



Implementación de herramienta case de generación de código java/sql a partir de diagramas de clase UML

Rodriguez Samanez Roger Hans

Orientador: Prof Victor Bustamante

*Tesis profesional presentada a la Escuela Académico
Profesional de Ingeniería de Sistemas como parte de los
requisitos para obtener el Título Profesional de Ingenie-
ro de Sistemas.*

Universidad Nacional Mayor de San Marcos
Junio de 2016

Aquí deberás colocar a quien va dedicada tu tesis por ejemplo: A Dios, por todo lo que me ha dado, a todos los profesores por sus enseñanzas y algunos amigos.

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS
ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA
DE SISTEMAS

Implementación de herramienta case de generación de
código java/sql a partir de diagramas de clase UML

Tesis de graduación presentado por el bachiller Rodriguez Samanez
Roger Hans en el cumplimiento de los requisitos para obtener el
título profesional de Ingeniero de Sistemas.

Lima, 26 de junio de 2016

Aprobado por:

Prof. Dr. Paterno1 Materno1
Nombres1
PRESIDENTE

Prof. Dr. Paterno2 Materno2
Nombres2
SECRETARIO

Prof. Dr. Paterno3 Materno3
Nombres3
INTEGRANTE

Prof. Dr. Paterno4 Materno4
Nombres4
EXTERNO
Universidad del ABC

Abreviaturas

UML *Unified Modeling Language*

IEEE *Institute of Electrical and Electronics Engineers*

ONGEI *Oficina Nacional de Gobierno Electrónico e Informática*

Agradecimientos

Agradezco en primer lugar a Dios, a mis padres que siempre he contado con ellos y a cada uno de mis mentores que les agradezco por dejarme aprender de ellos.

Resumen

Abstract

Índice general

1. Introducción	2
1.1. Antecedentes ó Realidad Problemática	2
1.2. Definición del Problema	2
1.3. Objetivos	2
1.3.1. Objetivos Específicos	2
1.4. Justificación	2
1.5. Alcances	2
2. Marco Teórico	3
2.1. Herramientas CASE	3
2.1.1. Historia de las herramientas CASE	3
2.1.2. Tipos de CASE	4
2.2. El Lenguaje de Modelado Unificado	6
2.2.1. Introducción	6
2.2.2. Las vistas de UML	8
2.2.3. Diagrama de Clases	9
2.3. Patrón DAO	10
3. Aporte Metodológico	13
Bibliografía	14

Índice de tablas

2.1. Elementos estructurales de un caso de uso	9
2.2. Relaciones dentro de un caso de uso	10
2.3. Relaciones dentro de un caso de uso	11
2.4. Relaciones dentro de un caso de uso	12

Índice de figuras

2.1. Historia de las Herramientas CASE	5
2.2. Tipos de Case	6
2.3. Clasificación de vistas y diagramas de UML	8

Capítulo 1

Introducción

1.1. Antecedentes ó Realidad Problemática

1.2. Definición del Problema

1.3. Objetivos

1.3.1. Objetivos Específicos

1.4. Justificación

1.5. Alcances

El alcance se refiere a las grandes actividades que se va desarrollar en la tesis. Estas son por lo general: - Estudio del arte del problema (revisión de todo lo que existe sobre el problema de la tesis). - Análisis, desarrollo y puesta en marcha de una solución tecnológica (software). - Prueba del sistema. Es deseable que la propuesta sea verificada en una organización, el cual constituye el estudio de caso. En este sentido se deberá indicar la organización (empresa, institución, industria, sector de gobierno, etc.) que será beneficiada.

Capítulo 2

Marco Teórico

2.1. Herramientas CASE

La Ingeniería del software es la ciencia que ayuda a elaborar sistemas con el fin de que sea económico, fiable y funcione eficientemente sobre las máquinas reales [8]. El uso de la Ingeniería del Software trae consigo algunas ventajas como: obtención de un nivel competitivo, mejora de la uniformidad de métodos, adaptación de la automatización del analista, cambio de métodos de trabajo, entre otros.

La *Institute of Electrical and Electronics Engineers* (IEEE) define en 1990 a la Ingeniería del Software como: "La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento de software"[4].

Dos definiciones de CASE formales son las siguientes:

"Herramientas individuales para ayudar al desarrollador de software o administrador de proyectos durante una o más fases del desarrollo (o mantenimiento) del software"[11].

Una combinación de herramientas de software y metodologías estructuradas de desarrollo"[6].

Luego entonces, la Ingeniería de Software Asistida por Computadora (CASE) tiene como objetivo proporcionar un conjunto de herramientas bien integradas y que ahorren trabajo, uniendo y automatizando todas o algunas de las fases del ciclo de vida del software, en otras palabras, CASE es una herramienta que ayuda a un ingeniero de software a desarrollar sistemas de cómputo.

2.1.1. Historia de las herramientas CASE

La historia de las herramientas CASE comienza a principios de los 70's con el procesador de palabras que se usaba en la creación de documentos. El desarrollo se centró inicialmente en herramientas de soporte de programas como traductores, compiladores, ensambladores y procesadores de macros [12]. Dado que la tecnología avanzó y el soft-

ware se volvió más complejo, el tamaño para el soporte también tuvo que crecer. Ahora se desarrollaban herramientas para diagramas (como los diagramas Entidad-Relación y diagramas de flujo), editores de programas, depuradores, analizadores de código y utilidades de impresión, como los generadores de reportes y documentación. Los desarrollos en el área de las herramientas CASE entre 1980 y 1990 tuvieron un enfoque hacia las herramientas que buscaban dar respuestas a los problemas en los sistemas de desarrollo. Esto dió inicio a la elaboración de los siguientes productos de herramientas CASE:

- Desarrollo Orientado a Objetos: Éstas ayudan a crear código reutilizable que pueda ser utilizado en diferentes lenguajes y plataformas. Con el gran crecimiento de desarrollos actual, este tipo de herramientas continúa incrementándose.
- Herramientas de desarrollo Visual: Estas herramientas permiten al desarrollador construir rápidamente interfaces de usuario, reportes y otras características de los sistemas, lo que hace posible que puedan ver los resultados de su trabajo en un instante, un ejemplo de estas herramientas son los lenguajes de programación visuales como Borland C++ Builder, Visual C++, entre otros.

Las actuales líneas de evolución de las herramientas CASE de acuerdo [10] son:

- Herramientas para sistemas bajo arquitectura cliente/servidor. Versiones que faciliten la distribución de los elementos de una aplicación entre los diferentes clientes y servidores.
- CASE multiplataforma. Herramientas que soportan combinaciones de diferentes plataformas físicas, sistemas operativos, interfaces gráficas de usuario, sistemas de gestión de bases de datos, lenguajes de programación y protocolos de red.
- CASE para ingeniería inversa y directa. Ya existen algunas herramientas de este tipo como IBM Rational Rose. Su evolución serán mejoras en la obtención de los diseños a partir del código ya existente (ingeniería inversa) y la regeneración del mismo (ingeniería directa).
- CASE para trabajo en grupo (groupware). Herramientas que se centran en el proceso de desarrollo más que en el producto a desarrollar.
- CASE para desarrollo de sistemas Orientados a Objetos. Casi todas las herramientas existentes cubren alguna de las fases del ciclo de vida de desarrollo de aplicaciones orientadas a objetos, ya sea la interfaz del usuario, análisis, diseño, programación, etc. Ahora el objetivo es cubrir el ciclo de vida completo.

La historia de las herramientas de software se resume en 2.1

2.1.2. Tipos de CASE

Según *Oficina Nacional de Gobierno Electrónico e Informática* (ONGEI) no existe una única clasificación de herramientas CASE y, en ocasiones, es difícil incluirlas en una clase determinada. Podrían clasificarse atendiendo a:

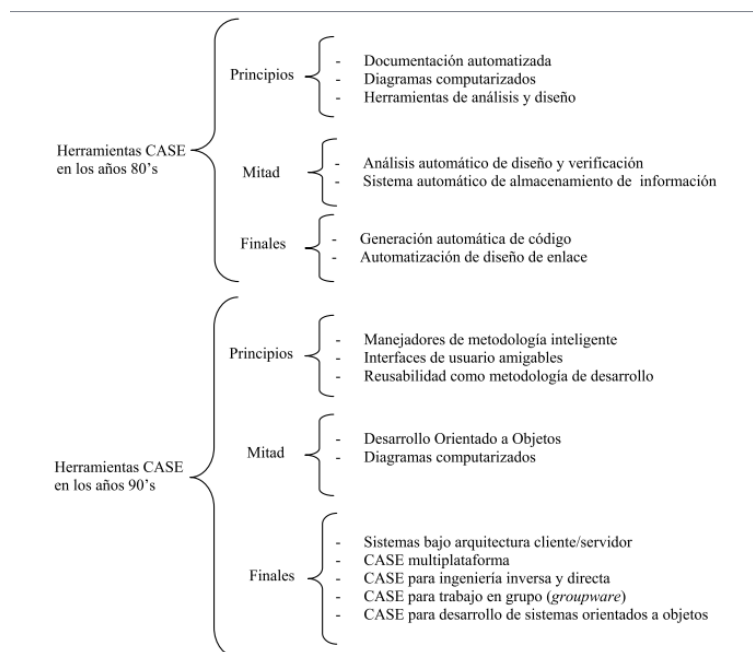


Figura 2.1: Historia de las Herramientas CASE

- Las plataformas que soportan.
- Las fases del ciclo de vida del desarrollo de sistemas que cubren.
- La arquitectura de las aplicaciones que producen.
- Su funcionalidad.

Además según ONGEI en función de las fases del ciclo de vida abarcadas, las herramientas CASE se pueden agrupar 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.
- Herramienta(s) que comprende(n) alguna(s) fase(s) del ciclo de vida de desarrollo de software:
- 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.
- Herramientas de bajo nivel, L-CASE (Lower CASE - CASE inferior) o back-end, dirigidas a las últimas fases del desarrollo: construcción e implantación.
- Juegos de herramientas o toolkits, 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.

Además según [7] se puede concluir el siguiente cuadro: 2.2

Tipo de Case	Ventajas	Desventajas
Upper Case	<ul style="list-style-type: none"> Se utiliza en plataforma PC, es aplicable a diferentes entornos. Menor costo 	<ul style="list-style-type: none"> Permite mejorar la calidad de los sistemas, pero no mejora la productividad. No permite la integración del ciclo de vida.
Lower Case	<ul style="list-style-type: none"> Permite lograr importantes mejoras de productividad a corto plazo. Permite un eficiente soporte al mantenimiento de sistemas. 	<ul style="list-style-type: none"> No garantiza la consistencia de los resultados a nivel corporativo. No garantiza la eficiencia del Análisis y Diseño. No permite la integración del ciclo de vida.
I-Case	<ul style="list-style-type: none"> Integra el ciclo de vida Permite lograr importantes mejoras de productividad a mediano plazo. Permite un eficiente soporte al mantenimiento de sistemas. Mantiene la consistencia de los sistemas a nivel corporativo. 	<ul style="list-style-type: none"> No es tan eficiente para soluciones simples, sino para soluciones complejas. Depende del Hardware y del Software. Es costoso.

Figura 2.2: Tipos de Case

2.2. El Lenguaje de Modelado Unificado

2.2.1. Introducción

La Ingeniería del Software ha tratado con el paso del tiempo simplificar cada vez más las complejidades que presentan el análisis y diseño de sistemas de software. Para atacar un problema muy grande es bien sabido que se hace uso de la descomposición, para el caso de la Ingeniería del Software se puede hacer una descomposición ya sea algorítmica u Orientada a Objetos, dónde esta última es la de más tendencia actualmente.

Debido a esta problemática los investigadores de la Ingeniería del Software han desarrollado diversas metodologías Orientadas a Objetos con la finalidad de proporcionar un soporte para los desarrolladores de sistemas para analizar y diseñar con más precisión los sistemas. Martin Fowler afirma que actualmente el *Unified Modeling Language* (UML) es considerado por muchos autores incluyendo a sus creadores James Rumbaugh, Ivar Jacobson y Grady Booch como el lenguaje que unifica los métodos de Análisis y Diseño Orientados a Objetos. [3].

Qué es el UML

El UML es un lenguaje de modelado gráfico y visual utilizado para especificar, visualizar, construir y documentar los componentes de un sistema de software. Está pensado para poder aplicarse en cualquier medio de aplicación 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.

El UML 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 de 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. [5]

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. El UML sólo define 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 [3]. La idea de usar los diagramas creados mediante el UML es simplemente para mejorar la comunicación, porque ayuda a que los desarrolladores de software se comuniquen con un mismo lenguaje de modelado independiente de las metodologías empleadas [3]. La ventaja principal del UML [1] sobre otras notaciones OO es que elimina las diferencias entre semánticas y notaciones.

Antecedentes del UML

Antes que el UML, hubo muchos intentos por unificar métodos, el más conocido que se señala en [5] es el caso de Fusion por Coleman y sus colegas que incluyó conceptos de los métodos OMT y Booch, pero como los autores de estos últimos no estaban involucrados en la unificación fue tomado como otro método más. El primer acercamiento a UML fue en 1994 cuando se da la noticia de que Jim Rumbaugh se une con Grady Booch en Rational Software Corporation con la finalidad de unificar sus métodos OMT y Booch respectivamente. En 1995 salió a luz la primera propuesta de su método integrado que fue la versión 0.8 del entonces llamado Método Unificado (Unified Method). En ese mismo año Ivar Jacobson se une a Rational para trabajar con Booch y Rumbaugh en el proceso de unificación, a partir de aquí a estos tres personajes se les conoce como "Los tres amigos".

En 1996, los tres amigos concluyen su trabajo y lo nombran UML, es entonces cuando el OMG 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 \ddot{U} entre las que figuran IBM, Oracle y Rational Software \ddot{U} 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 afines como lenguaje de modelado estándar. En diciembre de 2002 IBM adquirió las acciones de Rational Software Corporation.

Conceptos básicos

Como ya se ha mencionado, el UML no es un método sino un lenguaje, el cual define únicamente una notación y un metamodelo [3]. 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,

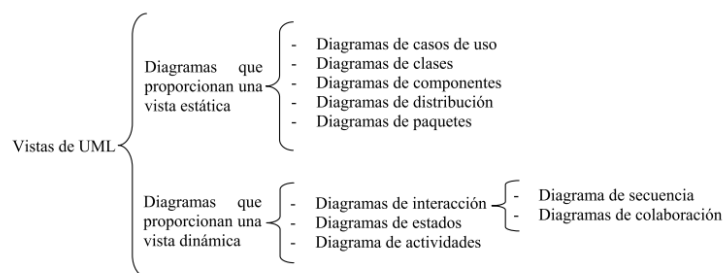


Figura 2.3: Clasificación de vistas y diagramas de UML

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 [3]. 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.

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.[3]

2.2.2. Las vistas de 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 [9]. Se muestra la tabla 2.3

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 [5].

Diagramas de Caso de Uso

Según [5] La vista que proporcionan los diagramas de casos de uso, modela 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.



 Actor	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. Por ejemplo, para el caso de uso de un sistema de ventas, el rol de “vendedor” lo puede hacer un vendedor tal cual, el dueño de la tienda, o incluso puede ser una máquina de autoservicio. Un actor se denota como una figura en forma de persona con su nombre debajo.
 Caso de uso	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.

Tabla 2.1: Elementos estructurales de un 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.

Elementos Estructurales

: Según [5] y [9] los elementos estructurales representan las partes físicas o conceptuales de un modelo. La tabla 2.1 muestra estos elementos.

Relaciones

: Según [5] y [9] las relaciones conectan a los elementos estructurales para darles sentido al diagrama de casos de uso. La tabla 2.2 muestra estas relaciones.

2.2.3. Diagrama de Clases

Según [5] y [9] 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





	Asociación La asociación es 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 caso de uso necesita utilizar a otro caso de uso. Es recomendable utilizar la inclusión cuando se repite un caso de uso más de una vez y se quiera evitar repeticiones. Se denota con una línea discontinua con una punta de flecha y con la palabra clave "include"
	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"

Tabla 2.2: Relaciones dentro de un caso de uso

- Generalizaciones
- Agregaciones
- Composiciones

Un diagrama de clases esta compuesto por los elementos mostrados en las tablas 2.3 y 2.4.

2.3. Patrón DAO

Según [2], cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces, sin hacer lo mismo dos veces.^{En} general un patrón tiene cuatro elementos esenciales:

- EL nombre del patrón que permite describir, en una o dos palabras, un problema de diseño junt con sus soluciones y consecuencias.
- El problema Describe cuando aplicar el patrón. Explica el problema y su context. Puede describir problemas concretos de diseño así como las estructuras de clases u objetos que son sintomáticas de un diseño inflexible. A veces el problema incluye una serie de condiciones que deben darse para que tenga sentido aplicar el patrón
- La solución, que describe los elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones.

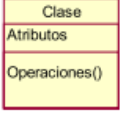

	<p>Clase Es la unidad básica que encapsula toda la información que comparten los objetos del mismo tipo. A través de una clase se puede modelar el entorno del sistema. En UML, una clase se representa por un rectángulo que posee tres divisiones: <i>Superior</i>: Contiene el nombre de la Clase <i>Intermedio</i>: Contiene los atributos que caracterizan a la Clase (pueden ser <i>private</i>, <i>protected</i> o <i>public</i>). <i>Inferior</i>: Contiene las operaciones, las cuales son la forma de cómo interactúa el objeto con los demás, dependiendo de la visibilidad (pueden ser <i>private</i>, <i>protected</i> o <i>public</i>).</p>
<p><i>sin imagen</i></p>	<p>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: <i>public</i> (pública), <i>protected</i> (protegida), <i>private</i> (privada). La sintaxis en UML para un atributo es: <i>visibilidad nombre : tipo = valor_inicial</i></p>
<p><i>sin imagen</i></p>	<p>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: <i>visibilidad nombre (parámetros) : tipo_de_retorno</i></p>
	<p>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.</p>

Tabla 2.3: Relaciones dentro de un caso de uso

- Las consecuencias son los resultados así como las ventajas e inconvenientes de aplicar el patrón.

Según [2], es bastante normal hacer aplicaciones que almacenan y recogen datos de una base de datos. Suele ser habitual, también, querer hacer nuestra aplicación lo más independiente posible de una base de datos concreta, de cómo se accede a los datos o incluso de si hay o no base de datos detrás. Nuestra aplicación debe conseguir los datos o ser capaz de guardarlos en algún sitio, pero no tiene por qué saber de dónde los está sacando o dónde se guardan. Hay una forma de hacer esto que ha resultado bastante eficiente en el mundo JEE y de aplicaciones web, pero que es aplicable a cualquier tipo de aplicación que deba recoger datos de algún sitio y almacenarlos. Es lo que se conoce como patrón DAO (Data Access Object). La idea de este patrón es sencilla. En primer lugar, debemos hacernos las clases que representan nuestros datos. Por ejemplo, podemos hacer una clase Persona con los datos de la persona y los métodos `set()` y `get()` correspondientes. Luego hacemos una interface. Esta interface tiene que tener los métodos necesarios para obtener y almacenar Personas. Esta interface no debe tener nada que la relacione con una base de datos ni cualquier otra cosa específica del medio de almacenamiento que vayamos a usar, es decir, ningún parámetro debería ser una *Connection*, ni un nombre de fichero, etc.



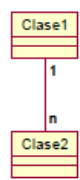

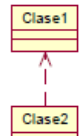
	<p>Agregación</p> <p>La agregación es un tipo especial de asociación, con la cual se pueden representar entidades formadas por varios componentes. La agregación es una relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye, en la programación OO se dice 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.</p>									
	<p>Composición</p> <p>La composición es similar a la agregación, ya que también representa entidades formadas por componentes, sólo que esta relación es estática, en donde el tiempo de vida del objeto incluido esta 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.</p>									
	<p>Multiplicidad</p> <p>Cada asociación tiene dos roles o papeles que se encuentran en cada extremo de la línea, cada uno de estos papeles tiene asignada una <i>multiplicidad</i>, 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:</p> <table><tr><td>1..*</td><td>(1..n)</td><td>Uno o más</td></tr><tr><td>*</td><td>(n)</td><td>Cero o más</td></tr><tr><td>m</td><td>(m es un entero)</td><td>Número fijo</td></tr></table> <p>Estos números se colocan sobre la línea de asociación junto a la clase correspondiente, como se ilustra.</p>	1..*	(1..n)	Uno o más	*	(n)	Cero o más	m	(m es un entero)	Número fijo
1..*	(1..n)	Uno o más								
*	(n)	Cero o más								
m	(m es un entero)	Número fijo								
	<p>Generalización (Herencia)</p> <p>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.</p>									
	<p>Dependencia</p> <p>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.</p>									

Tabla 2.4: Relaciones dentro de un caso de uso

Capítulo 3

Aporte Metodológico

Bibliografía

- [1] Derek Coleman, Viktor Ohnjec, John Artim, Erick Rivas, Jim Rumbaugh, and Rebecca Wirfs-Braclé. Uml (panel): The language of blueprints for software? *SIGPLAN Not.*, 32(10):201–205, October 1997.
- [2] John Vlissides Erich Gamma, Richard Helm Ralph Johnson. *Patrones de Diseño*. Addison wesley, 2003.
- [3] Martin Fowler. *UML gota a gota*. Pearson Educación, 1999.
- [4] IEEE. Ieee standard 610.12-1990. Technical report, The institution that published, 1990.
- [5] I. Jacobson J. Rumbaugh and G. Booch. *The title of the work*. El Lenguaje Unificado de Modelado Ú Manual de Referencia, S.A. Madrid, 2000.
- [6] Carma McClure. The case experience. *BYTE*, 14(4):235–244, April 1989.
- [7] ONGEI. Herramientas case. 2011.
- [8] Roger S. Pressman. *Ingeniería del Software: Un enfoque práctico*, Editorial: McGraw Hill España. McGraw Hill España, 1993.
- [9] Joseph Schmuller. *Aprendiendo UML en 24 horas*. Prentice Hall, 2000.
- [10] Simon Stobart. The case tool home. 1996.
- [11] B. Terry and D. Logee. Terminology for software engineering environment (see) and computer-aided software engineering (case). *SIGSOFT Softw. Eng. Notes*, 15(2):83–94, April 1990.
- [12] K Wayne. Computer Aided Software Engineering (CASE). 1987.