tablas, vistas, índices, reglas de integridad referencial (llaves primarias, llaves foraneas), valores por defecto y restricciones de campos y dominios.

Es una de las herramientas más usadas por los diseñadores de bases de datos en muchísimas organizaciones. Soporta principalmente bases de datos relacionales SQL y bases de datos que incluyen

Oracle, Microsoft SQL Server, Sybase, DB2, e Informix. El mismo modelo puede ser usado para generar múltiples bases de datos, o convertir una aplicación de una plataforma de base de datos a otra.

Borland Together

Es un entorno de desarrollo de aplicaciones que abarca el diseño de aplicaciones, el desarrollo y el despliegue, para que los equipos tengan flexibilidad para planear, construir y distribuir sistemas robustos. Algunos productos son:

Borland® Together® ControlCenter® and Borland® Together® Solo

Borland® Together® Edition for WebSphere® Studio

Borland® Together® Edition for JBuilder®

Borland® Together® Edition for Eclipse

Borland® Together® Edition for Microsoft® Visual Studio® .NET

Borland Together acelera el desarrollo con un modelado ágil. Ahorra tiempo y mejora las comunicaciones LiveSource (Control de Versiones). Supervisa la calidad durante todo el desarrollo con auditorias y toma de medidas de parámetros. Aprovecha las ventajas de la automatización para mejorar la productividad de los equipos de trabajo. Abrevia el ciclo de vida de las aplicaciones con numerosas interacciones. Sin embargo sus requerimientos de hardware exigen elevadas capacidades de memoria, almacenamiento, video y velocidad del CPU.

Together va generando el código a medida que avanzamos en el diagrama. Basta hacer doble clic en uno de los objetos para que en el panel inferior se vea su código fuente asociado. Aunque al igual que en las herramientas anteriores las especificaciones de las reglas del negocio para clases de formularios y sus componentes, no se expresan durante la generación de código fuente. [6]

System Architect

Esta posee un repositorio único que integra todas las herramientas, y metodologías usadas. En la elaboración de los diagramas, el System Architect conecta directamente al diccionario de datos, los elementos asociados comentarios, reglas de validaciones, normalización, etc. Posee control automático

de diagramas y datos normalizaciones y balanceamiento entre diagramas "Padre e Hijo", además de balanceamiento horizontal, que trabaja integrado con el diccionario de datos, asegurando la compatibilidad entre el Modelo de Datos y el Modelo Funcional.

System Architect es considerado un Upper Case, que puede ser integrado a la mayoría de los generadores de código. Traduce modelos de entidades, a partir de la enciclopedia, en esquemas para Sybase, DB2, Oracle u Oracle 7, Ingress, SQL Server, RDB, XDB, Progress, Paradox, SQL Base, AS400, Interbase, OS/2, DBMS, Dbase 111, Informix, entre otros. Genera también Windows DDL, definiciones de datos para lenguaje C/C++ y estructuras de datos en Cobol. En esta última versión del System Architect es posible a través de ODBC, la creación de bases de datos a partir del modelo de entidades, para los diversos manejadores de bases de datos arriba mencionados.

Posee esquemas de seguridad e integridad a través de contraseñas que posibilitan el acceso al sistema en diversos niveles, así se puede integrar a la seguridad de la red Novell o Windows/NT de ser necesario. Posee también un completo Help (ayuda) sensible al contexto.

System Architect posee un módulo específico para Ingeniería Reversa desde las Bases de Datos SQL más populares, incluyendo Sybase, DB2, Infonmix, Oracle y SQL Server (DLL), además de diálogos (DLG) y menús (MNU) desde Windows. La Ingeniería Reversa posibilita la creación, actualización y mantención, tanto del modelo lógico como de su documentación. El System Architect logra leer bases de datos y construir el modelo lógico o físico (diagrama), alimentando su diccionario de datos con las especificaciones de las tablas y de sus elementos de datos, incluyendo las relaciones entre tablas y su cardinalidad.

Cool:plex

Es una herramienta comercial, cuyo fin es diseñar aplicaciones cliente servidor, sin importar la plataforma.Su funcionamiento se basa en diagramas de entidad-relación extendidos. Usa patrones de diseño que son traducidos en la fase de generación a un lenguaje destino en particular.Sin embargo, la generacion de código no sobrepasa las posibilidades de la sherramientas descritas hasta el momento. [7]

eZway

Es una herramienta de desarrollo de aplicaciones ligada a plataformas y tecnologías de Microsoft. Dentro de la definición de sus funciones se puede apreciar la creación de diagramas de casos de uso, dibujados usando la herramienta VISIO.Permite la especificación de los casos de uso, a través de una tabla

textual.Con el uso de esta herramienta se pueden definir formularios, mediante el diseño de los mismos para Visual Basic. La generación de código ocurre a partir de los objetos definidos en cada capa, sin refinamientos en la capa del negocio.

eZway está muy ligado a la plataforma Microsoft. Emplea Visual Basic y objetos ActiveX en el proceso de diseño, lo que dificulta la migración del proyecto a otras plataformas como pueda ser Java.Es una herramienta dependiente de la tecnología Microsoft desde el punto de vista de la interfaz de usuario;esta característica impide su fácil migración a otros entornos.

1.2 Lenguaje de Modelado Unificado (UML)

El UML(Unified Modeling Language) está considerado como el lenguaje de modelado gráfico y visual por exelencia,utilizado para especificar, visualizar, construir y documentar los componentes de un sistema de software. Está pensado para poder aplicarse en cualquier medio que necesite capturar requerimientos y comportamientos del sistema que se desee construir. Ayuda a comprender y a mantener de una mejor forma un sistema basado en un área que el analista o desarrollador puede desconocer.Como se explica en la introducción de la presente tesis,UML surgió con el propósito de lograr una estandarización universal al crecido número de metodologías de desarrollo que habían estado apareciendo.

Este lenguaje permite captar información sobre la estructura estática y dinámica de un sistema, en donde la estructura estática proporciona información sobre los objetos que intervienen en determinado proceso y las relaciones que existen entre ellos, y la estructura dinámica define el comportamiento de los objetos a lo largo de todo el tiempo que estos interactúan hasta llegar a su o sus objetivos. [8]

Una característica sobresaliente del UML es que no es un método, sino un lenguaje de modelado. Un método define su notación (lenguaje) y su proceso a seguir durante el ciclo de vida de desarrollo del software. Se encarga de la notación gráfica y su significado, a partir de la cual se crearán los diseños de sistemas y no depende de un proceso, el cual sería el encargado de orientar los pasos a seguir para elaborar el diseño. Mejorar la comunicación es la idea a seguir con la creación de los diagramas mediante UML, sin dudas ayuda a que los desarrolladores de software se comuniquen con un mismo lenguaje de modelado independiente de las metodologías empleadas.

Por esta razón la mayoría de las fuentes coinciden en que la principal ventaja de UML sobre otras notaciones OO es que elimina las diferencias entre semánticas y notaciones.

1.2.1 Precedentes de UML

Antes de la creación del UML se realizaron varios intentos de creación de un lenguaje para propósitos similares. El más conocido es Fusion, que incluyó conceptos de los métodos OMT y Booch; este intento fue tomado como otro método debido a que los autores de estos dos últimos métodos no lograron involucrarse. El primer paso para lograr una unificación y acercarse a UML lo dió Jim Rumbaugh en el 1994, al unirse a Grady Booch en Rational Software Corporation con la finalidad de unificar sus métodos OMT y Booch respectivamente.

Un año más tarde salió la primera propuesta de su método integrado que fue la versión 0.8 del entonces llamado Método Unificado (Unified Method).Ivar Jacobson se une a Rational, trabaja con Booch y Rumbaugh en el proceso de unificación.En el año 1996 estos tres científicos nombran al resultado de su trabajo unificado UML, a partir de este momento el OMG(*Object Management Group* – Grupo Administrador de Objetos) decide convocar a otras compañías a participar con sus propuestas para mejorar el enfoque estándar que el UML pretendía.

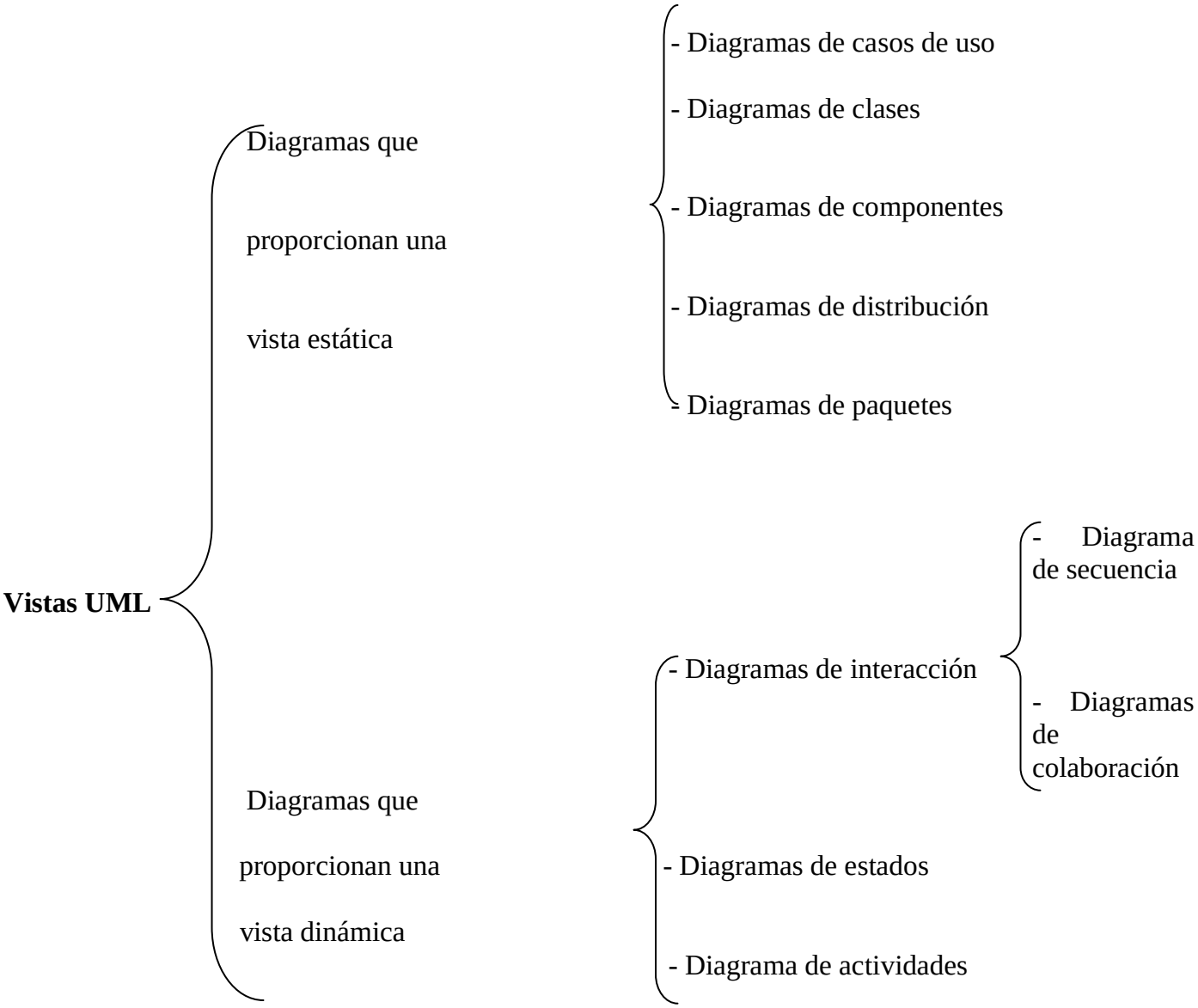
En enero de 1997, todas las propuestas de las empresas – entre las que figuran IBM, Oracle y Rational Software – se unieron en la versión 1.0 del UML que fue presentada ante el OMG para su consideración como estándar. Y finalmente en Noviembre de 1997 el UML fue adoptado por el OMG y otras organizaciones a fines como lenguaje de modelado estándar,hasta que a finales del 2002 IBM adquirió las acciones de Rational Software Corporation.[9]

1.2.2 Conceptos de UML

Hasta aquí se entiende que UML no es un método sino un lenguaje, el cual define únicamente una notación y un metamodelo.Como UML es un lenguaje estándar no depende de un proceso de desarrollo, logra unificar los métodos encontrando un lenguaje común entre los métodos diferentes, para que después en el desarrollo del proyecto el desarrollador escogiera la metodología que le sea más interesante utilizar.con UML, los ingenieros de software pueden tener la seguridad de que estarán creando diseños de software bajo un lenguaje que será comprendido por todos aquellos que utilicen el UML.

1.2.3 Vistas UML

Las vistas de UML se refieren a la forma en que se modela una parte del sistema a desarrollar. Los autores del UML proponen una clasificación de los diagramas que proporcionan las vistas de UML, y aunque pareciera que esta clasificación es algo intuitiva, aclaran que es simplemente una propuesta y que cada desarrollador puede crear su propia clasificación. [10]



Vistas de UML (Clasificación simple)

La vista estática es la representación de los elementos del sistema y sus relaciones, la vista dinámica muestra la especificación y la implementación del comportamiento a lo largo del tiempo, es decir muestran el cambio progresivo de los objetos . [9]

UML tiene la responsabilidad de representar graficamente la organización del Ciclo de Vida del proceso de desarrollo del Software. El importante apoyo de UML permite que diferentes empresas creen software en trabajo conjunto a lo largo de y ancho del mundo.

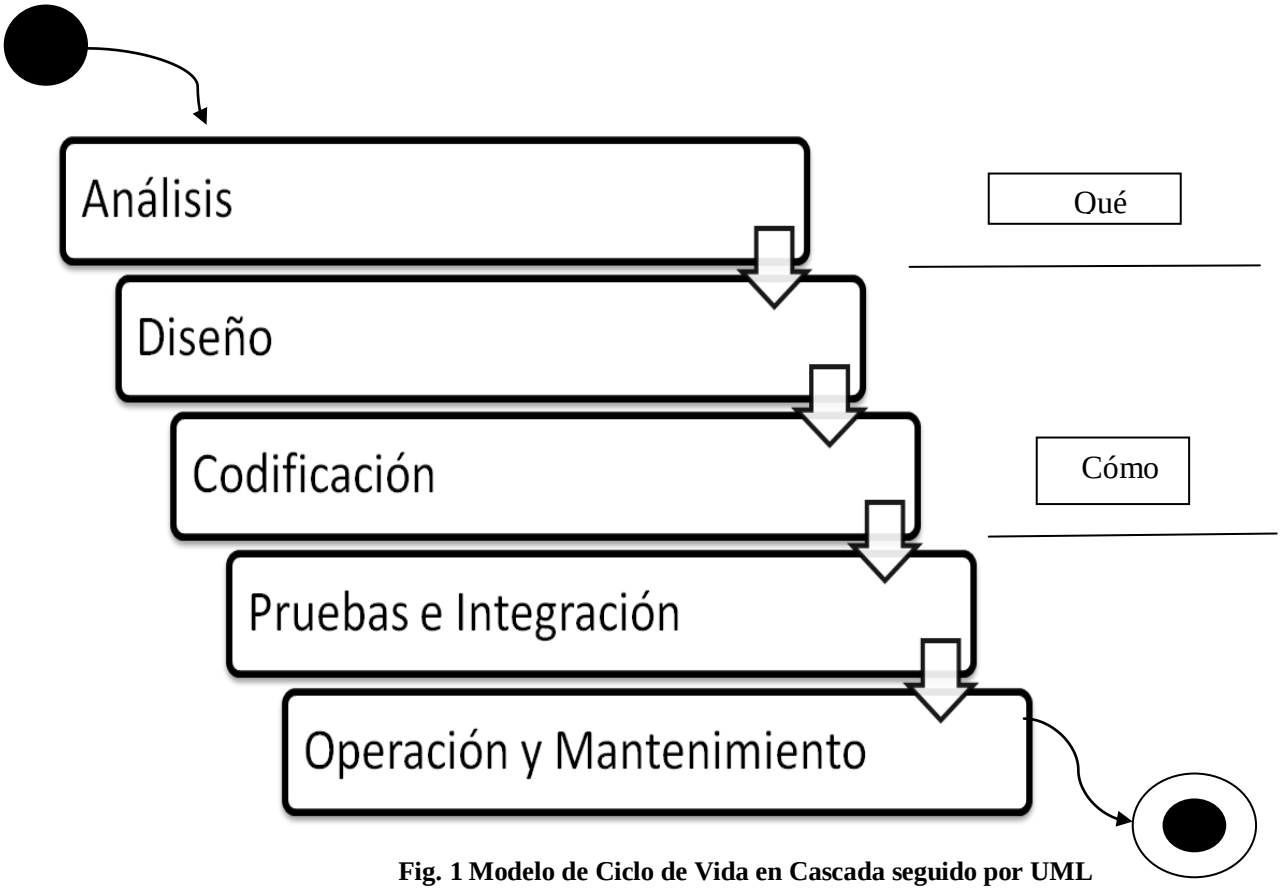


Fig. 1 Modelo de Ciclo de Vida en Cascada seguido por UML

1.2.4 Digramas UML

Diagramas de Casos de Uso.

Los diagramas de casos de uso, modelan la forma en cómo un actor interactúa con el sistema, es decir, describe una interacción entre uno o más actores y el sistema o subsistemas como una secuencia de mensajes que llevan una forma, tipo y orden. El propósito de esta vista es enumerar a los actores y los casos de uso, mostrando un comportamiento y determinar qué actores participan en cada caso de uso.

La vista de casos de uso es útil para tener una forma de comunicarse con los usuarios finales del sistema, ya que da una visión de cómo ellos esperan que el sistema se comporte.

Un diagrama de casos de uso es una descripción lógica de una parte funcional del sistema y no del sistema en su totalidad. Este diagrama consta de elementos estructurales y relaciones.

Actor

Un actor es un rol que un usuario juega cuando interacciona con el sistema, es un comportamiento específico frente a tal sistema. Un actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza ante al sistema, puede ser un humano, otro sistema de software o algún otro proceso. Se puede decir entonces que, varios usuarios pueden estar relacionados con un solo actor.

Caso de uso

Un caso de uso es una acción o tarea específica que el sistema lleva a cabo tras una petición ya sea de un actor o de otro caso de uso. Un caso de uso conduce a un estado observable de interés para un actor. Se denota como una elipse con su nombre dentro o debajo de ella.

Relaciones

Permiten la conexión entre los elementos estructurales para darle sentido al diagrama de casos de uso.

Asociación

La asociación es sin dudas la relación más simple del UML, es la comunicación que se da entre un actor y un caso de uso, o entre dos casos de uso. Se denota por una línea dirigida entre ellos.

Generalización

Este tipo de relación se da entre un caso de uso general y un caso de uso más específico. Donde este último hereda propiedades del primero y agrega sus propias acciones. Se denota como una línea continua con una punta de flecha en forma de triángulo sin rellenar que apunta hacia el caso de uso general.

Inclusión

Es el tipo de relación que se da entre casos de uso cuando se tiene una parte de comportamiento que es similar en más de un caso de uso, y no se desea copiar la descripción de tal conducta para cada uno de ellos o bien cuando un uso necesita utilizar a otro caso de uso. Es recomendable utilizar la inclusión .

Extensión

Esta relación se da también sólo en casos de uso cuando se tiene un caso de uso que es similar a otro, pero que hace un poco más. Es recomendable entonces, utilizar la extensión cuando se describa una variación de una conducta normal. Se denota con una línea discontinua con una punta de flecha y con la palabra clave “extend”.

Diagrama de Clases

La vista de los diagramas de clase, visualiza las relaciones entre las clases que se involucran en el sistema. Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones y las relaciones entre éstas, mostrando así, la estructura estática de un sistema. Los diagramas de clase pueden mostrar:

- Clases
- Atributos
- Operaciones
- Asociaciones
- Generalizaciones
- Agregaciones
- Composiciones

Clase

Es la unidad básica que encapsula toda la información que comparten los objetos del mismo tipo. A través de una clase se puede modelar el entorno del sistema. En UML, una clase se representa por un

rectángulo que posee en su primera división el nombre de la clase, en la división intermedia se presentan los atributos que caracterizan a la clase que pueden ser *private, protected o public*.

La tercera división del rectángulo Contiene las operaciones, las cuales son la forma de cómo interactúa el objeto con los demás, dependiendo de la visibilidad pueden ser *private, protected o public*.

Atributos

Los atributos o características de una Clase se utilizan para almacenar información, estos atributos tienen asignado un tipo de visibilidad que define el grado de comunicación con el entorno, los tipos de visibilidades son tres, de acuerdo a la Programación Orientada a Objetos: public (pública), protected (protegida), private (privada). La sintaxis en UML para un atributo es: *visibilidad nombre : tipo = valor_inicial.*

Operaciones

Las operaciones de una clase son la forma en cómo ésta interactúa con su entorno, éstas también pueden tener uno de los tipos de visibilidad arriba mencionadas. La sintaxis en UML para una operación es: *visibilidad nombre (parámetros): tipo_de_retorno.*

Asociación

La relación entre las instancias de las clases es conocida como Asociación, permite relacionar objetos que colaboran entre sí. La asociación puede ser unidireccional o bidireccional. Una asociación unidireccional significa que sólo existe comunicación de la clase de la que parte la flecha hacia la que apunta. En caso de que sea bidireccional significa que existe comunicación entre ambas clases. Para representar una asociación bidireccional sólo se dibuja una línea continua que una las dos clases.

Agregación

La agregación es un tipo especial de asociación, con la cual se pueden representar entidades formadas por varios componentes. La agregación es una relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye, en la programación OO se dice que ésta es una relación por referencia. La agregación se representa con una flecha con punta de rombo en blanco en el extremo del compuesto o base.

Composición

Esta relación representa entidades formadas por componentes, sólo que esta relación es estática, en donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye, ya que el objeto base se construye a partir del objeto incluido, en programación OO se dice que esta es una relación por valor. La composición se representa con una flecha con punta de rombo relleno en el extremo del compuesto o base.

Multiplicidad

Cada asociación tiene dos roles o papeles que se encuentran en cada extremo de la línea, cada uno de estos papeles tiene asignada una multiplicidad, la cual indica el número de objetos que pueden participar en la relación, es decir los límites inferior y superior de los objetos que participan. La multiplicidad puede ser:

- 1..* (1..n) Uno o más
- (n) Cero o más
- m (m es un entero) Número fijo

Estos números se colocan sobre la línea de asociación junto a la clase correspondiente, como se ilustra.

Generalización (Herencia)

De manera análoga a como se explica en los conceptos de diagramas de casos de usos, la generalización o herencia entre clases; indica que una subclase hereda las operaciones y atributos especificados por una superclase, por ende, la subclase además de poseer sus propios métodos y atributos, poseerá las operaciones y atributos de la superclase, siempre y cuando estos tengan visibilidad pública o protegida. Se denota como una línea continua con una punta de flecha en forma de triángulo sin rellenar que apunta hacia la superclase.

Dependencia

La dependencia es un tipo de relación entre dos elementos, donde el uso más particular de este tipo de relación es para denotar una dependencia de uso que tiene una clase con otra, en otras palabras, un cambio en una de ellas causa un cambio en la otra. En la dependencia el objeto utilizado no se almacena dentro del objeto que la crea. La dependencia se denota como una línea discontinua con punta de flecha formada por dos líneas.

Diagrama de paquetes

Los sistemas grandes es conveniente separarlos en piezas más pequeñas para poder trabajar más fácil con él, y son los diagramas de paquetes los que nos muestran las relaciones y las dependencias entre los diferentes paquetes en que un sistema está dividido.

Paquete

Un paquete es un mecanismo para organizar un conjunto de elementos. Los paquetes pueden contener otros elementos de modelo como diagramas, clases, máquinas de estado, diagramas de casos de uso o diagramas de interacción, incluso puede contener otros paquetes. Comúnmente un paquete representa un subsistema.

Interfaz

Una interfaz es un elemento de modelado que define un conjunto de comportamientos a través de operaciones ofrecidas por un elemento, ya sea una clase, un subsistema u otro componente.

Dependencia

La dependencia entre paquetes se da si existiera una dependencia entre dos clases en diferentes paquetes. La dependencia se denota como una línea discontinua con punta de flecha formada por dos líneas.

Realización

La realización es una relación que se da entre un paquete y una interfaz, lo que indica que el paquete define su comportamiento a través de una o más interfaces. La realización entre un paquete y una interfaz se denota como una línea continua.

Diagrama de interacción

La vista de los diagramas de interacción modela las secuencias de intercambio de mensajes entre objetos. Un diagrama de interacción puede ser de dos tipos: diagrama de secuencia o diagrama de colaboración. El primero se organiza de acuerdo al tiempo y el último de acuerdo al espacio. [8]

Diagrama de secuencia

Un diagrama de secuencia muestra un conjunto de mensajes que se envían de un objeto a otro a través del tiempo. El propósito de estos diagramas es mostrar la secuencia del comportamiento de la especificación de realización de un caso de uso. En el diagrama de secuencia los mensajes corresponden a las operaciones de las clases, es decir, cuando los objetos solicitan la realización de cierta operación a otros objetos o a ellos mismos. El diagrama de secuencia consta de tres elementos principales: objetos, mensajes y el tiempo.

1.2.5 Desventajas de UML

Falta de integración con respecto a otras técnicas como los patrones de diseño, interfaces de usuarios, documentación etc.

1.3 Esferas de la generación de código

1.3.1 Generación de código en .NET Framework utilizando un esquema XML

Tanto en las capas de acceso a datos, clases de entidad comercial o interfaces de usuarios, la generación automática puede aumentar significativamente la productividad de los desarrolladores.

Este proceso de generación podrá basarse en una serie de entradas, como por ejemplo una base de datos, un archivo XML arbitrario, un diagrama UML, etc. Microsoft® Visual Studio ® .NET cuenta con soporte integrado para generación de código desde archivos W3C XML Schema (XSD) en dos formas: datasets tipificados, y clases personalizadas para uso con XmlSerializer.

Los archivos XSD describen el contenido permitido en un documento XML a fin de ser considerado válido según el documento. La necesidad de manejar los datos, que finalmente serán serializados como XML para consumo, en forma de tipos seguros llevó a diversos enfoques para convertir XSD en clases.

XSD no fue creado como medio para describir objetos y sus relaciones. Ya existe un mejor formato para este fin, UML, y es muy utilizado para modelar aplicaciones y generar código a partir de ellas. Por lo tanto, existen algunas (esperadas) inconsistencias entre .NET y sus conceptos de programación orientada a objetos, y los de XSD. Es importante recordar esto cuando se realiza el proceso de XSD , mapeo de clases.

Pero XSD soporta prestaciones que no pueden mapearse a conceptos OO regulares. Por lo tanto, si utiliza XSD sólo para modelar sus clases en lugar de modelar sus documentos, la probabilidad de conflictos será casi nula.

Los datasets tipificados se utilizan cada vez más para representar entidades comerciales; es decir, para actuar como transporte para datos de entidades entre capas de una aplicación, o incluso como salida de Servicios Web. Los datasets tipificados, en comparación con el Dataset "normal", resultan atractivos porque también se obtiene acceso tipificado a tablas, filas, columnas, etc. No obstante, también tienen sus costos y/o limitaciones, como que incluyen un gran número de prestaciones que pueden no ser necesarias para sus entidades, como por ejemplo seguimiento de cambios, consultas del tipo SQL, visualización de datos, muchos eventos, etc. La serialización a/desde XML no es la más rápida. XmlSerializer supera fácilmente su desempeño. Puede ser problemático con clientes de Servicios Web no .NET que devuelven datasets tipificados. Además muchos documentos jerárquicos (y perfectamente válidos) y sus esquemas no pueden expresarse en un modelo de tablas planas.

El XmlSerializer mejora las posibilidades para el manejo de datos XML. Mediante atributos de serialización, XmlSerializer puede recomponer objetos desde su representación XML, así como también serializar nuevamente a la forma XML. Además, puede hacerlo en forma muy eficiente, dado que genera una clase basada en XmlReader (por lo tanto serializada) compilada en forma dinámica especialmente orientada a desserializar el tipo concreto. Es decir, es realmente rápido.

Un enfoque mucho mejor que la usual generación de código "StringBuilder.Append" es aprovechar las clases en el espacio de nombres **System.CodeDom**; y eso es exactamente lo que hacen las clases de generación de código integradas. CodeDom contiene clases que nos permiten representar casi cualquier construcción de programas en una forma independiente del lenguaje, en lo que se denomina AST (abstract syntax tree - árbol sintáctico abstracto). En forma posterior, otra clase, el generador de código, puede interpretarla y generar el código simple que usted desea, como por ejemplo Microsoft® Visual C# o Microsoft® Visual Basic®.NET. Ésta es la forma en la que se produce la mayoría de la generación de código en .NET Framework.
Pero como hemos comentado a lo largo de este primer capitulo, la generación de código, sólo queda expresada en las capas que no pertenecen al modelamiento del nogocio, por lo que en ocaciones también hay que proceder a su redefinición.

1.3.2 Generación de código basado en modelos de objetos

Esta generación está asociada con al menos un método de análisis y diseño, soportados por herramientas comerciales. La entrada básica para la obtención de código es el modelo de estructura de los objetos.Las herramientas ofrecen mecanismos para mantener los modelos y el código sincronizados. Han sido utilizados principalmente en desarrollo de sistemas de tiempo real y sistemas empotrados. Hay un mayor alcance en el código generado.

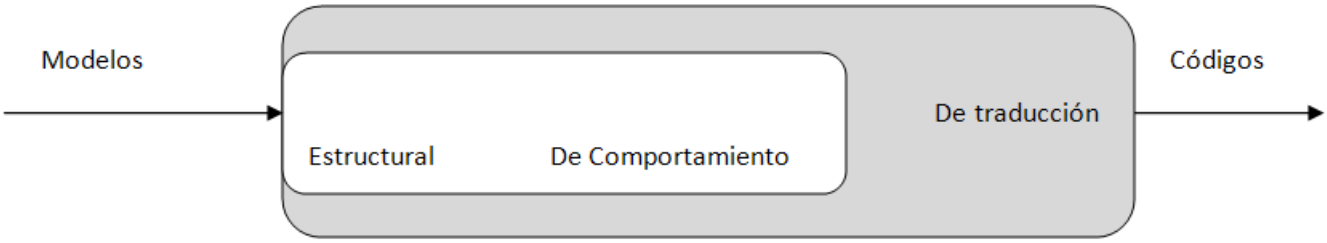


Fig. 2 Evolución de los generadores de código

El Enfoque Estructural : Genera plantillas de código desde el modelo de estructura de objetos. Apropiado para metodologías en las cuales los modelos son elaborados mediante trancisión gradual entre análisis, diseño y código. OMT, OOSE y OOAD.Como hemos comentado ,en este enfoque de generación de código tampoco existe generación de código para el comportamiento de los objetos. Aunque se utilicen máquinas de estado no se les asocia una semántica ejecutable. Algunos productos son productos son: Rational Rose, System Architect .

Se ofrecen mecanismos para integrar código escrito manualmente junto con la estructura generada. Algunas herramientas protegen el código escrito manualmente para evitar reescribirlo en sucesivas generaciones.Se suele ofrecer mecanismos de ingeniería inversa para establecer modelos (la parte estructural).

Enfoque de Comportamiento: Genera código completo usando máquinas de transición de estados y especificaciones de acciones .El beneficio adicional obtenido es la posibilidad de animar y validar el comportamiento del sistema (a partir de los modelos) pero antes de generar el código.Máquinas de estados extendidas para incluir: paralelismo, comunicación y/o jerarquía.Métodos usados para este enfoque: Specification and Description Language (SDL), Statecharts de Harel y ROOM. La programación

se reduce pero algunos aspectos aún deben incorporarse manualmente.Los traductores ofrecen poco control sobre la generación de código.Algunos productos que incorporan un enfoque de comportamiento: Rhapsody, ObjectTime, Tau/Developer.

Traducción: Usa un modelo de arquitectura independiente de la aplicación para dar un control total sobre la traducción a código.

Se desarrollan modelos de arquitectura: un conjunto de patrones de código llamados “archetypes” que establecen reglas de traducción.Típicos aspectos incluidos en un archetype establecen: concurrencia (threads, multitarea, sin concurrencia), manejo de eventos (colas, comunicación entre procesos, flujos I/O) y datos (estructuras, mecanismos, persistencia).El modelo de arquitectura es independiente del modelo de la aplicación. Esto favorece la reutilización de ambos modelos.La construcción de un modelo de arquitectura es en sí un proyecto. Se utiliza un lenguaje de scripts, accediendo al repositorio. Pueden ofrecerse librerías y macanismos de composición y especialización de arquitecturas.Suelen proveerse arquitecturas genéricas que pueden modificarse. Incluso podrían ser desarrolladas por otras empresas.Al igual que en el enfoque de comportamiento, la generación es sólo en un sentido.Algunos productos: BridgePoint, Intelligent CCG, OBLOG .

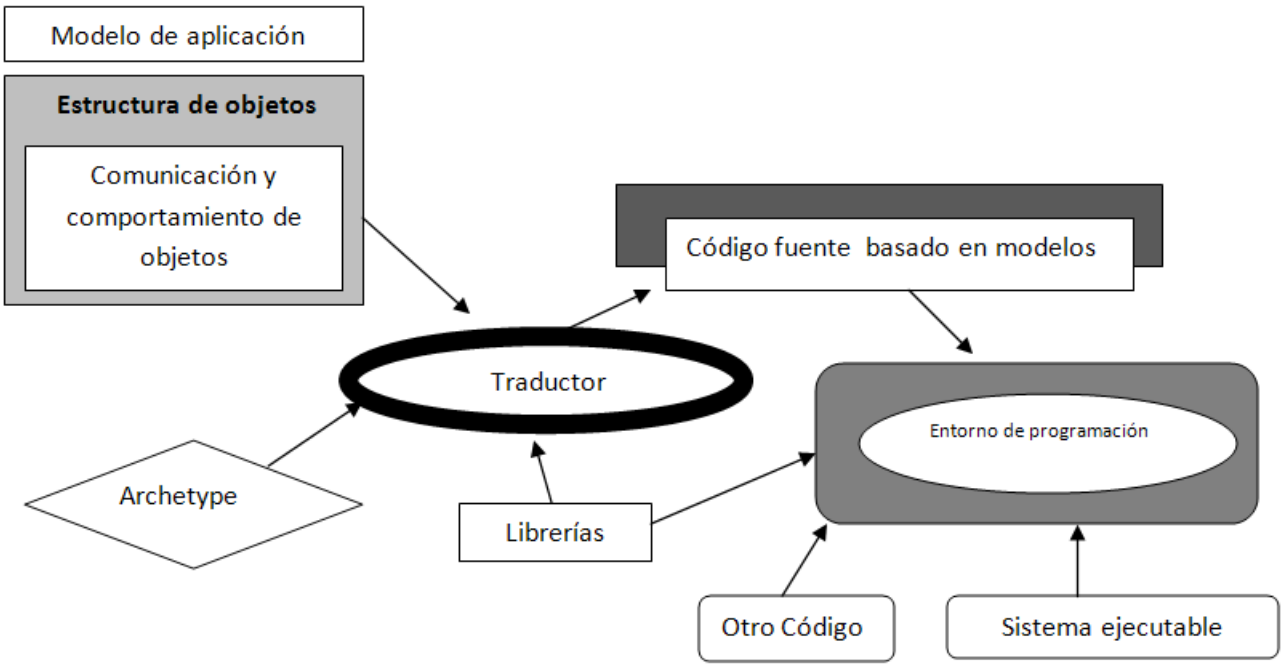


Fig. 3 Enfoques de la generación de código basada en modelos

1.4 Sistemas Expertos

Los Sistemas Expertos Basados en Conocimientos (SE) son programas de computadora diseñados para resolver problemas que requieren de expertos humanos (EH) para su solución. Entendemos por EH a la persona que tiene conocimientos profundos de un cierto tema (por lo general estrecho o específico, de ahí su expertez o destreza), y tiene experiencia en resolver con ellos problemas útiles; como por ejemplo: diagnosticar enfermedades en el caso del Médico, diseñar catalizadores en el caso del Químico, diseñar un puente en el caso de un Ingeniero Civil, detectar una falla en un automóvil en el caso de un Ingeniero Mecánico, o localizar un mineral en una determinada zona geográfica en el caso de un Geólogo.

La mayoría de las computadoras ejecutan hoy en día una gran cantidad de programas que realizan decisiones lógicas, sin embargo los programas utilizan poca cantidad de conocimiento. Estos programas están divididos en dos partes: algoritmos y datos. Los algoritmos especifican los pasos para resolver un problema específico, los datos caracterizan los parámetros del problema en particular. Los expertos humanos, por otra parte, no siguen este modelo para resolver un problema, utilizan fragmentos de conocimiento y su experiencia, para alcanzar la solución de un problema en particular.

Los SE representan estos fragmentos de experiencia y conocimiento en una base de conocimientos (BC), que posteriormente es accedida para razonar sobre un problema en particular. Como consecuencia de esto, los SE difieren de los programas convencionales en su arquitectura, en la forma en que se incorpora el conocimiento, en la manera interactiva en que se ejecutan y en la impresión que crean en los usuarios que lo utilizan; muestran, generalmente, un comportamiento similar al de un EH.

Los SE tienen capacidad para resolver problemas muy difíciles, tan bien o mejor que un EH, razonar heurísticamente utilizando reglas que los EH consideran eficaces, interactuar eficazmente y en lenguaje natural con las personas, manipular expresiones simbólicas y razonar sobre ellas, funcionar con datos erróneos y reglas imprecisas, contemplar múltiples hipótesis alternativas, explicar por qué plantean sus preguntas cuando están intentando resolver un problema, y justificar sus conclusiones.

El estudio y desarrollo de los sistemas expertos (SE) comenzó a mediados de la década del 60. Entre 1965 y 1972 fueron desarrollados varios de estos sistemas, muchos de ellos tuvieron un alcance muy limitado, otros como mycin , dendral y prospector , constituyeron la base histórica de los SE y aún en la actualidad son de gran interés para los investigadores que se dedican al estudio y construcción de los mismos. [11]

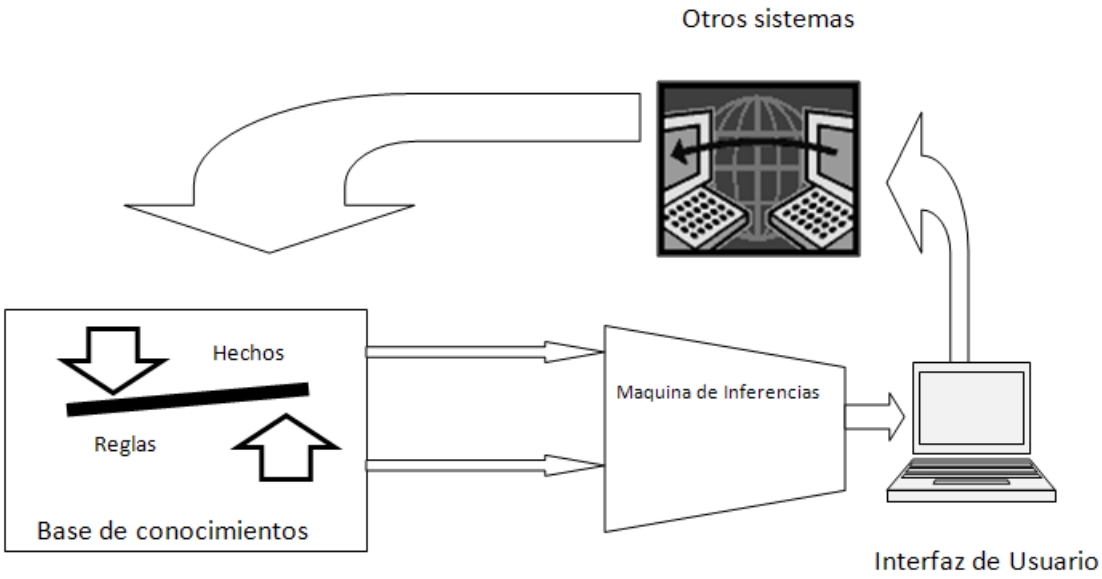


Fig. 4 Arquitectura de Sistema Experto

1.4.1 Sistemas Expertos basados en reglas

En la vida diaria se pueden encontrar muchas situaciones complejas gobernadas por reglas deterministas, como lo son los sistemas de control de tráfico, las transacciones bancarias, etc. Los sistemas basados en reglas son una herramienta eficiente para tratar estos problemas. Las reglas deterministas constituyen las más sencillas de las metodologías que utilizan los sistemas expertos. La Base de Conocimiento contiene las variables y el conjunto de reglas que definen el problema, por otra parte el motor de inferencia obtiene las conclusiones aplicando la lógica clásica a estas reglas. Una regla es una proposición lógica que relaciona dos o más objetos e incluye dos partes;la premisa y la conclusión . Cada una de ellas constituye un expresión lógica que tiene una o varias afirmaciones Objeto-Valor conectados mediante los operadores lógicos Y,O,o NO. Una regla se escribe normalmente como “si premisa, entonces conclusión ”.

Motor de Inferencia

Hay dos tipos de elementos, los datos que son hechos o evidencias y el conocimiento que lo forman el conjunto de reglas almacenadas en la base de conocimiento.El motor de inferencia usa ambos elementos

para obtener nuevas conclusiones o hechos.Si la premisa de una regla es cierta, entonces la conclusión de esa regla debe ser también cierta. Los datos iniciales o datos de partida como las conclusiones derivadas de ellos forman parte de los hechos o datos de que dispone en un instante dado.

Para obtener conclusiones, los expertos utilizan diferentes tipos de reglas y estrategias de inferencia y control. Como lo son las reglas Modus Ponens y Modus Tollens; y las estrategias de inferencia, por encadenamiento de reglas y por encadenamiento de reglas orientado a un objetivo.

Modus Ponens y modus Tollens

El modus Ponens es la regla de inferencia más utilizada, sobre todo para obtener conclusiones simples. En ella se examina la premisa de la regla, y si es cierta, entonces la conclusión pasa a formar parte del conocimiento. Modus Tollens tmabién se utiliza para obtener conclusiones simples,en este caso se examina la conclusión y si es falsa se concluye que la premisa también es falsa.

Encadenamiento de reglas

Esta estrategia puede utilizarse cuando las premisas de ciertas reglas coinciden con las conclusiones de otras. Cuando se encadenan las reglas los hechos pueden utilizarse para dar lugar a nuevos hechos. El tiempo de terminación del proceso depende de los hechos conocidos y de las reglass que se activan. Este algoritmo puede ser implementado de varias formas , una de ellas comienza con las reglas cuyas premisas tienen valores conocidos, estas reglas deben concluir y sus conclusiones dan lugar a nuevos hechos, estos se añaden a un conjunto de hechos ocnocidos, y el proceso se continúa hasta que no puedan obtenerse nuevos hechos.

La Figura 6 muestra un ejemplo de seis reglas que relacionan 13 objetos, del A al M. Las relaciones entre estos objetos implicadas por las seis reglas pueden representarse gráficamente, tal como se muestra en la Figura 7, donde cada objeto se representa por un nodo. Las aristas representan la conexión entre los objetos de la premisa de la regla y el objeto de su conclusión. Nótese que las premisas de algunas reglas coinciden con las conclusiones de otras reglas. Por ejemplo, las conclusiones de las Reglas 1 y 2 (objetos C y G) son las premisas de la Regla 4.

Supóngase que se sabe que los objetos A,B,D,E, F,H e I son ciertos y los restantes objetos son de valor desconocido. La Figura 8 distingue entre objetos con valor conocido (los hechos) y objetos con valores desconocidos.

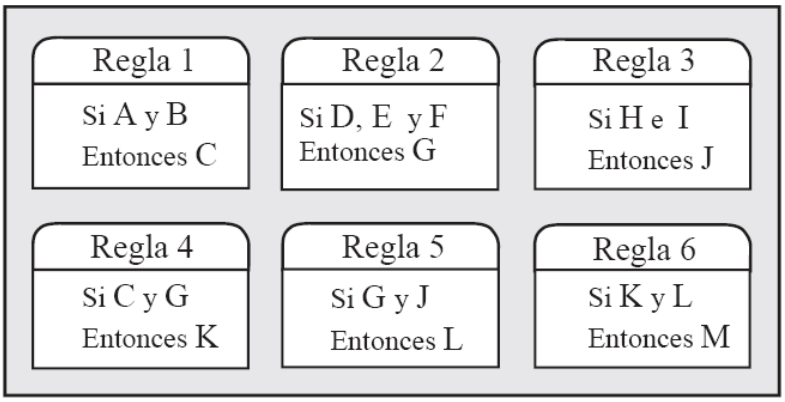


Fig. 5 Conjunto de 6 reglas relacionando 13 objetos

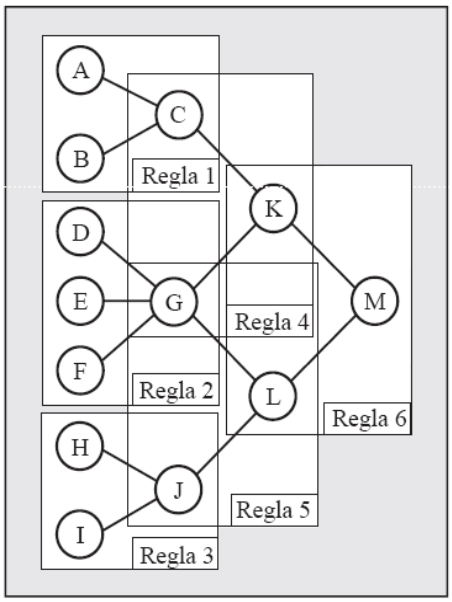


Fig. 6 Relaciones entre reglas

En este caso, el algoritmo de encadenamiento de reglas procede como sigue:

• La Regla 1 concluye que C = cierto.
• La Regla 2 concluye que G = cierto.
• La Regla 3 concluye que J = cierto.
• La Regla 4 concluye que K = cierto.
• La Regla 5 concluye que L = cierto.
• La Regla 6 concluye que M = cierto.

Puesto que no pueden obtenerse más conclusiones, el proceso se detiene.Este proceso se ilustra en la Figura 8, donde los números en el interior de los nodos indican el orden en el que se concluyen los hechos.

Considérense de nuevo las seis reglas de la Figura 6 y supngámos ahora que se dan los hechos

H = cierto, I = cierto, K = cierto y M = falso. Esto se ilustra en la Figura 8, donde los objetos con valores conocidos (los hechos) aparecen sombreados, la variable objetivo se muestra rodeada por una circunferencia.

Supóngase, en primer lugar, que el motor de inferencia usa las dos reglas de inferencia Modus Ponens y Modus Tollens. Entonces, aplicando el Algoritmo , se obtiene

1. La Regla 3 concluye que J = cierto (Modus Ponens).
2. La Regla 6 concluye (Modus Tollens) que K = falso o L = falso,
pero, puesto que K = cierto, deber´a ser L = falso.

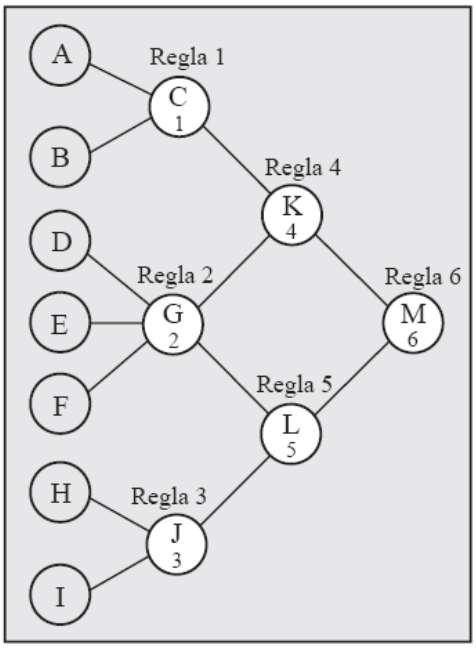


Fig. 7 Encadenamiento entre reglas

La Regla 5 concluye (Modus Tollens) que G = falso o J = falso, pero, puesto que J = cierto, deber´a ser G = falso.

En consecuencia, se obtiene la conclusión G = falso. Sin embargo, si el motor de inferencia sólo utiliza la regla de inferencia Modus Ponens, el algoritmo se detendrá en la Etapa 1, y no se concluirá nada para el objeto G. Este es otro ejemplo que ilustra la utilidad de la regla de inferencia Modus Tollens.Nótese que la estrategia de encadenamiento de reglas diferencia claramente entre la memoria de trabajo y la base de conocimiento. La memoria de trabajo contiene datos que surgen durante el período de consulta. Las premisas de las reglas se comparan con los contenidos de la memoria de trabajo y cuando se obtienen nuevas conclusiones son pasadas también a la memoria de trabajo.

Encadenamiento de Reglas Orientado a un Objetivo

El algoritmo de encadenamiento de reglas orientado a un objetivo requiere del usuario seleccionar, en primer lugar, una variable o nodo objetivo; entonces el algoritmo navega a través de las reglas en la búsqueda de una conclusión para el nodo objetivo.Si no se obtiene ninguna conclusión con la información existente,entonces el algoritmo fuerza a preguntar al usuario en busca de nueva información sobre elementos que son relevantes , para obtener información sobre el objetivo.

Control de la Coherencia

Puede darse el caso de que hasta los veraderos expertos den información inconsistente(por ejemplo, reglas inconsistentes y/o hechos),por eso es importante controlar la coherencia del conocimiento tanto durante la construcción de la base de conocimiento como durante los procesos de adquisición de datos y razonamiento.Si la base de conocimientos contiene información inconsistente, entonces es muy probable que el sistema experto se comporte de forma poco satisfactoria y obtenga conclusiones absurdas.

Controlar la coherencia logra los siguientes objetivos:

- Ayudar al usuario a no dar hechos inconsistentes,dándole al usuario las restricciones que debe satisfacer la información demandada.

- Evitar que entre en la base de conocimiento cualquier tipo de conocimiento inconsistente ocontradictorio.

Para lograr controlar la coherencia, ésta debe existir tanto en las reglas como en los hechos. (Gutiérrez).

Capítulo 2: Funcionalidades y soluciones técnicas del sistema.

Tal y como se explica en la Fundamentación teórica, proporcionar un conjunto de herramientas bien integradas, que simplifiquen el trabajo, utilizando técnicas de unificación y automatizando todas o algunas fases de ciclo de vida del software; ha sido el propósito fundamental del Software Asistido por Computadoras.

Durante el proceso de desarrollo de software, mientras se construyen los diagramas de clases, normalmente los desarrolladores representan como atributos de estas, componentes visuales interesantes para formar interfaces adecuadas .Actualmente el código fuente de estas clases que generan las herramientas CASE solo constituye un código fuente “esqueleto” o sea si tenemos una clase Esqueleto, con un atributo tipo entero llamado Cantidad_ huesos solo se puede generar lo que se muestra a continuación en código del lenguaje de programación C#.

```
public class Esqueleto
 {
   private int Cantidad_ huesos;
 }
```

Para muchos especialistas de la Ingeniería del Software resultaría de mucho interés lograr controlar la codificación de los componentes de las interfaces visuales de las clases desde herramientas CASE. Esta posibilidad permite que se genere el código fuente sin tener que esperar a la utilización de una plataforma de programación específica. Sin hacer un análisis estadístico de este aspecto podemos asegurar que al realizar las actividades de la gestión del tiempo del proyecto se obtendrán mejores resultados, así como la obtención temprana de defectos en la codificación. La mejora de productividad se consigue a través de la automatización de determinadas tareas, como la generación de código y la reutilización de objetos o módulos.

Actualmente para el desarrollo de software, tenemos muchas herramientas. Pero aún el mejor IDE(Entorno de Desarrollo Integrado), nos obliga a sentarnos enfrente, y armar las ventanas arrastrando de a uno los botones. ¿No se podrá delegar gran parte de ese trabajo en un agente de software? Un generador de código puede ser más que un wizard, o un activo de una "software factory" de la IDE de moda: puede ser un sistema experto, que tome decisiones, que dado un modelo, llegue más allá de lo automático. Tal vez, éste es el aspecto más oscuro de explicar pero será nuestro principal propósito

durante el capítulo . Podemos conseguir más que una herramienta, un agente. Algo más sobre "inteligencia artificial".

Una estrategia para generar código muy empleada es la compilación,la cual implica:

- Un lenguaje de alto nivel
- Una herramienta que pase de ese lenguaje a algo que entienda la máquina

Hace un tiempo, apareció el concepto de DSL (Domain Specific Languages). En lugar de usar un lenguaje que modele en alto nivel algo genérico (como algunos pretenden con UML), se pueden inventar lenguajes que modelen, expresen, las necesidades que tenemos y que haya herramientas que, interpretando, procesando esos modelos expresados en un lenguaje específicamente orientado al dominio a solucionar, vayan generando el artefacto final necesario para pasar del modelo de alto nivel a algo más cercano a la máquina. Pero no sirven para hacer sistema cualquiera. Por ejemplo nadie escribe formularios en SQL o hace cálculos científicos o rutinas recursivas. Sólo está dedicado a solucionar el tema de acceso a datos. Pero ha sido lo bastante potente y flexible, para que hoy lo hayamos adoptado en muchos de nuestros sistemas. Pero, aunque no son genéricos y son específicos de un dominio, si son flexibles y hacen lo que necesitamos hacer.

Estas dificultades mencionadas y otras, favorecen a los que mencionan que utilizaron generadores de código, para generar el código inicial de un sistema, y luego modificar el código generado, y no puede regenerar sin perder los cambios que se hicieron.

2.1 Objeto de automatización

Como se presenta desde el diseño teórico de la investigación, no existe ninguna herramienta que trate de resolver seriamente a los programadores la importante tarea de codificar las reglas del negocio durante la generación de código fuente para clases de formularios. El presente proyecto se propone lograr resolver esta problemática desde el punto de vista de la generación de código fuente que resulte de los componentes de los formularios a partir de un modelo de entrada UML que contenga la información primaria para que la herramienta logre su propósito; obtener el código resultante lo más parecido posible al que hubiera hecho un programador.

2.2 Tratamiento de la información

2.2.1 Modelo de entrada UML

La librería que se utiliza para dar tratamiento al modelo de entrada del generador aprovecha un conjunto de convenciones ampliamente aceptadas para representar los conceptos relacionados al análisis y diseño orientado a objetos que establece el Lenguaje de Modelado Unificado. La librería de clases nUML provee el espacio NUML.Uml2 para manipular modelos UML, tratando de cumplir con la versión 2.0 del estándar (UML 2.0 Superstructure FTF convenience document).

Tanto ExpertCoder como nUML soportan parcialmente XMI (XML Metadata Interchange) versión 2.0, en la librería NUML.Xmi2. Este estándar especifica cómo serializar modelos utilizando XML, y permite que distintas herramientas de modelado puedan intercambiar modelos entre sí. Esta última característica es de vital importancia para un proyecto futuro, brinda las funciones para automatizar el intercambio de servicios con la herramienta que se construye.

El proyecto utiliza las posibilidades de los estándares mencionados, obliga a encontrar alguna herramienta de modelado UML que soporte XMI 2.1 para crear los modelos que serán el punto de entrada del generador de código. También es posible usar herramientas para UML 1.4 y XMI 1.2 y otras variantes.

Las especificaciones sobre UML presentadas en el capítulo 1 deberían ser utilizadas por aquellos que no dominan los conceptos de UML, para continuar entendiendo los pasos que define nuestra herramienta.

En el ambiente de generación de código que propone la presente tesis; interesan los conceptos representados por las imágenes que muestran los diagramas. El conjunto de estos conceptos representados es lo que llamamos Modelo, y es en definitiva el diseño que el autor del diagrama quiso representar gráficamente, se convierte en el interés principal nuestra primera materia prima.

Utilizando las clases e interfaces de NUML.Uml2 podemos crear modelos mediante programación. El espacio de nombres NUml.Uml2 provee una interfaz por cada clase integradora definida en el estándar, y en cada una de estas interfaces están definidos los métodos y propiedades especificados en el estándar UML 2.0; por lo tanto, una fuente excelente de documentación es el mismo estándar, donde están perfectamente explicados todos los elementos del UML así como sus relaciones.

2.2.2 Las Plantillas en el generador de código.

Las plantillas son documentos de texto plano, donde pueden escribirse algunos marcadores especiales de los que este trabajo de tesis obtendrá importantes provechos. Estos marcadores pueden luego ser reemplazados por valores durante la ejecución de un programa. No hay un formato predefinido para los marcadores; cada programador puede escoger el esquema que más le guste o que más le convenga.

Dentro del proyecto ExpertCoder se encuentra la librería para manipular plantillas es ExpertCoder.Templates.dll. Esta librería garantiza el tratamiento a las plantillas desde cualquier parte del código que usaremos para escribir el generador.

Los marcadores pueden reemplazarse por una cadena de caracteres cualquiera, sin importar de donde se obtuvo dicha cadena. Por lo tanto, el valor de un marcador podría obtenerse a partir de otra plantilla. Aprovechando esto es posible crear árboles de plantillas, o sea, estructuras jerárquicas donde una plantilla contiene a otras plantillas, y estas a su vez podrian contener en su interior el código que se quiere generar organizado en una jerarquía de objetos.

2.2.3 Representación Intermedia del modelo

Es necesario comprender los conceptos detrás de los atributos de metadatos para entender cómo ocurre la serialización de XML. Cuando hablamos de los atributos de esta sección, de manera que siempre se refieren a atributos de los metadatos, y no los atributos XML. Para el caso específico de nuestra herramienta, la serialización nos interesa para entender la forma en que el modelo de entrada se presenta al generador de código que realizaremos.

La serialización es el proceso de convertir un objeto de forma que pueda transportarse fácilmente. Por ejemplo, puede serializar un objeto y transportarlo por Internet utilizando HTTP entre un cliente y un servidor. En el otro extremo, la deserialización reconstruye el objeto a partir de la secuencia.

La serialización XML sólo serializa los campos públicos y los valores de propiedad de un objeto en una secuencia XML. No incluye información de los tipos. Por ejemplo, si un objeto Book existe en el espacio de nombres Library, no hay ninguna garantía de que se deserialice en un objeto del mismo tipo.

La clase central de la serialización XML es XmlSerializer y sus métodos más importantes son Serialize y Deserialize. XmlSerializer crea archivos C# y los compila en archivos .dll para realizar esta serialización. En .NET Framework 2.0, la XML Serializer Generator Tool (Sgen.exe) está diseñada para generar estos

ensamblados de serialización con antelación con el fin de que puedan implementarse junto con su aplicación y mejorar el rendimiento de inicio.

Los datos de los objetos se describen utilizando construcciones de lenguaje de programación como clases, campos, propiedades, tipos primitivos, matrices e incluso XML incrustado en forma de objetos XmlElement o XmlAttribute. Puede crear clases propias, anotadas con atributos, o utilizar la herramienta Definición de esquemas XML para generar las clases de acuerdo con un esquema XML existente.

Si dispone de un esquema XML, puede ejecutar la herramienta Definición de esquemas XML con el fin de generar un conjunto de clases con establecimiento inflexible de tipos para el esquema y anotarlas con atributos. Cuando se serializa una instancia de dicha clase, el XML generado se ajusta al esquema XML. Con esa clase puede programar con respecto a un modelo de objeto fácilmente manipulado al tiempo que se asegura que el XML generado se adecua al esquema XML. Se trata de una alternativa a la utilización de otras clases de .NET Framework, como XmlReader y XmlWriter, para analizar y escribir una secuencia XML. (MSDN, 2008)

La siguiente Figura muestra un ejemplo de cómo es serializado y deserializado un conjunto de objetos utilizando XmlSerializer.

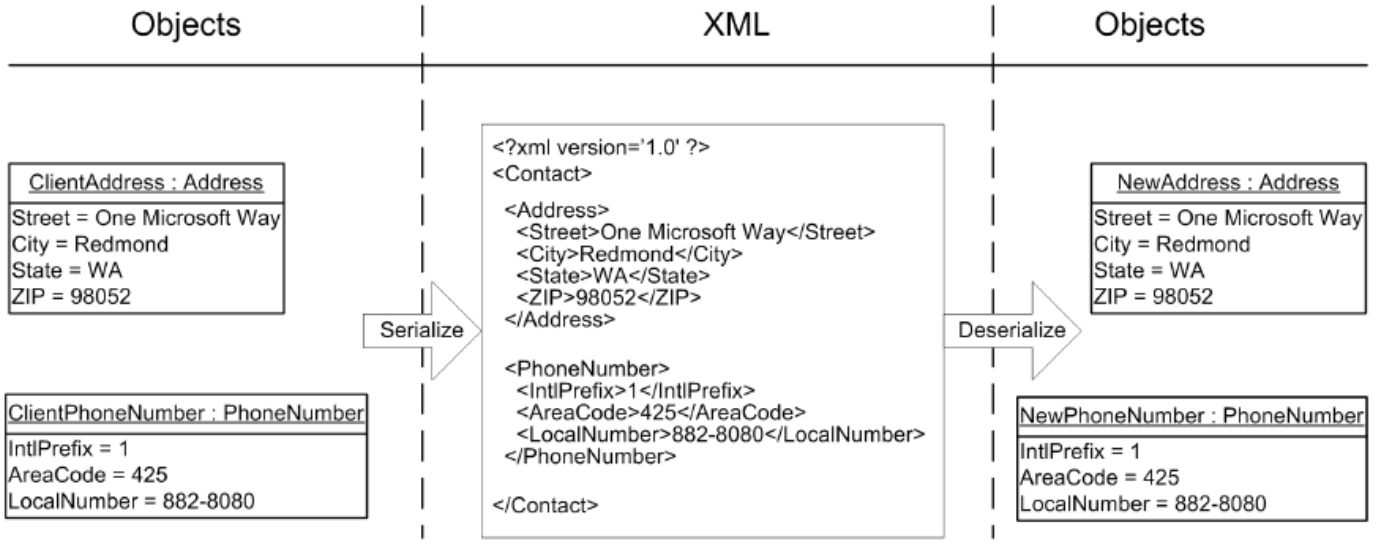


Fig. 8 Transformación de objetos UML

2.3 Funcionamiento de la herramienta

La Herramienta de Generación de código mediante un Sistema Experto(HGCE), que propone esta tesis, inserta dentro de sus necesidades a la librería de clases:ExpertCoder, es un conjunto de herramientas para escribir generadores de código basados en sistemas expertos que corre bajo la plataforma .NET. No es un generador de generadores de código, sino un conjunto de librerías que sirven para escribir generadores.

Este trabajo de diploma tiene el propósito de construir un conjunto de herramientas que provean al desarrollador de generadores de código dos avances importantes: La claridad que aporta el uso de plantillas para la generación de código. La potencia de la plataforma .NET, junto a su enorme librería de clases que junto a incorporación de estas funcionalidades brinda un gran poder de expresión al desarrollador.

Adicionalmente, al estar basado en los principios de los sistemas expertos, los generadores resultantes son fácilmente extensibles, modulares y su estructura es más declarativa que imperativa.

Para construir esta herramienta se creará un sistema experto, escribiendo un conjunto de reglas y especificando las distintas precedencias entre ellas.

Estas reglas son evaluadas por un motor de ejecución, quien determina en base a las precedencias y al estado de activación de cada regla cual ejecutar.

2.4 Curso del modelo en el generador

El siguiente digrama muestra los diferentes estados por los que pasa el modelo que se quiere convertir en el generador , así como el tratamiento que recibe en los diferentes componentes que conforman la herramienta.

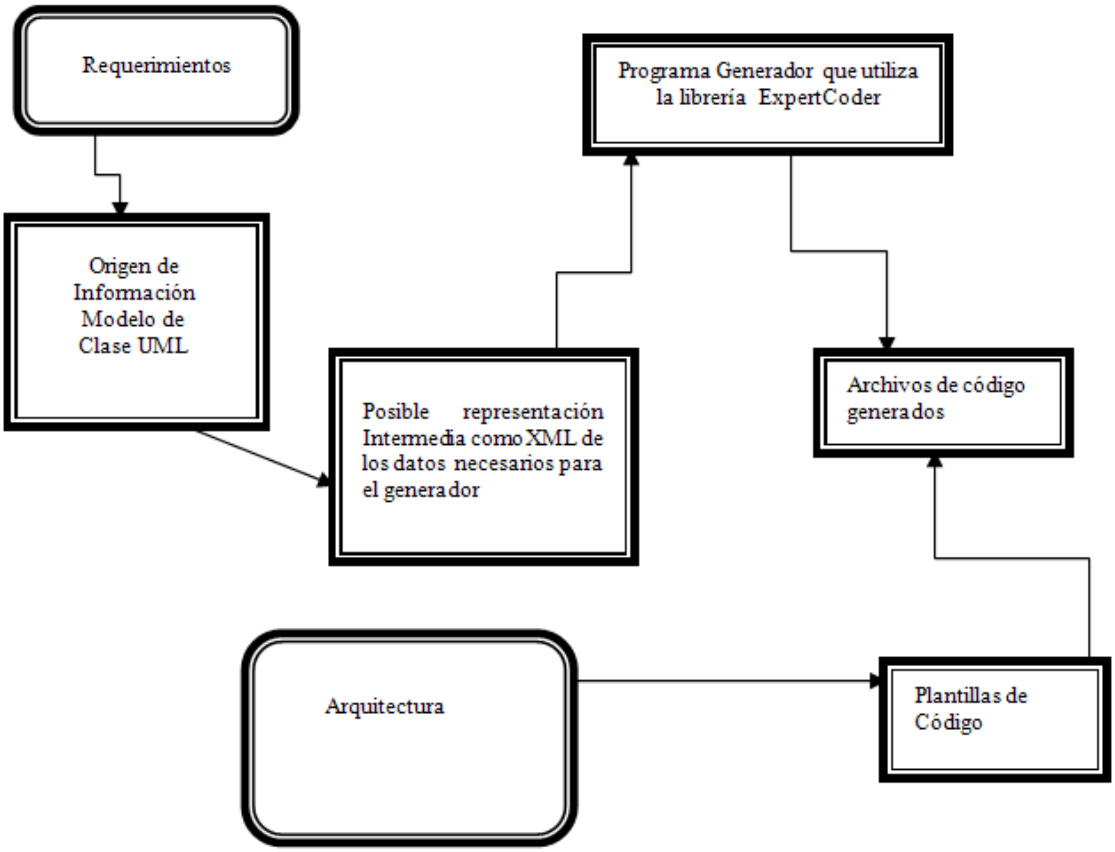


Fig. 9 Flujo de funciones en el generador

2.5 Modelo de dominio

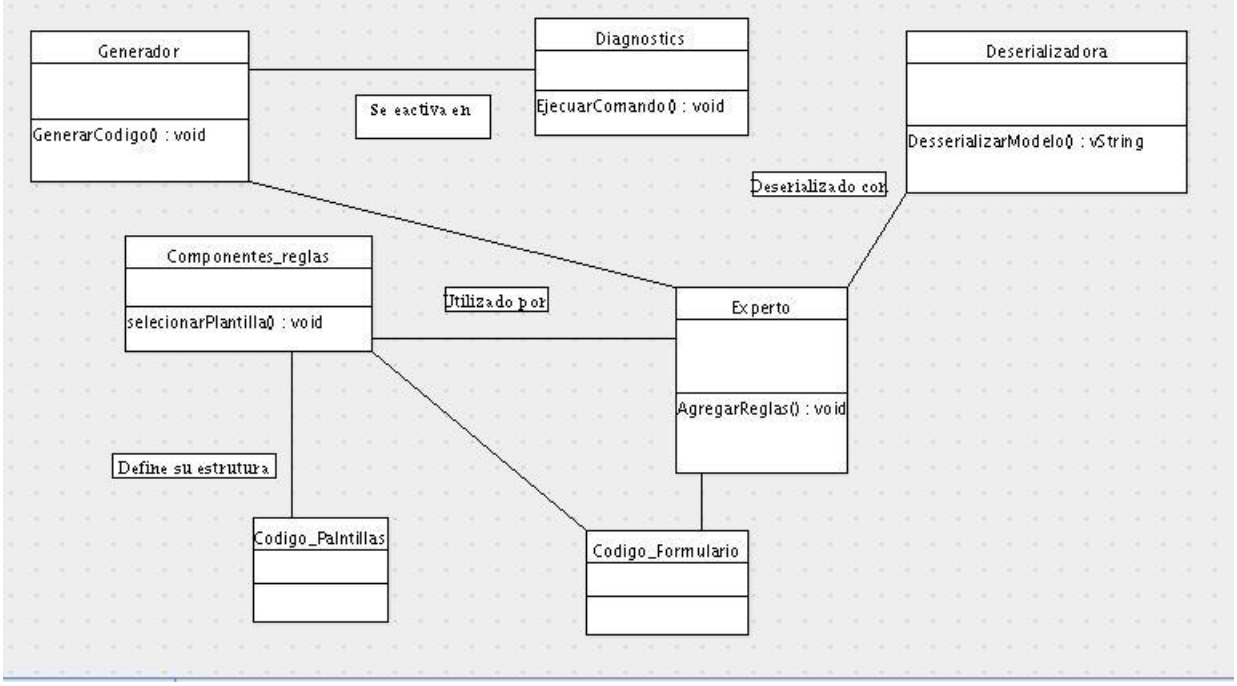


Fig. 10 Modelo de dominio

2.6 Prototipo de Interfaz de Usuario

La interfaz de usuario es el enlace entre el sistema experto y el usuario. Por ello, para que un sistema experto sea una herramienta efectiva, debe incorporar mecanismos eficientes para mostrar y obtener información de forma fácil y agradable. Un ejemplo de la información que tiene que ser mostrada tras el trabajo del motor de inferencia, es el de las conclusiones, las razones que expliquen tales conclusiones y una explicación de las acciones iniciadas por el sistema experto. Por otra parte, cuando el motor de inferencia no puede concluir debido, por ejemplo, a la ausencia de información, la interfaz de usuario es un vehículo para obtener la información necesaria del usuario. Consecuentemente, una implementación inadecuada de la interfaz de usuario que no facilite este proceso minaría notablemente la calidad de un sistema experto. Otra razón de la importancia de la interfaz de usuario es que los usuarios evalúan comúnmente los sistemas expertos y otros sistemas por la calidad de dicha interfaz más que por la del sistema experto mismo.

Fig. 11 Prototipo de interfaz de usuario

2.6 Especificación de los Requisitos del software

2.6.1 Dependencias y Relaciones con otros software.

Para esta primera versión de la herramienta, se decidió utilizar como modelo de entrada, un modelo de clase UML realizado en el software Argo UML.

ArgoUML es una aplicación de diagramado de UML escrita en Java y publicada bajo la Licencia BSD open source. Dado que es una aplicación Java, está disponible en cualquier plataforma soportada por Java.

Como entorno de desarrollo proponemos MonoDevelop es un entorno integrado, libre y gratuito, diseñado primordialmente para C# y otros lenguajes .NET como Nemerle, Boo, y Java. MonoDevelop originalmente fue una adaptación de SharpDevelop para Gtk#, pero desde entonces se ha desarrollado para las necesidades de los desarrolladores de Mono. El IDE incluye manejo de clases, ayuda incorporada, completación de código, Stetic (diseñador de GUI), soporte para proyectos, y un depurador integrado.

2.6.2 Requerimientos funcionales

1-Cargar modelo de clase UML.

1.1-La herramienta debe permitir examinar para seleccionar la ubicación donde se encuentra el modelo de clase construido en el software Argo UML.

2-Agregar nuevas plantillas y clases para tratar las plantillas en el generador.

2.1-Crear nuevas plantillas de extensión TXT y las respectivas clases para tratar estas plantillas.

2.2 El generador debe permitir examinar para guardar estas plantillas y sus clases en la carpeta (Templates) donde está el código del generador.

3-Generar código fuente en un lenguaje de programación detallado en las plantillas.

3.1- Permitir guardar el código en una ubicación especificada por el usuario

2.6.3 Requerimientos no funcionales

La herramienta de generación de código debe tener una interfaz amigable, entendible para los usuarios que son los desarrolladores con las funcionalidades fácilmente visibles y sin ambigüedades.

Es necesario utilizar los softwares mencionados anteriormente para el correcto funcionamiento de esta primera versión de la Herramienta de generación de código mediante el sistema experto.

2.7 Proceso Unificado de Desarrollo como metodología para la construcción de la herramienta.

Una buena arquitectura ofrece una plataforma estable, ésta se crea a partir de la creación de los componentes reutilizables, diseñados de tal forma que puedan ser utilizados conjuntamente. El UML ayuda a este proceso, ya que crea componentes específicos que pueden estar disponibles para su reutilización.

Se conoce que los casos de uso interaccionan con la arquitectura. Si el sistema proporciona los casos de uso correctos, los usuarios no tendrán mayor problema para llevar a cabo sus objetivos usando el sistema. Para lograr obtener esos casos de uso es necesario construir una arquitectura que nos permita implementar los casos de uso de una forma económica en todo momento.

Los casos de uso dirigen a la arquitectura, pero también se utiliza el conocimiento de la arquitectura para obtener nuevos casos de uso. Dicho de otra forma, la arquitectura guía los casos de uso.

Es de mucho interés construir primeramente una arquitectura básica a partir de la comprensión del dominio en que se desarrollará el sistema, crear los casos de uso sin ser detallados, a partir de ahí, seleccionar algunos casos de uso y hacer que la arquitectura se adapte a ellos, con esto se logra que se fortalezca más la arquitectura; posteriormente se seleccionan otros casos de uso y se sigue el mismo proceso.

Un proceso de desarrollo eficaz debe tener una serie de hitos, los cuales proporcionarán los criterios suficientes para saber en que momento se pueda pasar de una fase a otra dentro del ciclo de vida del sistema. Dentro de cada fase, el proceso unificado pasa a través de una serie de iteraciones, las cuales conducen a esos criterios. Esta fortaleza combiene, pues ayuda a organizarnos e ir evaluando el desarrollo de nuestro software.

2.8 UML como lenguaje de modelado seleccionado.

Como ya se ha mencionado, el UML no es un método sino un lenguaje, el cual define únicamente una notación y un metamodelo. El UML al ser un lenguaje estándar no depende de un proceso de desarrollo, y esto es precisamente lo que se quería al lograr unificar los métodos, que se tuviera un lenguaje en común entre los diferentes métodos, para que el desarrollador tuviera la libertad de escoger la metodología de su agrado. Con esto los desarrolladores implicados en un proyecto pueden tener la seguridad de que estarán creando diseños de software bajo un lenguaje que será comprendido por todos aquellos que utilicen el UML.

La notación en el UML son los componentes gráficos que se utilizan para crear los metamodelos, es decir, es la sintaxis del lenguaje de modelado. Para el caso de los diagramas de clase, la notación es la forma en cómo se dibuja una clase, la asociación, la multiplicidad, la agregación, etc. Pero el concepto de modelo uml utilizado por la librería de clases ExpertCoder y nUML, concretan la justificación de la utilización de este lenguaje de modelado.

Un metamodelo o modelo, es la representación de algo en cierta forma, para el caso de la Ingeniería del Software un modelo es un diagrama que representa la definición de la notación.

2.9 C Sharp como lenguaje de porgramación utilizado.

El código escrito en C# es autocontenido, lo que significa que no necesita de ficheros adicionales al propio fuente tales como ficheros de cabecera o ficheros IDL.El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código. No se incluyen elementos poco útiles de lenguajes como C++ tales como macros, herencia múltiple o la necesidad de un operador diferente del punto (.) acceder a miembros de espacios de nombres (::). Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que el de C# es más puro en tanto que no admiten ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

La orientación a componentes ha sido uno de nuestros principales motivos para seleccionar el lenguaje. La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas. Es decir, la sintaxis de C# permite definir

cómodamente propiedades (similares a campos de acceso controlado), eventos (asociación controlada de funciones de respuesta a notificaciones) o atributos (información sobre un tipo o sus miembros).

Todo lenguaje de .NET tiene a su disposición el recolector de basura del CLR. Esto tiene el efecto en el lenguaje de que no es necesario incluir instrucciones de destrucción de objetos. C# también proporciona un mecanismo de liberación de recursos determinista a través de la instrucción using.

Para facilitar la migración de programadores, C# mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en código escrito en C# fragmentos de código escrito en estos lenguajes. Existen otras caracteristicas por las que podríamos decir que utilizamos el lenguaje de programación C #,pero aquí están expuestas las fundamentales.

2.10 MonoDevelop el IDE para el desarrollo de la herramienta.

Mono es una libre implementación de la plataforma de desarrollo .Net, enfocada principalmente a sistemas Unix, y también con el propósito de brindar puentes de migración de Windows hacia Linux. El interés es tener un sistema de desarrollo multiplataforma, que permita la interoperación entre diversos lenguajes de programación, así como una arquitectura extensible.

El compilador que más soporta mono es el de C#, que soporta además la sintaxis de C# 2.0, y que ha alcanzado un muy buen grado de estabilidad. También tiene compiladores para JScript, Visual Basic, que están en estados de desarrollo.

Monodevelop, está muy bien ordenado y visualmente muy atractivo. Es importante destacar el diseño de formularios de forma gráfica, donde que incluyeron Stetic, que es un diseñador de formularios Gtk#.

MonoDevelop es una herramienta que acerca un poco más a los programadores que no se atrevían a trabajar sobre linux con lenguajes como C# , o simplemente a los que están acostumbrados a trabajar sobre lenguajes "visuales"

2.11 Argo UML Herramienta para el modelado del proyecto

El principal interés en utilizar esta herramienta tiene que ver con la necesidad de la familiarización con sus funcionalidades; pues es una herramienta implementada en java y con licencia de código abierto. Permite modelar usando UML. Precisamente esta apertura de código permite la creación de módulos

adicionales para la generación de código que podría ser un propósito de futuras versiones o proyectos desglosados de ésta tesis.

Capítulo 3: Estructura interna del generador

Luego de haber definido lo que nuestra herramienta debe hacer y de haber identificado algunas de las funcionalidades requeridas; nos encargaremos de explicar en este capítulo como estará estructurado internamente el generador de código fuente que constituye el componente principal del software construido. Con el apoyo de la documentación de la librería ExpertCoder especificada en la bibliografía de esta tesis se organiza por temas los pasos realizados para organizar internamente el generador de código experto.

3.1Diagrama de Caso de Uso

La siguiente figura muestra las funcionalidades de la versión de la heramienta que presenta este pryecto, englobadas en los casos de uso que darán origen a la arquitectura de nuestro software.

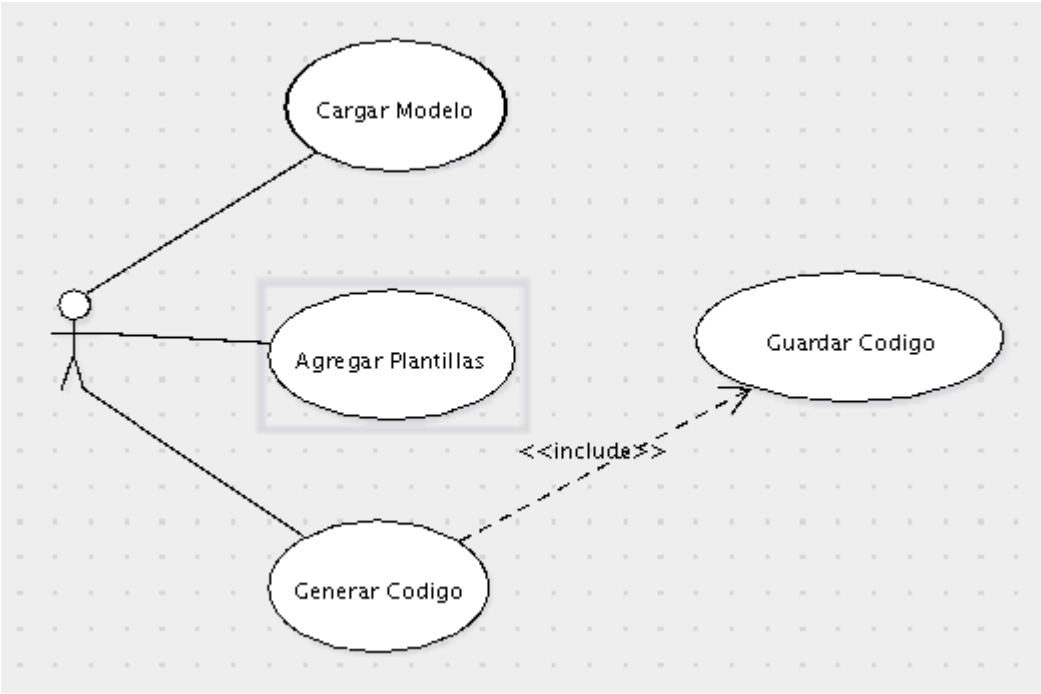


Fig. 12 Diagrama de casos de uso

3.2 Modelo de entrada y salida del generador.

Como se menciona en los capítulos anteriores, el formato del modelo de entrada será UML, resultado de la creación de una clase para formularios con el software Argo UML u otro que permita exportar el codigo en los estándares de XMI especificados. En este proyecto solo se prestará atención a las clases y sus atributos, ignorando el resto de los elementos del modelo, como está incorporado en la implementación de la herraminenta.

Pero para un mejor tratamiento de los elementos que conforman el modelo de entrada, se utilizan las potencialidades que brinda la librería ExpertCoder en cuanto a la serialización de modelos. Para poner a funcionar el generador se utilizó una aplicación llamada ecengine.exe disponible junto a las dlls de ExpertCoder. Esta aplicación toma como entrada un fichero XML de configuración, donde se especifica la tarea a ejecutar. En este fichero se indican el proveedor de modelo, la ubicación del modelo, el generador a usar y sus parámetros. El proveedor de modelo alimenta al generador con el resultado de la deserialización del modelo, que consiste en un IList, que es el formato luego de la transformación del modelo.

Cada modelo de clase se representa en el esquema por un elemento de XML cuyo nombre es el nombre de la clase. La declaración del tipo lista las propiedades de la clase. Por defecto, los modelos satisfechos de elementos de XML que corresponden para modelar las clases no imponen un orden en las propiedades. XMI permite fabricar en serie rasgos que usan elementos de XML o atributos de XML; sin embargo, XMI permite especificar cómo fabricarlos en serie cuando se necesita. (Group, 2007)

El fichero XML que trata la aplicación ecengine.exe también permite la especificación de la ubicación del modelo de salida, que será el resultado final del generador que controla la herramienta.

En el apéndice A se presenta un ejemplo de representación XMI tomado de la documentación de la librería nUML disponible en la documentación especificada en la bibliografía.

El siguiente diagrama es una representación del modelo anterior, tomado de la documentación de la librería de clases nUML.

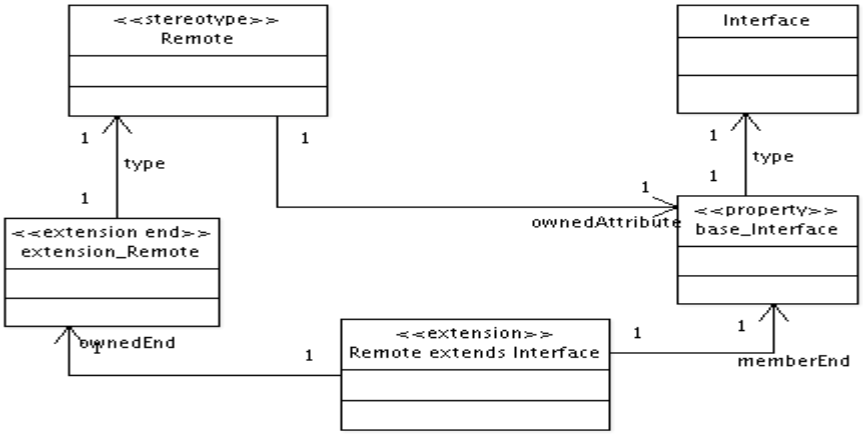


Fig. 13 Representación UML de un modelo XMI

Para la construcción del generador se hubiese podido utilizar como elemento de entrada un modelo con extensión .zargo resultado de la construcción del mismo en la herramienta CASE Argo UML; para darle tratamiento al mismo desde la librería nUML. El proyecto nUML, ya está independiente de ExpertCoder, por lo que se pueden utilizar ambas librerías vinculando las funcionalidades de las mismas para la construcción de la herramienta. En este caso obtendríamos el modelo como se muestra a continuación:

Descomprimiendo el fichero

```
unzip -p <file>.zargo < <file>.xmi | xsltproc <path_to_extras_dir>/fromXMI1_2.xslt - >
<file>.out.xmi
```

Leyendo el fichero descomprimido utilizando nUML:

```
SerializationDriver ser = new SerializationDriver ();
ser.AddSerializer (new NUML.Uml2.Serialization.Serializer ());
IList modelElements = ser.Deserialize (filename);
```

Esta última característica no está incorporada a la herramienta en esta primera versión del generador.

3.3 Creación de plantillas para componentes de las clases formularios.

Como se menciona en el capítulo anterior, las plantillas son documentos de texto plano, que presentan el espacio donde pueden escribirse algunos marcadores especiales, los que pueden ser reemplazados por valores durante la ejecución de un programa, para resultar en un encapsulamiento de datos que contengan el código fuente que se quiere obtener.

Las clases plantillas sólo exponen un campo *Message* de tipo *string* hacia el exterior; este campo permite asignar el valor del marcador que se halla especificado para englobar más información, en nuestro caso código fuente. Esta clase hereda de *Template*, quien provee toda la funcionalidad. En el constructor, se pasan dos parámetros, el primer parámetro indica la ubicación del fichero que contiene la plantilla de texto plano, y un arreglo con las cadenas que deben interpretarse como marcadores. Hasta aquí se le indica a la librería, cual es el texto de la plantilla y cuales son los marcadores; todavía falta indicar el valor de estos últimos.

Para saber el valor de los marcadores cuando utiliza el método *ToString* se llama al método *SetPlaceHoldersValues* que permite obtener los valores que se asignarán a los marcadores. La librería utiliza la sintaxis siguiente para lograr los propósitos mencionados

```
base["$Datos!"] = Datos;
```

asigna el valor del campo Datos al marcador `$Datos!`.

Los marcadores pueden reemplazarse por otras cadenas de caracteres, Por lo tanto, el valor de un marcador podría obtenerse a partir de otra plantilla. Los árboles de plantillas son estructuras jerárquicas donde una plantilla contiene a otras, lo que permite utilizar esta posibilidad para definir variables, que nos permitirán diferenciar solo algunos elementos entre componentes de una clase formulario y no necesariamente al nivel componentes . La clase *TemplateCollection* permite estructurar estas funcionalidades e incluye la del control del formato de lo que imprimen los marcadores del generador de nuestra herramienta.

3.4 Reglas y funcionamiento del Sistema Experto para el generador.

El sistema experto está conformado por dos tipos de reglas, las de navegación y las de producción. Para ciertos tipos de elementos en la entrada del generador de código se activan estas reglas de navegación, navegan las relaciones del elemento y cambia el actual en el modelo de entrada.

Es imprescindible para el funcionamiento correcto del generador de la herramienta, que las reglas de producción se activen ante la presencia de algún elemento en la entrada y en determinada especificación en la salida. Estas reglas aplicarán un algoritmo que garantiza generar nodos a la salida utilizando la información que se encuentra en la entrada y en lo que se considera memoria activa.

Como se explica en la fundamentación teórica en el motor de inferencia de los sistemas expertos basados en reglas, hay dos tipos de elementos, los datos, que son hechos o evidencias y el conocimiento que lo forman el conjunto de reglas almacenadas en la base de conocimiento. El motor de inferencia usa ambos elementos para obtener nuevas conclusiones o hechos.

Para la herramienta, el motor de ejecución provee un entorno, donde hay información proveniente de tres fuentes. Los parámetros que se encuentran almacenados en ficheros de configuración. La entrada del generador que es lo que se quiere convertir, y el conocimiento deducido, el mismo sistema experto puede modificar su memoria activa. De esta manera se puede implementar un mecanismo de interacción indirecto entre reglas. En el entorno del motor se mantiene referencias directas al modelo de entrada, al elemento actual del modelo de entrada, al modelo de salida y al elemento actual del modelo de salida. El subsistema de ejecución de ordenes es la componente que permite al sistema experto iniciar acciones. Estas acciones se basan en las conclusiones sacadas por el motor de inferencia. El siguiente gráfico representa la interacción entre los diferentes recursos utilizados por el sistema Experto para la decisión adecuada en la generación.

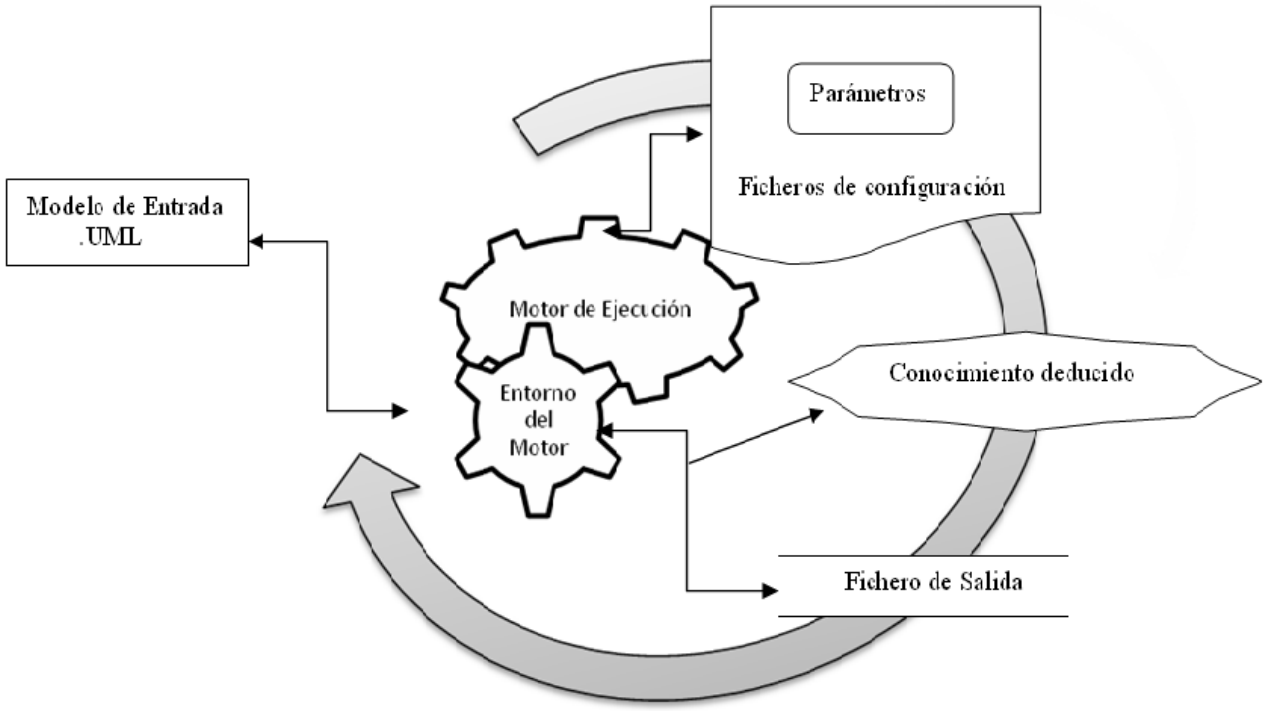


Fig. 14 Representación del funcionamiento del sistema experto

Una de las clases más importantes del generador es la clase que hará función de Sistema Experto. Utilizando la librería ExpertCoder.ExpertSystem y heredando de esta logramos definir nuestra clase experto la cual concretará su funcionamiento con el tratamiento a las reglas.

La API de esta librería de clases brinda la posibilidad de utilizar atributos para agregar información acerca del experto que se construya, como el nombre de la herramienta o más específico del generador,asi como el propósito del mismo. Para generadores que se insertan dentro del funcionamiento de una herramienta *ExpertSystemInformationAttribute* brinda la posibilidad de especificar en el último de sus parámetros, si la información producida en el generador podrá ser utilizada o no por otras herramientas más amigables que el propio generador.

Una regla consiste de una condición de activación y una acción. Cuando la regla derive de la función TypeGuardedRule; la condición de activación está en función del tipo de datos del elemento actual del modelo de entrada y, opcionalmente, del modelo de salida. Estos tipos de datos se indican en el

constructor de la regla. Por ello definimos para el caso de nuestro sistema experto el tipo de datos UML que será el modelo de entrada del generador.

Cada regla define un método que indica que se ejecute la misma, al sobrescribirlo se está especificando la acción de la regla. El motor de ejecución inicializa los parámetros. El método recibe como argumento el *entorno de ejecución* de la regla, a través del cual es posible acceder al sistema experto, consultar cuáles son, y también asignar los elementos actuales de los modelos de entrada y salida.

Para agregar la regla al sistema experto, se especifica que se trabaja sobre el mismo entorno de ejecución. En la clase base del sistema experto se encuentra el método AddRule que permite agregar la regla.

Cuando se provee el modelo, lo que se tiene es el resultado de la deserialización del mismo, es un *IList* si las reglas anteriores se activan solamente ante la presencia de instancias de la clase *class* de la librería UML, se debe especificar otra regla que se active cuando aparezca un *IList* en la entrada del generador.

Esta herramienta necesita informar al sistema experto que la clase base *TypeGuardedRule* debe activarse cuando a la entrada haya una propiedad UML y a la salida una instancia de la plantilla que tiene el código resultante. Al llamar al método AddRule de la clase base y pasarle la regla como parámetro, se está adicionando la regla al sistema experto.

3.5 La precedencia entre reglas y el resultado del generador

Para aprovechar las potencialidades de la librería con la que se construye el sistema experto, en este epígrafe se explica la importancia de la precedencia entre reglas para la efectividad de la generación del código fuente.

El comportamiento por defecto de un generador ante un caso específico, se puede evitar introduciendo una nueva regla y una precedencia. La nueva regla deberá activarse cuando se produzca dicho caso específico, y la precedencia debe establecer que la nueva regla reemplaza a la regla que normalmente se hubiera disparado.
El subsistema de control de la coherencia ha aparecido en los sistemas expertos muy recientemente. Sin embargo, es una componente esencial de un sistema experto. Este subsistema controla la consistencia de la base de datos y evita que unidades de conocimiento inconsistentes entren en la misma. En situaciones complejas incluso un experto humano puede formular afirmaciones inconsistentes. Esta tarea es una responsabilidad humana en nuestro generador.

Cada regla puede contener la especificación que necesitemos para lograr los propósitos de la herramienta. Para determinados tipos de datos de los atributos estas reglas podrían dispararse y generar el código fuente específico del modelo en cuestión.
Como vamos a necesitar más datos además de los que se encuentran en la entrada y la salida, no se hace uso de la clase *TypeGuardedRule* que agrega reglas al generador donde interesa solamente estos dos parámetros. Derivaremos de de la clase *Rule* e implementamos nuestra condición de activación.

Esto podría convertirse en un algoritmo de encadenamiento de reglas orientado a un objetivo que requiere del usuario seleccionar, en primer lugar, una variable o nodo objetivo; entonces el algoritmo navega a través de las reglas en búsqueda de una conclusión para el nodo objetivo. Si no se obtiene ninguna conclusión con la información existente, entonces el algoritmo fuerza a preguntar al usuario en busca de nueva información sobre los elementos que son relevantes para obtener información sobre el objetivo.

El mecanismo de inhibición que se escoja determina cómo se ejecutará la regla, para nuestro generador será muy práctico utilizar *InputElementInhibition* garantizando que la regla se ejecute al menos una vez por elemento de entrada.

Se debe indicar que esta nueva regla reemplaza a alguna regla anterior en específico en caso de que lo necesitemos cuando ambas estén activas. Esto se logra añadiendo una relación de precedencia entre ambas, caracterizada de tipo *reemplazo*. Esto es fruto de lo que sería en el sistema experto el subsistema de adquisición del conocimiento; que controla el flujo del nuevo conocimiento que fluye del experto humano a la base de datos. El sistema determina qué nuevo conocimiento se necesita, o si el conocimiento recibido es en realidad nuevo, es decir, si debe incluirse en la base de datos y, en caso necesario, incorpora estos conocimientos a la misma.

En lugar de directamente agregar las reglas, nos quedamos con las referencias. Luego utilizamos estas referencias para crear una precedencia, donde se indica quien tiene mayor precedencia para anunciar cual regla reemplaza a la otra.

Luego de entender como es el comportamiento de las reglas afirmamos que el motor de Inferencias es el responsable del control. Emplea las reglas, según el contenido de la base de hechos, para solucionar el problema. El motor de inferencias trabaja de modo cíclico, ya que logra una búsqueda de reglas cuyas

premisas son hechos establecidos en la base de hechos. Selecciona una regla aplicable según una estrategia de control. Realiza las acciones de la regla seleccionada, y deduce nuevos hechos que se incluirán en la base de hechos hasta resolver el problema.

3.6 Tratamiento del modelo

La librería utilizada para la construcción del sistema experto y el generador, permite crear programáticamente los modelos de los que se quiere obtener el código fuente. Sin embargo para una mejor interacción con los usuarios; la herramienta en construcción pretende utilizar como se menciona en los capítulos anteriores un modelo construido con otro software, por ahora el software Argo UML que permite guardar modelos con extensión UML. En este proyecto también utilizará la librería de clases nUml.

Esta librería se utiliza para el tratamiento del modelo que contiene la información de la transformación para nuestro sistema experto. Argo UML, en ocasiones utiliza un directorio comprimido para guardar sus archivos. Por eso se debe extraer en estos casos antes de cualquier conversión o procesado. El **Apéndice A** muestra el código exportado a XMI de un modelo de objetos que podrá ser utilizado por la herramienta para obtener el código fuente.

Capítulo 4: Construcción de la herramienta

La herramienta de generación de código construida es un Software de sistemas: software que ayuda (sirve de base) a otro software. (Rojas, 2006). En la fundamentación teórica se define con claridad los objetivos del software que luego se detallaron en el capítulo 2.

4.1 Diagrama de diseño de la herramienta

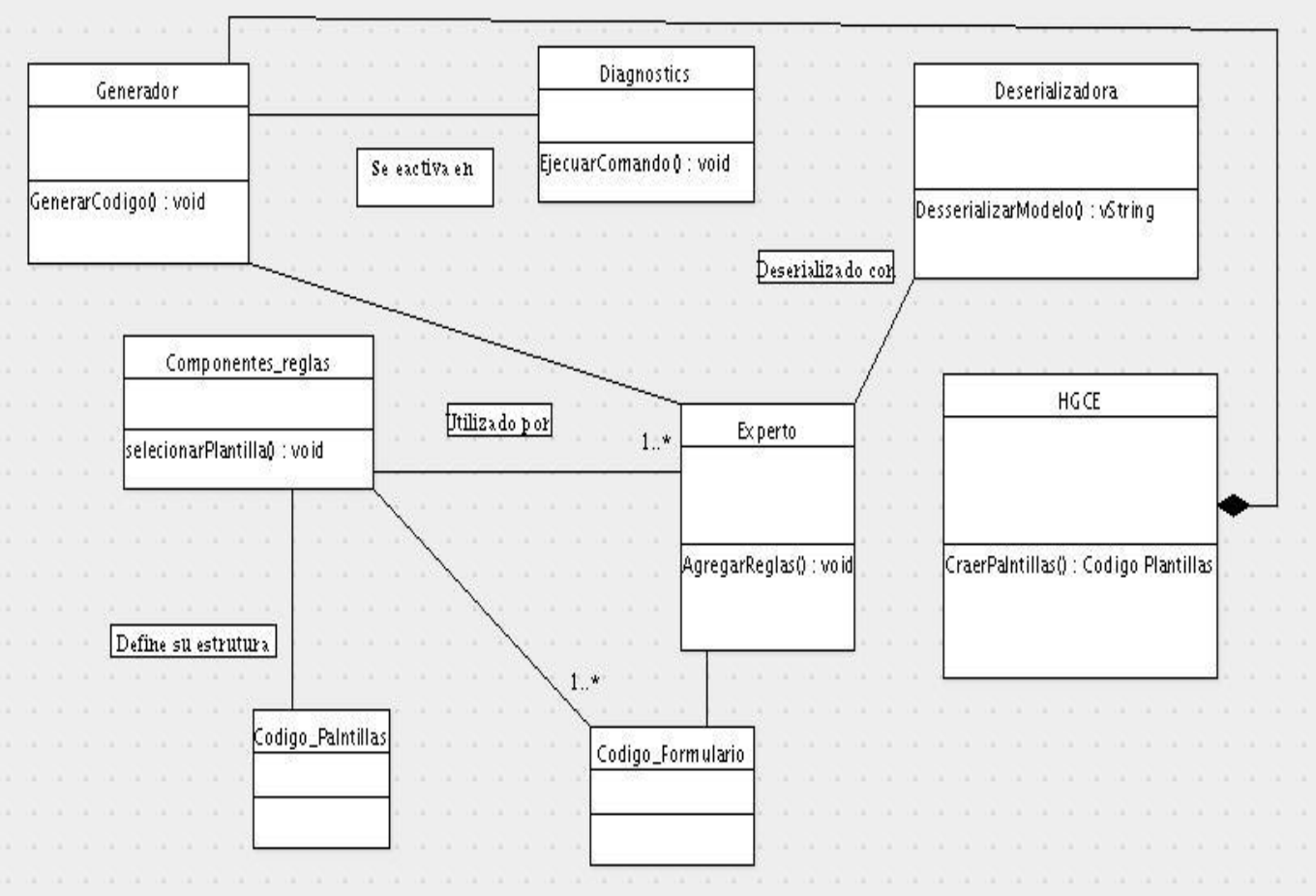


Fig. 15 Diagrama de diseño

Construcción de las funcionalidades.

La primera versión de la herramienta que se presenta en el pretende solamente dar cumplimiento inmediato a los objetivos de la tesis. El resto de las funcionalidades que poseen otras herramientas CASE podrían incorporarse en el futuro del proyecto. El siguiente diagrama muestra una vista de la arquitectura creada para la implementación de la herramienta

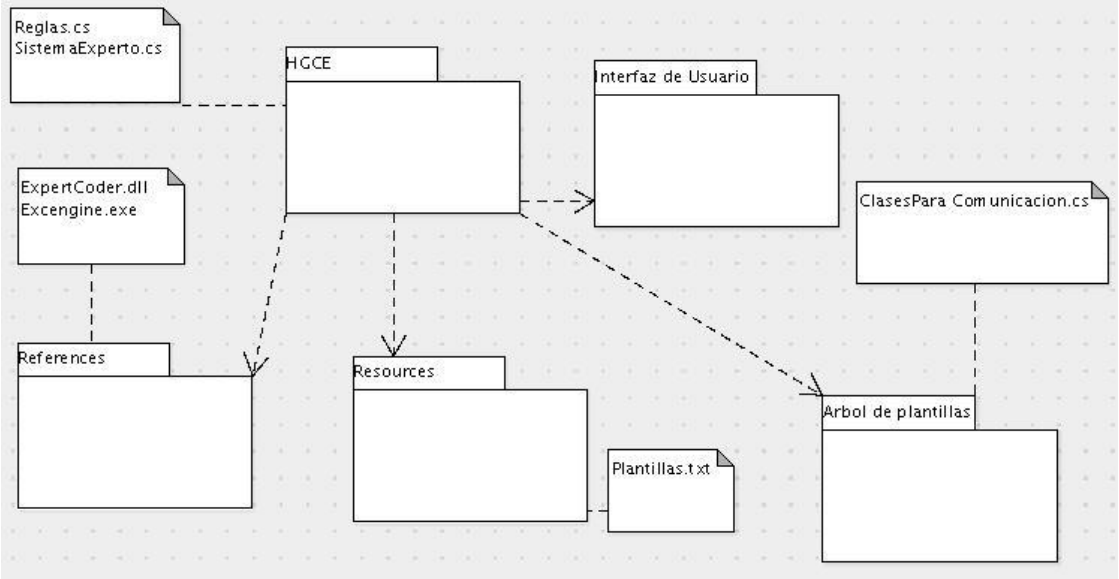


Fig. 16 Arquitectura de HGCE

4.2 Diagrama de Componentes

El siguiente diagrama de componentes muestra la utilización de las dlls de la biblioteca de clases ExpertCoder utilizadas para la construcción del sistema experto, así como el ejecutable que coordina las funcionalidades.

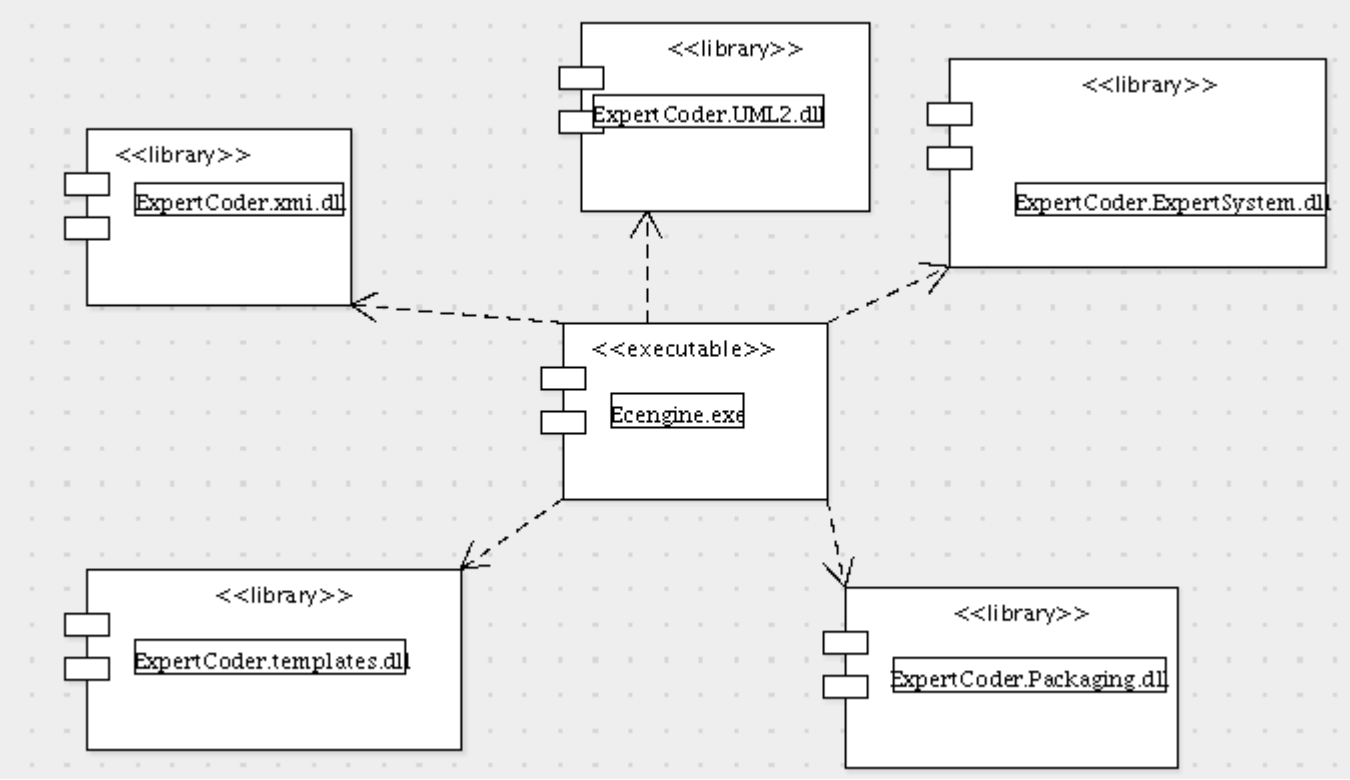


Fig. 17 Diagrama de componentes externos

4.3 Cargar modelo UML.

Como hemos mencionado anteriormente, el modelo que toma nuestra herramienta para generar el código fuente tiene extensión UML. El proceso de obtención de este modelo y la obtención de sus propiedades dentro del generador es el principal objetivo de este epígrafe.

Para lograr estos propósitos utilizamos la librería de clases nUML, inicialmente parte del proyecto ExpertCoder, actualmente es un proyecto de código abierto totalmente independiente.

Al accionar la opción cargar modelo, la herramienta permite seleccionar el modelo del cual queremos obtener el código fuente para ubicarlo en la carpeta de modelos que está dentro de las carpetas que contienen el código de la herramienta. Esto permite tener un camino fijo en los ficheros de configuración a la hora de especificar de donde el generador tomará el modelo, el modelo está serializado con XMI versión 2.0, que soporta XML, lo que garantiza que se puedan serializar estos modelos en diferentes herramientas de modelado, así como posibilitar el intercambio. El **Apéndice A**, muestra un ejemplo del código de estos modelos.

4.4 Crear plantillas de texto plano.

Las plantillas de texto plano son un importante recurso que esta herramienta utiliza. Tienen la responsabilidad de contener el código que se quiere generar en un lenguaje de programación seleccionado por el desarrollador. Constituyen la base del modelo de estructura de objetos sobre la cual trabaja el sistema experto basado en reglas. Cada vez que se necesite incorporar un nuevo componente visual a la base de conocimiento de nuestro sistema experto tendremos que definir su código en una plantilla de texto plano, esta opción la brinda la propia herramienta, que permite guardarla en la propia carpeta Template, donde se encuentran el resto de las plantillas de los demás componentes. En el **Apéndice B** se muestra un ejemplo de la definición de varios componentes que están en nuestra herramienta. Donde los marcadores $NombreCampo$, $Etiqueta$ Y $Literales* contienen el nombre del campo que define el componente, el texto que se presentará al usuario en su etiqueta y una lista de cadenas de caracteres, cada una de las cuales representa un posible valor. En el **Apéndice E**, se muestra la clase que conecta la regla que crea el formulario con la plantilla de texto plano que la define.

4.4.1 Reparar plantillas texto plano del generador.

En ocasiones es necesario redefinir el código fuente que queremos generar; actualmente la herramienta tiene que utilizar prototipos de los IDE seleccionados por el equipo de proyecto para generar código parecido al del IDE que utilice el equipo de desarrollo. Una vez seleccionado el entorno de desarrollo integrado se desglosan las plantillas teniendo en cuenta los patrones de codificación del software

mencionado. Habitualmente este código puede tener algunas inconsistencias e imperfecciones que si no es problema de las decisiones del sistema experto se pueden reparar arreglando el patrón de código con el cual se escribió la plantilla y sus marcadores.

4.5 Creando las reglas del generador de la herramienta.

Crear la regla del sistema experto significa escribir el generador que constituye el corazón de esta herramienta de generación de código fuente. En el **Apéndice C** se muestra como se define el sistema experto haciendo uso de la librería de clases ExpertCoder. Los componentes constituyen los atributos de nuestro modelo de entrada, en el **Apéndice F** se muestra como se crea la regla que se activa cuando aparece un ComboBox en nuestra entrada.

Pero para definir las reglas se tiene en cuenta un principio de los sistemas expertos donde el conocimiento y la forma de usar ese conocimiento para deducir conclusiones están separados. Un sistema experto en el ámbito de esta biblioteca está compuesto por reglas y precedencias entre reglas. En estas reglas, la condición de activación es función del tipo de datos del elemento actual del modelo de entrada y, opcionalmente, del de salida. Estos tipos de datos se indican en el constructor, el **Apéndice D** representa el código de una regla para definir la estructura de código de un formulario, sin agregarle aun sus componentes.

4.6 Generando código fuente.

La herramienta, para correr el generador, hace uso de la aplicación *ecengine.exe*; que se activa al seleccionar al opción generar del software desarrollado. Esta aplicación toma como entrada un fichero XML de configuración, donde se especifica la tarea a ejecutar. En este fichero se indican el proveedor de modelo, la ubicación del modelo, el generador a usar y sus parámetros.

Utilizando System.Diagnostics se inicia un nuevo proceso que permite pasar un path con la aplicación ecengine.exe y el fichero XML que contiene la información de nuestro generador. Esta habilidad de C# permite pasar como parámetro de tipo string un path con definición de comandos en sus caracteres.

Conclusiones

Muchos son los caminos mediante los cuales los desarrolladores pueden simplificar su trabajo y disminuir el valor de la poderosa variable tiempo en los procesos de desarrollo del software. Utilizarlos eficazmente posibilita una abstracción en la programación necesaria para las técnicas actuales de desarrollo.

Es importante a la hora de crear un generador definir el formato del modelo de entrada, para, a partir de este, definir incluso los lineamientos de la arquitectura que utilizará el generador. Las herramientas de generación de código actuales implementan variadas funcionalidades que permiten propósitos deseados por un equipo de desarrollo. Incorporarle un sistema experto a la gestión de sus funcionalidades garantiza un nuevo paso de avance en el campo de la generación de código; sin necesidad de que las reglas de ese sistema tengan que verificarse para un lenguaje en específico de programación.

Luego de detallar las funcionalidades de esta herramienta se pudiese incluir a esta versión de HGCE dentro del grupo de las L-CASE, o sea una herramienta CASE inferior (se dedica a la codificación) que mejora la productividad a corto plazo, sin lograr la eficiencia, en el análisis y el diseño de los proyectos; y sin integrar el ciclo de vida del software. Esta herramienta podría decirse que reafirma uno de los propósitos de todas los proyectos de software asistido por computadoras, que es el de lograr que los analistas tengan más tiempo para el análisis y diseño disminuyendo el tiempo para la codificación en el equipo de desarrollo.

Con ésta herramienta será posible generar una versión inicial de la interfaz gráfica de una aplicación, aunque con seguridad las ventanas serán modificadas en el futuro en la medida en que se planteen los requisitos del sistema. Es muy difícil establecer un esquema para que cuando cambie el modelo se pueda volver a generar el código sin perder las modificaciones hechas por el programador. Por eso podemos asegurar que esta primera versión no tiene en cuenta el mantenimiento.

Con los resultados de la herramienta que presenta el proyecto; se asegura una estructura consistente. Se implementan patrones de traducción más rigurosos, se disminuye la ocurrencia de varios tipos de errores en la codificación manual y se garantiza la reutilización, puesto que se puede volver a utilizar el software y la estructura para generar otros códigos, debido al poder que brinda la concepción del mismo mediante Sistemas Expertos.

.

Recomendaciones

La meta del proyecto es extensa. Obtener un generador de código capaz de obtener de un conjunto de modelos; el código del sistema completo, podría ser un objetivo futuro. A ello podría sumársele la incorporación de los mecanismos de compilación de código para aumentar los servicios al usuario. Un generador por capa perfeccionaría a la HGCE. Varios generadores de código pudiesen ayudar en el propósito. Para ello habría que lograr una arquitectura de código donde todo se conjugue a la perfección y permitan espacio también a la labor del programador. Cuando se logre minimizar al máximo los casos donde se cree solo la versión inicial de un fichero que el programador tiene que modificar, entonces se podrá incorporar las interacciones automáticas entre el código generado y el escrito por el experto humano, el programador.

Apéndice A

```
<?xml version="1.0" encoding="utf-8"?>
<xmi: XMI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmi:version="2.1"
xmlns: uml="http://schema.omg.org/spec/uml/2.0"
xmlns:xmi="http://schema.omg.org/spec/xmi/2.1">

 <uml:Profile xmi:id="1" name="SampleProfile">

   <ownedMember xmi:type="uml: Stereotype" xmi:id="_Remote"
name="Remote">
     <ownedAttribute xmi:id="5" name="base_Interface"
visibility="private" type="_UML_Interface" association="3" />
   </ownedMember>

   <ownedMember xmi:type="uml:Extension" xmi:id="3" name="Remote extends
Interface" memberEnd="5">
     <ownedEnd xmi:id="4" name="extension_Remote" visibility="private"
lower="0" type="_Remote" />
   </ownedMember>

 </uml:Profile>

 <uml:Model xmi:id="_UML" name="UML">
   <ownedMember xmi:type="uml:Class" xmi:id="_UML_Interface"
name="Interface" />
 </uml:Model>

</xmi:XMI>
```

Apéndice B

```
// fichero: DefiniendoCheckButton.txt
// $Etiqueta$
$NombreCampo$ = AddCheckButton(components, sizeGroup, "$Etiqueta$:");
```

```
// fichero: DefiniendoComboBox.txt
// $etiqueta$
$NombreCampo$ = AddComboBox(
      components, sizeGroup, "$etiqueta$:",
      new string[] {$Literals*}
);
```

```
// fichero: DefieniendoEntry.txt
// $Etiqueta$
$NombreCampo$ = AddEntry(components, sizeGroup, "$Etiqueta$:");
```

Apéndice C

Fichero:Expert.cs

```
using LibreriaES = ExpertCoder.ExpertSystem;

namespace ECTutorial.SampleGenerator
{
	[LibreriaES.ExpertSystemInformation(
		"Generador de Formularios",
		"Genera codigo fuente de clases de formularios",
		true)]
	public class Expert : LibreriaES.Expert
	{
		public Expert(LibreriaES.Environment env) : base(env)
		{
			LibreriaES.Rule AtributoEntry = new ReglaEntry();
			LibreriaES.Rule AtributoComboBox = new
ReglaComboBox();
			LibreriaES.Rule AtributoCheckButton = new
ReglaCheckButton();
			base.AddRule( new ReglaForma() );


			base.AddRule( new DeserializarModelo() );
			base.AddRule( new ExtraerPackage() );
			base.AddRule( AtributoEntry );
			base.AddRule( new EscanearClass() );
			base.AddRule( AtributoComboBox );
			base.AddRule( AtributoCheckButton );


			base.AddPrecedence(
				new LibreriaES.Precedence(
					AtributoComboBox, AtributoEntry,
LibreriaES.PrecedenceKind.Replacement ) );
			base.AddPrecedence(
				new LibreriaES.Precedence(
					AtributoCheckButton, AtributoEntry,
LibreriaES.PrecedenceKind.Replacement ) );
		}
	}
}
```

Apéndice D

```
// fichero: ReglaForma.cs
using System.IO;
using LibreriaES = ExpertCoder.ExpertSystem;
using UML = NUml.Uml2;
using Plantillas = ECTutorial.SampleGenerator.TemplateTree;

namespace ECTutorial.SampleGenerator
{
	public class ReglaForma : LibreriaES.TypeGuardedRule
	{
		public ReglaForma () : base(typeof(UML.Class))
		{ }

		[LibreriaES.Parameter("camino de salida", "directorio donde
seran creadas", true)]
		public string OutputPath;

		public override void Execute(LibreriaES.Environment env)
		{
			UML.Class cls = (UML.Class)env.CurrentInputElement;
			Plantillas.Form form = new Plantillas.Form();
			form.Caption = cls.Name;
			form.Name = cls.Name.Replace(" ", "");
			env.CurrentOutputElement = form;
			env.Expert.Process();
			StreamWriter sw;
			using( sw = new StreamWriter(Path.Combine(OutputPath,
form.Name + "Form.cs")) )
			{
				sw.WriteLine(form.ToString());
			}
		}
	}
}
```

Apéndice E

Ficero:DefiniendoForma.cs

```
using System;
using ExpertCoder.Templates;

namespace ECTutorial.SampleGenerator.TemplateTree
{
	public class DefiniendoForma : Template
	{
		private static string[] _phs
			= new string[] { "$Etiqueta$", "$Campo*", "$Nombre!",
"$WidgetsCreation*" };

		public DefiniendoForma() : base("Forma.txt",
typeof(DefiniendoForma).Assembly, _phs)
		{
			Fields = new TemplateCollection("\n");
			Fields.Tabs = 1;
			WidgetsCreation = new TemplateCollection("\n");
			WidgetsCreation.Tabs = 2;
		}

		public string Caption;
		public readonly TemplateCollection Fields;
		public string Name;
		public readonly TemplateCollection WidgetsCreation;

		protected override void SetPlaceHoldersValues()
		{
			base["$Etiqueta$"] = Name;
			base["$Campo*"] = Fields.ToString();
			base["$Nombre$"] = Name;
			base["$WidgetsCreation*"] =
WidgetsCreation.ToString();
		}
	}
}
```

Apéndice F

Fichero:ReglaCombobox.cs

```
using LibreriaES = ExpertCoder.ExpertSystem;
using UML = ExpertCoder.Uml2;
using  Plantillas  = ECTutorial.SampleGenerator.TemplateTree;

namespace ECTutorial.SampleGenerator
{
      public class ReglaComboBox : LibreriaES.Rule
      {
            public ReglaComboBox() : base(new
LibreriaES.InputElementInhibition())
            { }

            public override bool IsActiveInState(LibreriaES.Environment
env)
            {
                  UML.Property attrib = env.CurrentInputElement as
UML.Property;
                  return attrib != null
                        && attrib.Type != null
                        && attrib.Type is UML.Enumeration
                        && env.CurrentOutputElement is
Plantillas.Form;
            }

            public override void Execute(LibreriaES.Environment env)
            {
                  UML.Property attrib =
(UML.Property)env.CurrentInputElement;
                  Plantillas.Form form =
(Plantillas.Form)env.CurrentOutputElement;
                  string fieldName = "_" + attrib.Name.Replace(" ", "")
+ "ComboBox";
                  // recojo los datos para l aplantilla del combobox
                  Plantillas.DefiniendoComboBox entryCreation = new
Plantillas.DefiniendoComboBox();
                  entryCreation.Caption = attrib.Name;
                  entryCreation.FieldName = fieldName;
                  UML.Enumeration enumeration =
(UML.Enumeration)attrib.Type;
                  foreach(UML.EnumerationLiteral literal in
enumeration.OwnedLiteral)
                  {
```

```
                        entryCreation.Literals.Add("\"" + literal.Name
+ "\"");
                }
                form.WidgetsCreation.Add(entryCreation);
                // definiendo el campo del combobox
                Plantillas.DefiniendoCampo definicion = new
Plantillas.DefiniendoCampo();
                definicion.FieldName = fieldName;
                definicion.FieldType = "ComboBox";
                form.Fields.Add(definicion);
            }
        }
}
```

Bibliografía

Alonso, Enrique Barreiro. 2007. *Modelo de Dominio.* s.l. : Universidad de Vigo, 2007.

Booch, Grady, Jacobson, I y Raumbaugh, J. 2000. *El Lenguaje de Modelado Unificado.* Madrid : Pearson Education,SA, 2000.

Buschmann, y otros. 1996. *Pattern-Oriented Software Architecture: A System of patterns (POSA).* 1996.

Campero, Rodolfo. 2005. SourceForge.net. *An NDoc Document Class Library.* [En línea] 2005. [Citado el: 20 de Diciembre de 2007.] http://expertcoder.sourceforge.net/docs/.

—. 2007. SOURCEFORGE.NET. [En línea] 11 de Octubre de 2007. [Citado el: 22 de Diciembre de 2007.] http://numl.sourceforge.net/index.php/Main Page.

Cárdenas Fernández, C, Cerdán Cruz, M A y Puma Gamboa, A. 2000. *BORLAND TOGETHER Herramientas CASE.* 2000.

De Lara, J. 2001. *Modelado de Software con UML.* Madrid : s.n., 2001.

Gamma, E, y otros. 1995. *Design Patterms:Elements of reusable Object Oriented Software.* 1995.

Gutiérrez, J M. 2007. *Sistemas Expertos Basados en Reglas.* s.l. : Departamento de matemática aplicada Universidad de Cantabria, 2007.

Hart, A. 1992. *Knowlege acquisition for expert system.* New york : McGraw- Hill, 1992.

Hayes-Roth, F, Waterman, D y Lenat, D. 1983. *Building Expert System.* 1983.

IEEE. 1993. s.l. : IEEE Standard 610.12.1990, 1993.

Instituto Nacional de cultura informática. 2006. *Herramientas CASE.* Lima : s.n., 2006.

Jimenez, Pedro Aurelio Llamas. 2004. *Generacion Automatica de Codigo para modelos de componentes dritribuidos Corba LC.* 2004. Universidad de Murcia

Management, Group Object. 2007. WWW.omg.org. [En línea] 2007. [Citado el: 2 de Marzo de 2007.] http://www.omg.org/spec/XMI/2.1/PDF.

Microsoft Corporation. 2000. *C Sharp Specification.* 2000.

Moisés Daniel Díaz. 2007. *Arquitecturas Refexivas y la Generación de Código.* [En línea] 2007. http://Moisesdaniel.com.

Odutola, K, Oguntimehin, A y der Wulp, Van. WWW.Teologic.com. [En línea] [Citado el: 26 de 10 de 2007.] http://www.teologic.com/products/systemarchitect/index.cfm.

Presman, R S. 1993. *Un enfoque práctico.* s.l. : McGraw Hill España, 1993.

Rojas, Juan Carlos Olivares. 2006. *Introducción a la programación de sistemas.* 2006.

Shumller, J. 2002. *Aprendiendo UML en 24 Horas.* Mexico : Pearson Education SA, 2002.

2006. *Automted Code Generation.* [En línea] 2006. http://c2.com/cgi/wiki?AutomatedCodeGeneration.

2007. Code Generation is a Design Smell. [En línea] 2007. http://c2.com/cgi/wiki?CodeGenerationIsaDesignSmell.

Reflection versus Code Generation. [En línea] http://www.javaworld.com/javaworld/jw-11-2001/jw-1102-.

2008. WWW.MSDN2.microsoft.com. [En línea] 2008. [Citado el: 9 de Enero de 2008.] http://msdn2.microsoft.com/es-es/182eeyhh(VS.80).aspx.

Referencias bibliográficas

[1] **Booch, Grady, Jacobson, I y Raumbaugh, J. 2000.** *El Lenguaje de Modelado Unificado.* Madrid : Pearson Education,SA, 2000.

[2] **Presman, R S. 1993.** *Un enfoque práctico.* s.l. : McGraw Hill España, 1993.

[3] **IEEE. 1993.** s.l. : IEEE Standard 610.12.1990, 1993.

[4] **Instituto Nacional de cultura informática. 2006.** *Herramientas CASE.* Lima: s.n., 2006.

[5] **Odutola, K, Oguntimehin, A y der Wulp, Van.** WWW.Teologic.com. [En línea] [Citado el: 26 de 10 de 2007.] http://www.teologic.com/products/systemarchitect/index.cfm.

[6] **Cárdenas Fernández, C, Cerdán Cruz, M A y Puma Gamboa, A. 2000.** *BORLAND TOGETHER Herramientas CASE.* 2000.

[7] **Gamma, E, y otros. 1995.** *Design Patterms:Elements of reusable Object Oriented Software.* 1995.

[8] **De Lara, J. 2001.** *Modelado de Software con UML.* Madrid: s.n., 2001.

[9] **Booch, Grady, Jacobson, I y Raumbaugh, J. 2000.** *El Lenguaje de Modelado Unificado.* Madrid : Pearson Education,SA, 2000.

[10] **Shumller, J. 2002.** *Aprendiendo UML en 24 Horas.* Mexico: Pearson Education SA, 2002.

[11] **Gutiérrez, J M. 2007.** *Sistemas Expertos Basados en Reglas.* s.l. : Departamento de matemática aplicada Universidad de Cantabria, 2007.

Índice de Figuras