Universidad de las Ciencias Informáticas
"Facultad 3"

Título: "Modelo para la ayuda a la toma de decisiones en la selección de patrones de desarrollo de software."

Trabajo de Diploma para optar por el Título de

Ingeniero en Ciencias Informáticas

Autor: "Madelys Cuesta Villa"

Tutor: "Dr.C Pedro Yobanis Piñero Pérez"

Mayo, 2007

DECLARACIÓN DE AUTORÍA

Yo: **Madelys Cuesta Villa** me declaro como única autora de este trabajo y autorizo a la infraestructura Productiva de la Universidad de las Ciencias Informáticas (UCI) para que hagan el uso del mismo de la manera que estimen conveniente.

Y para que así conste firmo la presente a los ___ días del mes de _____ del 2007.

Y para que así conste firmo la presente a los 16 días del mes de Mayo del 2007.

_____ _____

Firma del Autor Firma del Tutor

Dedicatoria

A mis padres, hermana, abuelos y seres queridos, por su apoyo constante y por sus incontables enseñanzas y valores.

A ti kiosmy, por tu cariño, apoyo y consejos, por estar siempre presente cuando te necesito.

El triunfo no está en vencer siempre, sino en nunca desanimarse.

Agradecimientos

Agradezco a todo el que de una forma u otra me ayudado a salir adelante, especialmente le agradezco a:

Mi tutor Pedro Yobanis Piñero Pérez por sus valiosas ideas, comentarios y ayuda aportados para el desarrollo de la tesis, por su amabilidad y sobre todo su disponibilidad para atenderme a discutir temas de la tesis.

Al grupo Sherpa por haberme brindados sus conocimientos y experiencias sobre uno de los temas desarrollados en la tesis, y especialmente quisiera agradecer a kiosmy por su ayuda constante.

A mis amigos de siempre: Yule, Marien, Yainelys por ayudarme en múltiples situaciones y estar siempre ahí cuando las necesitaba.

A mis compañeros del IPI de Cienfuegos por ayudarme a en esos momentos tan difíciles salir adelante, especialmente quiero agradecer a mi amiga Rachel por todo el apoyo brindado en este tiempo.

A Rosa, amiga de la familia por brindarme su casa por más de seis meses para poder realizar la tesis, sin su ayuda no hubiera logrado terminar a tiempo este proyecto, por eso le estoy eternamente agradecida.

A los estudiantes del proyecto Proyecto DataPcLabs por su ayuda, en especial al estudiante Jose Angel.

A mis padres por apoyarme siempre y ayudarme a salir adelante en los obstáculos que se me presentaron.

Y por último y no por ser el último es el menos importante sino todo lo contrario, quiero agradecer a la Revolución por haberme dado la posibilidad de estudiar y llegar a ser hoy lo que soy.

Resumen

Los patrones de diseño, es un tema muy tratado por programadores, diseñadores y arquitectos de software pues es una de las claves principales para la obtención de un buen software. Estudios realizados demuestran la necesidad de la aplicación de los mismos en el desarrollo de cualquier aplicación a construir, pero estos estudios también demuestran que no todos los desarrolladores, diseñadores y arquitectos cuentan con los conocimientos necesarios para la aplicación de los mismos, debido fundamentalmente al grado de inmadurez de estos. Debido a esto se ha querido con esta tesis elaborar un modelo que basado en técnicas de inteligencia artificial sea capaz de tomar decisiones ante determinadas situaciones, donde por sus características sea necesario aplicar determinado patrón.

Para lograr la creación del modelo esta tesis investiga acerca de la aplicación de patrones de diseños en el desarrollo de arquitectura del software y de los sistemas basados en la ayuda a la toma de decisiones en situaciones específicas donde se apliquen estos patrones. Con la presentación del modelo, se permite caracterizar los patrones según sus características específicas y basado en estas características permite la ayuda a la toma de decisiones para aplicar los mismos en situaciones específicas de desarrollo de software. Como parte del modelo desarrollado se presentan algoritmos que permitirán realizar el proceso de ayuda a la toma de decisiones. El modelo finalmente permite ayudar al programador, diseñador y al arquitecto inexperto en la toma de decisiones para la aplicación de un patrón de diseño en una situación específica.

Palabras Claves: Patrones de desarrollo de software, técnicas de Inteligencia Artificial, arquitectura de software.

Índice

Introducción 1

Actualidad del tema 1
Formulación del Problema: 2
Hipótesis 3
Objetivo general 3
Objetivos específicos: 3
Objeto de estudio: 3
Campo de acción: 3
Variables 3
Tareas de investigación: 3
Metodología utilizada: 4
Novedad científica 4
Valor práctico 5
Estructura del trabajo de diploma 5

Capítulo 1: Estado del arte sobre patrones de diseño 6

¿Qué son los Patrones? 6
Tipos de patrones 6
¿Qué son los Patrones de Diseño? 7
Catálogo de Patrones: Libro GOF. 10
Catálogo de Patrones de Diseño J2EE. 12
Enterprise Solution Patterns, Using Microsoft .NET 15
Introduction to Design Patterns in C# 16
Herramientas para el trabajo con patrones de diseño 17

Herramienta PLinker 17
Herramienta Rational Software Architecture 19
Arquitectura para una Herramienta de Patrones de Diseño 20
Code Navigator for C++ (Quintesoft) 23
SanFrancisco 25
CoGen 26
Conclusiones del capítulo 29
Capítulo 2: Estrategias y algoritmos 30
Estrategias y algoritmos basados en inteligencia artificial 30
Algoritmos de Métodos de solución de problemas. 30
Primero en profundidad. 33
Primero en anchura. 33
Algoritmo Best-First (Primero el mejor) 34
Búsqueda de costo uniforme 35
Algoritmo A* 36
Algoritmos de clasificación sin aprendizaje 38
Algoritmo c-Means 39
Algoritmo CLASS 42
Algoritmo Holotipo 45
Conclusiones del capítulo 47
Capítulo 3:Modelo para la selección de patrones de Arquitectura. 48
Características que distinguen la aplicación de determinado patrón. 48
Modelo matemático para la identificación de patrones. 58
Análisis de los algoritmos 74
Conclusiones del capítulo 79

Conclusiones 80

Recomendaciones 81

Referencias bibliográficas: 82

Anexos 85

Anexo 1: Encuesta de arquitectura 85
Anexo 2: Matriz patrón vs. Características. 89
Anexo 3: Matriz Patrón vs. Patrón. 92

Lista de Figuras

Figura 1. Patrones del catálogo J2EE. [3] 14
Figura 2. Herramienta Quintesoft[2] 24
Figura 3. Página informativa de la herramienta.[13] 27
Figura 4. Página de generación de código. [13] 28
Figura 5. Estructura del modelo para la toma de decisiones en situaciones donde se apliquen los patrones de diseño. 59
Figura 6. Diagrama del algoritmo presentado. 67
Figura 7. Función de pertenencia asociada a las variables lingüísticas. 69
Figura 8. Representación geométrica del cluster. 73

Introducción

Actualidad del tema

Los patrones de diseño se han convertido en una técnica popular para el rehúso de conocimiento de diseño de software. La motivación principal de los patrones de diseño la constituye el hecho de encontrar frecuentemente, en diferentes diseños, problemas similares.

Los patrones de diseño es un concepto que ha venido desarrollándose conjuntamente con el desarrollo y la madurez de la Ingeniería de Software, estos patrones son una solución a un problema de diseño no trivial que es efectiva y reusable. Son soluciones de sentido común que deberían formar parte del conocimiento de un diseñador experto. Además facilitan la comunicación entre diseñadores, pues establecen un marco de referencia. Por otro lado, los patrones de diseño, facilitan el aprendizaje al programador inexperto, pudiendo establecer parejas problema-solución. El propósito de un patrón de diseño es capturar el conocimiento de un diseño de software y posibilitar su rehúso.

Los patrones han sido agrupados y organizados en catálogos, cada uno dando diferentes clasificaciones y descripciones. Algunos catálogos describen patrones de análisis mientras que otros patrones de diseño, incluyendo también patrones de un dominio particular, o bien, independientes del dominio. El proceso de construcción de aplicaciones utilizando patrones se reduce a que, cada vez que el diseñador encuentra un problema, busca en los catálogos un patrón que lo resuelva dependiendo en gran medida de la experiencia del diseñador de software. Si no existiera tal patrón, el diseñador debe pensar en una solución. En cambio, si existe un patrón, el diseñador puede aplicarlo en su diseño. En general, esto significa que los elementos de diseño de la descripción correspondiente al patrón deben ser mapeados e integrados con los elementos de diseño preexistente. Es decir, se deben identificar las clases, métodos y atributos de la aplicación que juegan el rol de aquellos prescritos por el patrón. Lamentablemente, los aspectos que implican integrar un patrón en un diseño parcial existente no son resueltos por la descripción de los patrones. Cada patrón provee información acerca de su propia implementación de una manera informal, pero no resuelve uno de los aspectos generales de la arquitectura de software con patrones, como es la integración de los mismos en diseños preexistentes. Es decir, no existe una guía o descripción que asista al diseñador a la hora de integrar un patrón a un diseño existente [1].

Actualmente los arquitectos, desarrolladores y diseñadores de software se encuentran con miles de problemas en situaciones específicas que no saben resolver o que simplemente la solución que dan no es la más idónea, esto es debido a su falta de experiencia en el campo que están desarrollando. El desarrollo de software tiene un campo de acción demasiado grande por lo que los mismos no necesariamente tienen que conocer las posibles soluciones a determinado problema presentado. De ahí que se necesite que los

experimentados en el tema dejen su solución a determinados problemas, documentada y es aquí cuando surge el concepto de patrones de diseño que como se había dicho antes son soluciones a problemas específicos, ya probadas en un determinado momento.

Formulación del Problema:

Existen diferentes patrones de desarrollo de software que ayudan a crear y realizar análisis y diseños de aplicaciones cada vez más fuertes y flexibles. Diseños que garantizan que el producto obtenido cumpla con los diferentes requisitos de funcionamiento: flexibilidad, extensibilidad, reusabilidad, y rendimiento. Estos patrones brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares, es decir son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. La aplicación de estos patrones requiere un gran conocimiento y estudio de los mismos.

En la actualidad existen numerosos documentos que abordan el tema del uso de los patrones de diseño pero aun así quedan muchas dudas e imprecisiones sobre cómo aplicarlos pues los desarrolladores de software carecen de la experiencia necesaria para aplicar los mismos. Debido a esto principalmente, los sistemas que se desarrollan hoy en día en su mayoría son poco flexibles y su extensibilidad y reusabilidad no es la deseada, además que no soportan cambios una vez construidos.

Para mejorar la aplicación de los patrones de diseño se necesita de un modelo que permita recoger y almacenar la experiencia ya desarrollada y documentada por expertos; y que ayude a la toma de decisiones en determinadas situaciones.

Como ya se había dicho anteriormente las experiencias de los desarrolladores en la solución de determinados problemas de diseño no están lo suficientemente bien detallados en la documentación y además la documentación por sí sola no es capaz de darle al desarrollador la idea de que patrón usar en determinado momento, el desarrollador necesita del conocimiento de expertos y de las decisiones de dichos expertos en las distintas situaciones.

Una vez mostrada la situación en que nos encontramos estamos en condiciones de plantearnos nuestro problema científico:

- No disponer de un modelo que facilite la ayuda a la toma de decisiones en la selección de patrones de desarrollo de software está afectando la obtención de mejores diseños y la calidad del producto final en el desarrollo de software.

Hipótesis

- Si existiera un modelo para ayuda a la toma de decisiones en la selección de patrones de arquitectura entonces se podría mejorar el proceso de diseño de los productos de software, su flexibilidad y adaptabilidad al proceso de gestión de cambios.

Objetivo general

- Desarrollar un modelo para la ayuda a la toma de decisiones basado en técnicas de inteligencia artificial y estadísticas que permita la detección de situaciones donde sea recomendable la aplicación de patrones de desarrollo de software.

Objetivos específicos:

- Desarrollar un estudio del estado del arte acerca de diferentes estrategias para la identificación y caracterización de situaciones donde se requiera la aplicación de estos patrones de diseño.
- Elaborar un modelo que permita la identificación y caracterización de situaciones donde se requiera la aplicación de los patrones de diseño.

Objeto de estudio:

Proceso de Análisis y Diseño de software

Campo de acción:

Desarrollo de patrones de arquitectura

Variables:

Variables independientes: Identificación de características relevantes de los patrones de diseño, algoritmos para identificación de patrones relevantes dada una situación específica.

Variables dependientes: El desarrollo de mejores diseños de software a partir de la identificación de situaciones donde se apliquen patrones.

Tareas de investigación:

- Realizar un análisis del estado del arte sobre las diferentes estrategias de identificación de situaciones donde se apliquen patrones de diseño.
- Estudiar los posibles algoritmos basados en la inteligencia artificial que puedan ser utilizados en el proceso de ayuda a la toma de decisiones.

- Identificar un algoritmo que permita la búsqueda heurística de la solución a un problema.
- Realizar una caracterización de cada patrón de diseño que permita identificar rasgos propios de los mismos.
- Realizar una búsqueda de ejemplos donde se apliquen diferentes patrones en conjunto.
- Determinar la frecuencia de aparición de los patrones de diseño, tomados los patrones dos a dos, basados en la búsqueda de ejemplos concretos.
- Desarrollar un modelo basado en el algoritmo identificado que permita dada las características y la frecuencia de aparición y correlación entre patrones tomar decisiones sobre la aplicación de determinados patrones.

Metodología utilizada:

Entre las estrategias de investigación que se utilizó están la exploratoria y la explicativa.

Se exploró diferentes técnicas y tendencias en el estudio de los patrones de diseño con vista a desarrollar un modelo que permita la toma de decisiones en los diferentes problemas de desarrollo de software.

Métodos teóricos: Histórico lógico, Hipotético deductivo, Sistémico.

Se enfoca la problemática de la aplicación de los patrones de desarrollo de software desde un enfoque histórico lógico, en la primera parte de la investigación desarrollada se realiza un estudio del estado del arte de las diferentes tendencias del uso de patrones de diseño y de diferentes técnicas de Inteligencia artificial y estadísticas para el desarrollo de sistema de ayuda a la toma de decisiones. La investigación sigue además un método hipotético deductivo porque a partir del problema concreto se plantean objetivos específicos e hipótesis que en el transcurso de la investigación son resueltas. Se utiliza además el método sistémico pues a partir del modelo que se crea se puede construir un sistema que permita la toma de decisiones en diferentes problemas de desarrollo de software.

Novedad científica:

La novedad científica del presente trabajo se resume en el punto siguiente:

- Se elabora un modelo que permite la identificación, caracterización y la toma de decisiones en situaciones donde se requiera la aplicación de patrones de desarrollo de software.

Valor práctico:

El modelo para la identificación y caracterización de situaciones propuesto en la tesis tiene un alto valor práctico que se manifiesta en su aplicación potencial en la construcción de un sistema que permita la ayuda a la toma de decisiones en la solución de problemas en el desarrollo de software.

Estructura del trabajo de diploma:

La tesis está conformada por tres capítulos. El primer capítulo analiza la importancia de los patrones de diseño en el desarrollo de software y se hace un estudio sobre los principales catálogos de patrones existentes.

El segundo capítulo se realiza un análisis sobre las diferentes estrategias y algoritmos basados en inteligencia artificial para ser utilizados en sistemas de ayuda en la toma de decisiones.

En el capítulo 3 se presenta un modelo para la ayuda en la toma de decisiones ante situaciones donde sea necesario aplicar determinados patrones de diseño. Se hace una caracterización de los patrones que permite buscar las relaciones de unos con otros lo cual facilitará la toma de decisiones para la aplicación de los mismos. Además presentan dos posibles algoritmos a seguir para generar los patrones a utilizar en situaciones específicas. Se finaliza el capítulo con análisis de resultados obtenidos al aplicar un algoritmo implementado.

1 Estado del arte sobre patrones de diseño

Los patrones de diseño son de gran importancia para el desarrollo de software, debido a la necesidad de experticidad para su uso, y debido fundamentalmente al insuficiente grado de madurez de la ingeniería de software no siendo así para otras ingenierías que como la ingeniería electrónica o civil, que hace muchos años que tienen sus patrones de diseños bien definidos.

En este capítulo se abordan temas como: la importancia de los patrones de diseño en el desarrollo de software, las diferentes herramientas y sistemas de ayuda a la toma de decisiones en situaciones donde se aplican los patrones.

Este capítulo es parte del fundamento teórico de lo que se abordará en el capítulo 3.

¿Qué son los Patrones?

Todos los escritores que abordan el tema de los patrones hacen referencia a la obra del arquitecto Christopher Alexander, quien propuso por primera vez un lenguaje de patrones para poder diseñar ciudades, barrios, grupos de edificios, plazas, edificios, habitaciones, etc.

Según Christopher Alexander:

“Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, además describe el núcleo de la solución de ese problema de forma que se puede utilizar esta solución un millón de veces, sin hacerlo nunca de la misma manera”.[2]

Aunque Christopher Alexander trataba el tema de patrones para la construcción de edificios y ciudades, lo que planteaba también es aplicable a los patrones de software. Sin embargo, un patrón no se limita a dar una solución, sino que proporciona una definición del problema, restricciones y una discusión de las consecuencias de aplicar esta solución. Un patrón no se convierte en tal, hasta que no se haya probado una y otra vez, y se haya verificado que resuelve un problema que se presenta muchas veces.

Tipos de patrones

En general, hay patrones para todos los aspectos de la Ingeniería del Software, entre otros: organización del desarrollo, planificación de proyectos, ingeniería de requerimientos, y gestión de la configuración del software.

Debido a la gran aceptación del libro del GOF, el cual se analizará en secciones posteriores, los patrones de diseño se han situado como los más importantes. Sin embargo, hay otros tipos de patrones ampliamente difundidos. Algunos de ellos, son los patrones de:

Implementación: (también llamados idiomas): Son de un nivel más bajo que los de diseño ya que tienen una granularidad distinta y tratan sobre el código.

Análisis: Análogos a los de diseño, solo que referentes a la fase de análisis.

Organización: Tratan la organización del desarrollo.

Interfaz: Ayudan en la creación del interfaz.

De todos los citados, los que mayor auge están tomando son los de implementación y los de organización, aunque los de diseño siguen siendo los más populares.[2]

¿Qué son los Patrones de Diseño?

Analistas y programadores desarrollan a diario habilidades para resolver problemas usuales que se presentan en el desarrollo del software. Por cada problema que se presenta se piensa en distintas formas de resolverlo, incluyendo soluciones exitosas que ya se han usado anteriormente en problemas similares. Es así que a mayor experiencia que se tenga, el abanico de posibilidades para resolver un problema crece, pero al final siempre deberá escogerse la solución que mejor se adapte a la aplicación o la de menor riesgo. Si se documenta esta solución, se puede reutilizar y compartir esa información para resolver de la mejor manera un problema específico.

Los patrones del diseño tratan los problemas del diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes. Existen muchas formas de implementar patrones de diseño.

Un patrón de diseño es una abstracción de una solución en un nivel alto. Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo; desde el análisis hasta el diseño y desde la arquitectura hasta la implementación. [3]

En general, un patrón de diseño tiene cuatro elementos esenciales:

La Identidad: que es usada para describir un problema de diseño, su solución y consecuencias en una o dos palabras. La identidad o nombre de un patrón incrementa el vocabulario de diseño y permite así, poder comunicar un diseño a un nivel más abstracto que las notaciones básicas.

El Comportamiento o Problema, como también se le puede llamar, describe cuando aplicar el patrón. Es decir, provee una descripción del problema y su contexto. Generalmente esta descripción es acompañada por una lista de condiciones que deben cumplirse antes de poder aplicar el patrón en algún diseño.

La Solución describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboraciones. Esta solución no describe un diseño en concreto, ni una implementación del mismo, ya que un patrón puede ser visto como una plantilla que puede ser aplicado a diferentes situaciones. El patrón provee una descripción abstracta de un problema de diseño y como un conjunto general de elementos (clases y objetos) lo resuelven.

Las Consecuencias son los resultados de aplicar el patrón. Estas consecuencias pueden verse como críticas a la hora de evaluar alternativas de diseño y entender los costos y beneficios de aplicar el patrón.

Existen varias razones por las cuales es beneficioso utilizar patrones de diseño en el desarrollo de una aplicación orientada a objetos. Las principales ventajas de los patrones incluyen las siguientes:

- Un patrón trata un problema de diseño recurrente en diversas situaciones específicas de diseño y presenta una solución a este problema.
- Los patrones documentan la ya existente y bien demostrada experiencia de diseño. Es decir, los patrones no son creados artificialmente, si no que surgen de la experiencia y conocimiento de los diseñadores expertos. Entonces, cualquier patrón puede ser aplicado a diferentes problemas de diseño sin tener que ser descubierto nuevamente por el diseñador.
- Los patrones identifican y especifican abstracciones que se encuentran por sobre el nivel de las clases e instancias o de los componentes. Un patrón de diseño, como ya se mencionó, describe diferentes componentes, clases u objetos y especifica sus responsabilidades y relaciones como también sus colaboraciones. Todos estos componentes juntos resuelven el problema, que el patrón trata, de una manera más eficiente que un simple componente.
- Los patrones de diseño brindan un vocabulario común de diseño entre los desarrolladores, ya que proveen un nombre y una descripción para cada una de las estructuras, permitiendo así poder comunicar un diseño en función de un lenguaje de más alto nivel que las notaciones básicas.
- Los patrones soportan la construcción de software con propiedades definidas. Los patrones proveen un comportamiento funcional definido con lo que ayudan así a implementar la funcionalidad específica de un sistema. Este esqueleto comportamental provisto por el patrón trata

explícitamente algunos requerimientos no funcionales como la fiabilidad, reusabilidad, adaptabilidad, etc.

- Los patrones ayudan a construir complejas y heterogéneas arquitecturas de software. Los patrones proporcionan un conjunto predefinido de componentes, roles y colaboraciones entre ellos. Entonces, cada patrón puede ser usado para especificar aspectos particulares de una estructura de software particular. Es decir que, los patrones actúan como bloques de construcción menores para desarrollar diseños más complejos.

- Los patrones ayudan a manejar la complejidad del software. Cada patrón describe una manera para mejorar el problema que este resuelve. Entonces cuando se encuentra un patrón que soluciona un problema de diseño en particular, no vale la pena invertir tiempo inventando una nueva solución. Si no que aplicando correctamente el patrón se consigue obtener la solución al problema. [1]

Muchos diseñadores y arquitectos de software han definido el término de patrón de diseño de varias formas que corresponden al ámbito a la cual se aplican los patrones. Luego, se dividió los patrones en diferentes categorías de acuerdo a su uso.

Los diseñadores de software extendieron la idea de patrones de diseño al proceso de desarrollo de software. Debido a las características que proporcionaron los lenguajes orientados a objetos (como herencia, abstracción y encapsulamiento) les permitieron relacionar entidades de los lenguajes de programación a entidades del mundo real fácilmente, los diseñadores empezaron a aplicar esas características para crear soluciones comunes y reutilizables para problemas frecuentes que exhibían patrones similares.[3]

Debido a la necesidad y la importancia de la aplicación de los patrones de desarrollo de software, se han desarrollados diferentes catálogos que permiten reunirlos de forma que resulten accesibles mediante distintos criterios, pues lo que se necesita no es tan sólo la completa descripción de cada uno de los patrones sino, esencialmente, la correspondencia entre un problema real y un patrón (o conjunto de patrones) determinado. [4]

Lo que se pretende con los catálogos de patrones no es favorecer al diseñador experto que quizás no necesite en absoluto de los patrones, sino más bien ayudar al diseñador inexperto a adquirir con cierta rapidez las habilidades del diseñador experto, como también comunicar al posible cliente, si es el caso, las decisiones de diseño de forma clara y autosuficiente.

Existen varios catálogos de patrones pero la capacidad de elección todavía se basa en el conocimiento completo de los mismos.

El catálogo de patrones más famoso es el contenido en el libro “Design Patterns: Elements of Reusable Object-Oriented Software”, también conocido como el LIBRO GOF (Gang-Of-Four Book) [5]. Es considerado uno de los libros más vendidos del mundo de la programación orientación a objetos y recomendado para todos aquellos que se quieren dedicar en serio a la misma. Con la publicación de este libro, los patrones de software adquirieron una gran relevancia.

Otros catálogos son: Catálogo de Patrones de Diseño J2EE. [3]

En los epígrafes siguientes se analizarán cada uno de los anteriores catálogos.

Catálogo de Patrones: Libro GOF.

Uno de los principales catálogos de patrones es el propuesto por Erich Gamma [5], el cuál divide el conjunto de patrones en función de dos criterios ortogonales: alcance y propósito. Cada patrón es descrito por una ficha, en la cual se detallan las características relevantes para su utilización.

El primer criterio de división, el alcance, especifica si el patrón es aplicable primariamente a clases o a objetos. Los patrones de clase, tratan con relaciones entre clases y subclases. Estas relaciones son establecidas a través de herencia, de manera que se trata de relaciones estáticas. Los patrones de objeto, en cambio, tratan con relaciones entre objetos. Estas relaciones son más dinámicas, ya que pueden ser cambiadas en tiempo de ejecución.

El otro criterio propuesto por Gamma para organizar el catálogo de patrones es el propósito. Este criterio describe la función que el patrón cumple. Los patrones de diseño pueden tener propósito creacional, estructural o de comportamiento:

Creacionales: Patrones creacionales involucran el proceso de creación de los objetos. Esta clase de patrones abstrae el proceso de instanciación, permitiendo crear sistemas de software independientes de cómo sus objetos componentes son creados, compuestos o representados. Los patrones creacionales de clase utilizan la herencia para variar la clase instanciada, mientras que los patrones creacionales de objetos delegarán la instanciación a otro objeto.

Estructurales: Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos. Los patrones estructurales de clase utilizan el

mecanismo de herencia para componer interfaces. Los patrones estructurales de objetos, en cambio describen la manera de componer objetos para brindar nueva funcionalidad.

Comportamiento: Los patrones de comportamiento nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos. Caracterizan la manera en que clases u objetos interactúan y distribuyen responsabilidades. Los patrones de esta categoría, no sólo describen patrones de clases y objetos, si no también patrones de comunicación entre ellos. Los patrones comportamentales de clase usan herencia para distribuir comportamiento, mientras que los de objetos lo hacen por medio de la composición.

En la tabla 1 se muestra el conjunto de patrones propuestos por este catálogo organizados por los dos criterios anteriormente mencionados.

Tabla 1. Conjunto de patrones

		Propósito		
		Creacional	Estructural	Comportamental
Alcance	Clase	Factory Method	Adapter (class)	Interpreter Template Method
	Objetos	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

En este catálogo se discute de forma general los patrones creacionales, estructurales y de comportamiento, para llegar a un entendimiento mucho mejor del estudio de cada uno de ellos; en esta discusión se enfatiza en la relación entre ellos en su importancia desde el punto de vista de su clasificación. El catálogo hace un análisis exhaustivo de cada patrón, de cada uno analiza su intención, motivación, aplicabilidad, estructura, consecuencias, implementación, y la relación con otros patrones.

Catálogo de Patrones de Diseño J2EE.

Con la aparición del J2EE, todo un nuevo catálogo de patrones de diseño apareció. Desde que J2EE es una arquitectura por si misma que involucra otras arquitecturas, incluyendo servlets, JavaServer Pages, Enterprise JavaBeans, y más, merece su propio conjunto de patrones específicos para diferentes aplicaciones empresariales.

Este catálogo explica 15 patrones J2EE que están divididos en 3 de las capas: presentación, negocios e integración. Los patrones J2EE se han estructurado de acuerdo a una plantilla de patrón definida. Cada sección de la plantilla del patrón contribuye a entender el patrón particular. También se le intenta dar un nombre de patrón descriptivo a cada uno. Los nombres de los patrones proporcionan suficiente información sobre la función del mismo.

Se ha adoptado una plantilla de patrón que consta de las siguientes secciones:

Contexto:
Conjunto de entornos bajo los cuales existe el patrón.

Problema:
Describe los problemas de diseño que se ha encontrado el desarrollador.

Causas:
Lista los razones y motivos que afectan al problema y la solución. La lista de causas ilumina las razones por las que uno podría haber elegido utilizar el patrón y proporciona una justificación de su uso.

Solución:
Describe brevemente la solución y sus elementos en más detalle. La sección de la solución contiene dos sub-secciones:

Estructura:
Utiliza diagramas de clases UML para mostrar la estructura básica de la solución. Los diagramas de

secuencias UML de esta sección presentan los mecanismos dinámicos de la solución. Hay una explicación detalladas de los participantes y las colaboraciones.

Estrategias:
Describe diferentes formas en las que se podría implementar el patrón.

Consecuencias:
Aquí se describen las compensaciones del patrón, esta sección se enfoca en los resultados de la utilización de un patrón en particular o de su estrategia, y anota los pros y los contras que podrían resultar de la aplicación del patrón.

Patrones Relacionados:

Esta sección lista otros patrones relevantes en el catálogo de patrones J2EE u otros recursos externos, como los patrones de diseño GOF. Por cada patrón relacionado, hay una breve descripción de su relación con el patrón que se está describiendo.

A continuación se muestra en la figura 1, el conjunto de estos patrones.

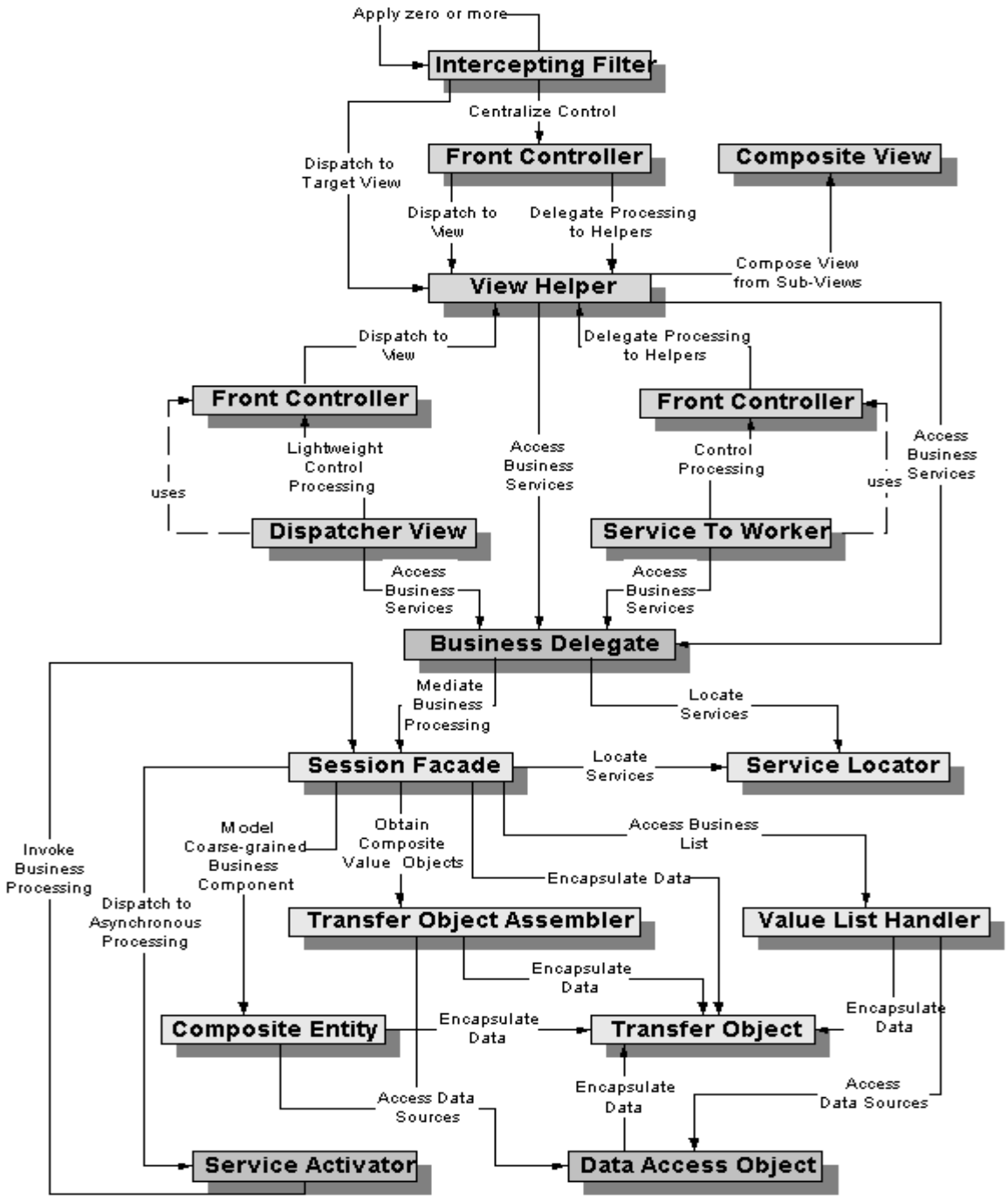

Figura 1. Patrones del catálogo J2EE. [3]

Los patrones que se encuentran en la capa presentación son: Service to Worker, Composite View, Dispatcher View, Front Controller, Intercepting Filter, View Helper. Para la capa de de Negocio y de Integración: Busisness Delegate, Service Locator, Session Facade, Transfer Object, Transfer Object Assembler, Value List Handler, Composite Entity, Data Access Object, Service Activator.

El principal inconveniente de este catálogo con respecto a los demás, como por ejemplo el de GOF es que describe los patrones que son específicos solo para java dejando fuera de se alcance un gran grupo de patrones muy útiles para el diseño de software independientemente del ambiente en que se desarrollen.

Enterprise Solution Patterns, Using Microsoft .NET

Enterprise Solution Patterns. Using Microsoft. NET, es una guía concreta para introducir patrones y describir una nueva forma de acercarse a categorías desde varios puntos de vistas y relaciones. Este libro presenta una serie de patrones que abarca estos puntos de vistas y explica cómo pueden ser integrados en soluciones de empresas.[6]

El libro introduce los conceptos y primeras nociones de los patrones y muestra como una colección de patrones, provee un lenguaje común para desarrolladores y arquitectos. Para ilustrar estos conceptos aplica versiones de patrones actuales en situaciones desarrolladas en la vida real. Además explica como los patrones aparecen en diferentes niveles de abstracción y a través de variedad de dominio. Explora niveles de patrones en detalle y organiza bocetos que ayudan a buscar patrones relevantes rápidamente.

El libro presentan un catálogo de 27 patrones los cuales son agrupados hasta clusters, estos clusters son clasificados en: Web presentation patterns, Deployment Patterns, Distributed Systems Patterns, y Services Patterns

Web Presentation Patterns: describen el diseño e implementación de los patrones relacionados para la construcción de aplicaciones web dinámicas y aquí se tiene a los patrones siguientes: Model-View-Controller, Page Controller, Front Controller, Intercepting Filter, Page Cache, Observer

Deployment Patterns: ayudan a reducir la tensión entre el desarrollo de aplicaciones la infraestructura de equipos del sistema, ofreciendo una guía de cómo estructurar la aplicación y la infraestructura técnica para cumplir eficientemente los requerimientos de la solución. Estos patrones son organizados en las capas lógicas de la aplicación, refinando las capas. Layered Application, Three-Layered Services Application, Tiered Distribution, Three-Tiered Distribution, Deployment Plan.

Distributed Systems Patterns: Introducen conceptos relevantes para sistemas distribuidos. Broker, Data transfer object, singleton.

Services Patterns: contrario a los sistemas distribuidos, estos patrones tienen que ver con sistemas conectados con muchos enlaces usando servicio basado en colaboración. Service Interface, Service Gateway.

Enterprise Solution Patterns ayuda a desarrollar mejores aplicaciones de empresas con el trabajo de patrones. Patrones que han sido clasificados y analizados para este tipo de sistemas. Este libro ofrece una buena ayuda para diseñadores y arquitectos de software que deben construir modelos de diseñas cada vez más flexibles y reusables.

Este catálogo comparado con el de GOF [5], coinciden en algunos patrones por ejemplo el singleton, pero trata muchos otros, que GOF no toca pues son patrones específicos para el desarrollo de aplicaciones de empresas, GOF trata patrones de diseño para cualquier software a desarrollar, mientras que este libro abarca otros tipos de patrones pero solo los necesarios en este campo de acción.

Introduction to Design Patterns in C#

Introduction to Desing Pattrerns in C# es un libro práctico que aborda el tema: sobre cómo escribir programas en el lenguaje c# usando los patrones de diseño más comunes [7]. Los patrones descritos son estructurados en capítulos cortos, cada uno describe un patrón de diseño y dando un programa de ejemplo visual que usa el patrón. Cada capítulo también incluye los diagramas de UML, ilustrando cómo las clases actúan recíprocamente.

Este libro no es un compendio como lo es el libro de los cuatro [5], en cambio es un tutorial para personas que quieran aprender sobre patrones de diseño y como usarlos en su trabajo. Este libro está organizado en seis secciones principales, una descripción introductoria, una descripción de c# y la descripción de los patrones agrupados como creacionales, estructurales y de comportamiento, es decir trata los 23 patrones abordados en el catálogo de los cuatro [5]. Para cada patrón se ofrece una descripción breve y un ejemplo programado, Cada uno de estos ejemplos es un programa visual que se puede correr y examinar.

Una vez terminado de estudiar este libro el lector tendrá los conocimientos básicos sobre estos patrones y podrá empezar a usarlos en su programación en el lenguaje c#.

Este libro tiene aspectos positivos desde el punto de vista de que se aprende patrones vinculado a un lenguaje específico, pero esto tiene sus inconvenientes pues el libro no se dedica solo a los patrones dejando el estudio de los mismos poco profundo. En él no se hace un análisis exhaustivo como se hace en el libro de GOF, provocando que el lector no adquiera el suficiente conocimiento sobre dichos patrones.

Herramientas para el trabajo con patrones de diseño

Hasta ahora se ha abordado sobre los diferentes libros o catálogos que tratan los patrones de diseño, pero no son solo catálogos los que abordan este tema. Con la creación de esta serie de libros sobre los patrones y el desarrollo de la ingeniería de software muchos se han interesado por crear herramientas que ayuden al desarrollador a aplicar estos patrones en sus diseños de sistemas. A continuación se describirán algunas de estas herramientas.

Herramienta PLinker

Se ha desarrollado una herramienta denominada pLinker la cual pretende que el usuario aproveche las ventajas del uso de patrones de diseño y, principalmente de las relaciones entre los mismos sin tener un conocimiento avanzado sobre el tema.

Esta herramienta fue la tesis de grado de dos estudiantes de la Facultad de Ciencias Exactas, Universidad Nacional del Centro de la Provincia de Buenos Aires: Leonardo Liebener, Marcos Andres Rossi dirigida por la Dra. Claudia Marcos. La herramienta implementada más que nada demuestra conceptos (en particular, como aprovechar las relaciones entre patrones de diseño) y está lejos de ser práctica, aunque es funcional.

PLinker consiste de un editor gráfico de clases UML que facilita la utilización y administración de patrones de diseño, permitiendo mantener y actualizar un catálogo de patrones de diseño y las relaciones entre los mismos.

Por otro lado, provee mecanismos que simplifican la incorporación de patrones a diseños preexistentes, aprovechando además, las relaciones entre dichos patrones.

PLinker posee dos modos de trabajo:

- El primer modo está orientado a la construcción de diseños con patrones, permite modelar visualmente clases, interfaces y relaciones entre las mismas (herencia, uso, e implementación). Pero su principal característica es la posibilidad de incorporación de forma sencilla de patrones de diseño a un modelo de clases existente. Además, permite distinguir cuales fueron los patrones utilizados y ofrece opciones que ayudan a mejorar el diseño en desarrollo incorporando nuevos patrones en forma automática, valiéndose para ello de las relaciones existentes entre los patrones involucrados.

- El segundo modo está destinado a mantener y actualizar el catálogo de patrones de diseño y sus relaciones. La incorporación de un nuevo patrón puede hacerse tomando como referencia la plantilla de documentación del mismo, disponible en cualquiera de los catálogos de patrones. Además, de incorporar la descripción del nuevo patrón se deben describir las relaciones del mismo con los patrones ya existentes en la herramienta y las acciones a desarrollar en cada uno de los casos.

Otra característica que posee pLinker es que permite crear diseños partiendo de arquitecturas de software, organizando de esta manera las clases del modelo del usuario en componentes arquitectónicos. Esto brinda una ayuda adicional a la hora de utilizar un patrón de diseño, ya que por cada componente arquitectónico, pLinker propone los patrones de diseño que se usan más comúnmente en él.

Por cada patrón de diseño, pLinker mantiene tres tipos de información: la descripción general, la estructura de clases que conforma el patrón, y las actividades a realizar para complementar su uso.

En la descripción general se deben especificar el nombre del patrón, la intención, la motivación y otros datos relevantes que ayuden al usuario a optar o no por un determinado patrón de diseño. La estructura de clases se refiere a las clases y relaciones entre ellas involucradas para resolver un problema de diseño particular. Finalmente, las actividades representan un conjunto de tareas que debe realizar el usuario con el fin de completar la inserción de un patrón en su diseño. Por ejemplo, una actividad que surge generalmente es la de renombrar una clase del patrón para que su nombre represente exactamente lo que modela. Una actividad tiene un nombre (como podría ser: “Renombrar clase C”), una descripción que muestra los detalles y una prioridad que indica la importancia de la misma.

Para incorporar la información correspondiente a esta relación, el usuario debe indicar cuáles son los patrones involucrados, el tipo de relación y la descripción de la relación. También deben establecerse los vínculos entre las clases que permiten mantener la consistencia entre las estructuras de diseño de los patrones involucrados en la relación. Al establecer estos vínculos, pLinker creará automáticamente las acciones necesarias que permitan incorporar el segundo patrón de la relación en la estructura de clases del primero.

Adicionalmente, pLinker permite almacenar información acerca de arquitecturas de software y sus componentes. Un componente arquitectónico puede estar diseñado con un conjunto de patrones de diseño. Por cada uno de estos componentes, pLinker permite establecer cuáles son los patrones de diseño más frecuentemente utilizados en su diseño. [1]

Básicamente pLinker tiene como objetivo asistir al usuario en la construcción de diseños utilizando patrones y sus relaciones de manera simple, este garantiza, una vez representado el diseño y la incorporación del patrón a usar, la relación del mismo con otros modificando el modelo de diseño.

Herramienta Rational Software Architecture

La herramienta Rational Software Architecture (RSA) proporciona un soporte de desarrollo y diseño integrado para el desarrollo dirigido por el modelo con UML. Es una herramienta para la creación de servicios y aplicaciones con arquitecturas sólidas. Se puede unificar todos los aspectos del desarrollo y el diseño de software[8]. Permite desarrollar aplicaciones de forma más productiva que nunca, a continuación se ofrecen algunas de sus características:

- Utiliza lo último de la tecnología de lenguajes de modelado.
- Revisa y controla la estructura de sus aplicaciones Java.
- Refuerza la plataforma de modelado ampliable y abierto.
- Simplifica solución de herramienta de desarrollo y diseño.
- Integra la solución con otros aspectos del ciclo de vida.

IBM Rational Software Architect está disponible para su adquisición con licencias de usuario autorizado (usuario único), Licencias de Término Fijo (FTL) de usuario autorizado (usuario único) o licencias flotantes (usuario simultáneo).

Sistemas Operativos y Plataformas de Hardware Apropiadas: Linux, Windows 2000, Windows - WS2003, Windows XP.

Otras de las funcionalidades que incluye esta herramienta es el trabajo con patrones de diseño.

RSA trae predefinido un conjunto de patrones que pueden ser aplicados por el usuario. Un patrón puede ser aplicado para cambiar la estructura de un elemento existente o usado como plantilla para crear un nuevo elemento, los patrones son tratados como transformaciones que ocurren sobre un modelo.[9]

RSA presenta un explorador de patrones, el cual contiene un árbol, un panel de información que contiene un resumen del patrón y una breve descripción, estos patrones se encuentran organizados por la clasificación del grupo de los cuatro, creacionales, estructurales y de comportamiento. El árbol presenta una lista jerárquica de patrones organizados por grupos; se permite copiar y mover un patrón, renombrar

un grupo etc. desde este panel se puede aplicar el patrón seleccionado. El resumen del panel de información te muestra un diagrama de clases del patrón así como los objetos que participan. La forma de usar estos patrones es simplemente seleccionar el patrón que desee y aplicarlo sobre el modelo de diseño.

Esta herramienta facilita el trabajo con patrones de diseño, desde el punto de vista que ofrece el diagrama de clases de cada uno de ellos así como una breve descripción, el desarrollador selecciona el patrón que desea utilizar y automáticamente se incluye el diagrama de clases del patrón dentro del diagrama construido por el mismo, esto pude ser una gran ventaja pues le ahorra trabajo al desarrollador pero todavía no logra lo que se desea, que es tener una herramienta que ayude a la toma de decisiones en situación específicas donde se apliquen determinados patrones de diseño.

Arquitectura para una Herramienta de Patrones de Diseño

Proyecto realizado en el año 1999, por José Sáez Martínez, Jesús García Molina, Pedro J. Jiménez García del Departamento de Informática, Lenguajes y Sistemas de la Facultad de Informática en la Universidad de Murcia.

La herramienta, está hecha en Java (incluye los ficheros de proyecto para JBuilder, pero es bastante fácil pasarla a cualquier otro entorno Java). [2]

Este proyecto presenta una arquitectura para representar patrones de diseño, de forma que puedan ser fácilmente integrados y tratados en una herramienta de desarrollo. La arquitectura se ha diseñado, teniendo en cuenta los patrones de diseño descritos en el libro de los cuatro [5], y ha sido evaluada mediante su implementación en una herramienta que permite el trabajo con patrones de diseño.[2]

Para realizar esta arquitectura primeramente los autores hacen un análisis de las necesidades más importantes que debe cubrir una herramienta que permita el trabajo con patrones de diseño y posteriormente a este se presenta el diseño de la arquitectura que se ha establecido y se hacen algunas consideraciones sobre las posibilidades de integración de esta arquitectura en herramientas de desarrollo.

Para comenzar el análisis se hace una distinción entre la descripción estructural genérica de un patrón (plantilla del patrón) y una instancia concreta de ese patrón que se esté utilizando en un proyecto determinado (instancia del patrón). Por ejemplo, un proyecto puede contener dos instanciaciones de la plantilla del patrón Observer: una para conectar las vistas y el modelo de datos, y la otra para comunicar los cambios que se produzcan en un objeto a otros objetos. [8]

En este análisis tratan al conjunto de clases y relaciones entre ellas, de un patrón como roles del mismo. Por ejemplo, el patrón Observer contiene cuatro roles de clase, que son Subject, Observer, ConcreteSubject y ConcreteObserver. Un rol de clase puede contener métodos y atributos, que sirven como parámetros de la plantilla del patrón. Por ejemplo, el rol de clase Observer incluye el método update. La relación observers entre los roles de clase Subject y Observer es un ejemplo de rol de relación. [8]

Una aplicación orientada a objetos consta de una estructura de clases que forma un grafo, cuyos nodos son las clases y los arcos son las relaciones entre ellas (herencia, asociación,...) [8] Los autores utilizan el término proyecto para hacer referencia a esa estructura de clases que construye el programador.

Para facilitar el trabajo con patrones de diseño, de forma que el usuario pueda aplicarlos fácilmente, definieron una serie de aspectos que la herramienta debe cubrir para lograr lo plateado. Algunos de estos aspectos son por ejemplo:

- Mantener una colección de plantillas de patrones que puedan ser instanciadas por el usuario.
- Permitir la eliminación y adición de las platillas.
- A partir de los elementos constituyentes de un patrón de diseño, se debe poder generar automáticamente la declaración de clases, interfaces, relaciones, atributos y operaciones, y permitir hacer cambios sobre los elementos generados.
- Debe tratar de forma general las relaciones entre patrones.
- Deberá mantener la integridad del significado de los patrones implicados en un sistema, es decir, se debe comprobar que no se rompe la estructura de una instancia de un patrón tras una acción (eliminar un atributo,...). Por este motivo, se deberá llevar un control sobre las clases que participan en patrones y las ajenas a ellos. [8]

En este trabajo se identifican las principales características que debe proporcionar una herramienta de desarrollo que facilite el uso de patrones de diseño (instanciación de patrones, mantenimiento de la consistencia de los patrones, gestión de vistas, generación de código), y se ha presentado una arquitectura para una herramienta de este tipo.

Las principales ventajas asociadas a esta arquitectura son las siguientes:

- Permite manipular los patrones de diseño como elementos básicos de modelado.
- Es flexible, y se pueden añadir fácilmente nuevos elementos estructurales a un patrón, tales como código (asociado a los métodos), notas, restricciones, etc.
- Puede ser fácilmente introducida en una herramienta CASE de más alto nivel, que trabaje al nivel de modelos de diseño de sistemas software.
- Puede utilizarse tanto para representar patrones de diseño como estructuras mayores.

Por ejemplo, toda la información de un modelo de diseño completo puede mantenerse con esta arquitectura. Uno de los principales inconvenientes de esta arquitectura es el mantener la semántica de la colección componentes de los elementos compuestos. Las limitaciones sobre los tipos y cardinalidades de los componentes significativos para un elemento compuesto deben ser mantenidas explícitamente mediante el código. Por ejemplo, puede haber limitaciones sobre los componentes de una Interfaz (puede contener Métodos, pero no Atributos), o de una Relación (sólo puede contener dos componentes de tipo Clase), y otras parecidas. [8]

A partir de esta arquitectura propuesta se crea una herramienta que es implementada con la misma, la cual resume todo los aspectos tratados anteriormente. Esta aplicación agrupa sus funcionalidades en siete categorías de operaciones, que son las siguientes:

- Sobre los proyectos que crea.
- De edición
- Sobre los patrones.
- Sobre los elementos de software.
- Para generación de resultados.
- De ayuda.

Abordando brevemente sobre cada una de ellas se puede decir que las operaciones sobre los proyectos deben permitir guardar y recuperar los diseños realizados. Sobre los patrones debe permitir instanciar un patrón, borrar un patrón, añadir o borrar un elemento (clase interfaces, métodos y atributos). Sobre los

elementos debe permitir crear una clase, añadir métodos y atributos. Finalmente sobre la generación de resultado debe generar el código sobre los diagramas creados y generar informes sobre los diseños realizados. [2]

Resumiendo esta herramienta permite la creación de diagramas de diseños y a los mismos se le puede adjuntar patrones de diseño escogidos por el usuario, una vez tenido el diagrama completo podrá generar el código de dicho diagrama.

Code Navigator for C++ (Quintesoft)

Es una herramienta que permite una visualización gráfica de la jerarquía de clases. Todo de una forma gráfica. Además tiene opciones de generación de informes, bloqueo de módulos, control de revisiones, etc.

Principalmente permite tratar con patrones de código, aunque la empresa que la ha desarrollado afirma que es la primera aplicación comercial que incluye patrones de diseño orientado a objetos. La herramienta permite generación automática del código necesario para la implementación de los conceptos aportados por los patrones. El aspecto general de la aplicación se puede observar en la **¡Error! No se encuentra el origen de la referencia.**:

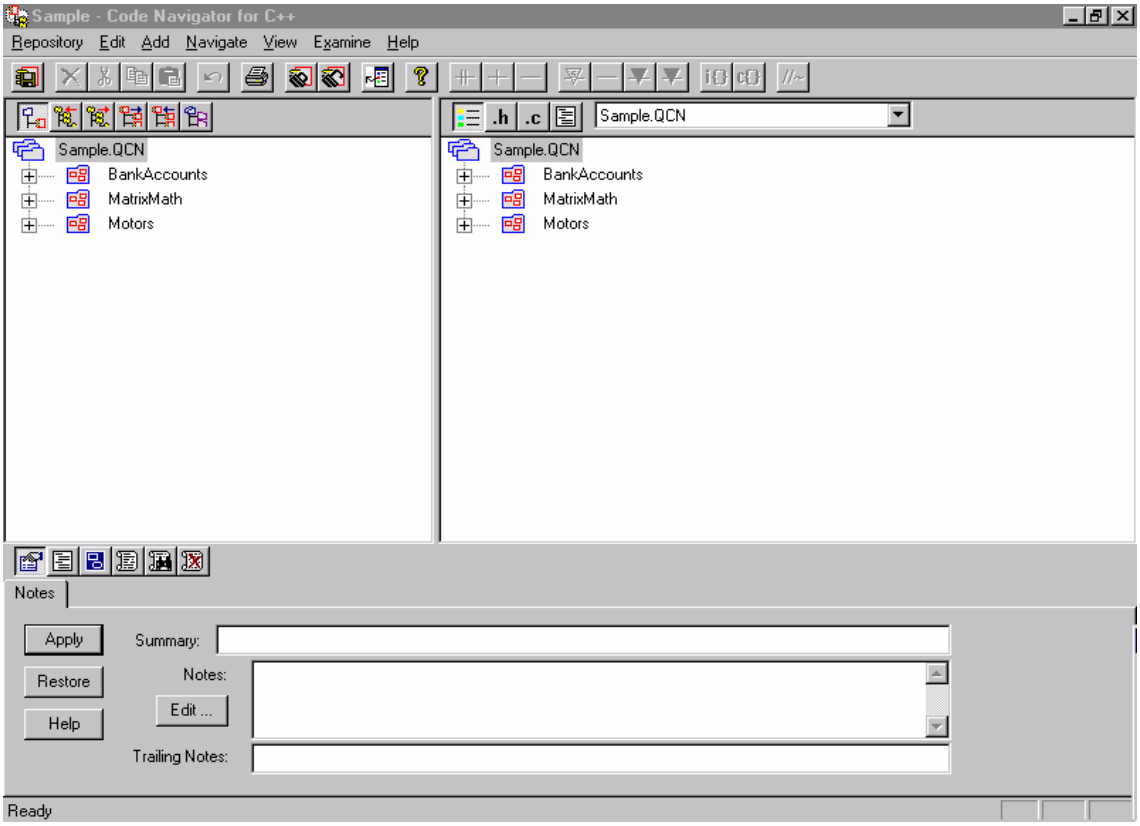

Figura 2. Herramienta Quintesoft[2]

La secuencia de pasos para generar el código de un patrón es la siguiente:

- Creación de las clases de los participantes en el patrón. A estas clases se les puede dar un nombre cualquiera.
- Asignación mediante el ratón (drag-and-drop) de los participantes del patrón con las clases ya creadas.
- Indicación al programa para que se aplique el patrón. Con esto, se genera el código automáticamente.

Problemas en la utilización de esta herramienta:

- No se lleva un registro de la aplicación de patrones. Por lo tanto, es imposible deshacer la aplicación de un patrón, o simplemente saber cuántos patrones se han aplicado.
- Un patrón puede tener varias implementaciones. El código que genera la herramienta es fijo para cada patrón y sin embargo algunas veces es necesario implementar el patrón de otra manera.[2]

SanFrancisco

IBM SanFrancisco es un producto que se desarrolló sobre los años 90, el cual pretendía convertirse en un framework para aplicaciones de gestión. Su objetivo era ayudar a los desarrolladores de aplicaciones a construir rápidamente aplicaciones distribuidas y orientadas a objetos. Facilitaba un conjunto base de infraestructura orientada a objetos y lógica de aplicación que podía ser ampliada y enlazada por cada desarrollador. Estaba orientado al desarrollo con el lenguaje Java. [10] A continuación se abordará sobre este producto.

SanFrancisco proporciona un conjunto de frameworks que permiten a los vendedores de software centrarse en sus áreas de dominio específicas dejando de lado cuestiones tecnológicas. Tiene una arquitectura con distintas capas que evita a los programadores tener que tratar con detalles de implementación de aspectos relacionados con distribución e integridad de transacciones.[2]

Los frameworks San Francisco se distribuyen en tres capas distintas de código orientado a objetos, que se sitúan encima de la JVM (Máquina Virtual Java) de cualquier plataforma soportada. La inferior, la capa de Cimientos y Utilidades (Foundation and Utilities), proporciona una interfaz de programación y una estructura sólidas para la creación de aplicaciones. La capa intermedia, Objetos Comunes de Empresa (Common Business Objects), incluye funciones y servicios comunes a diferentes tipos de aplicaciones, como por ejemplo objetos relacionados con direcciones, facturación, cambio de moneda, y agenda. La capa superior contiene los Procesos Fundamentales de Empresa (Core Business Processes) específicos a determinados tipos de aplicaciones, como por ejemplo, contabilidad general y Cuentas corrientes. Los desarrolladores pueden elegir la capa en la que quieren añadir su código: mejorando y ampliando los Procesos Fundamentales de Empresa, añadiendo el código directamente encima de los servicios de Cimientos, o una aproximación intermedia empezando por los Objetos Comunes de Empresa. [11]

SanFrancisco ofrece la implementación de numerosos patrones de diseño listos para aplicar. Entre ellos están el patrón *“Proxy”* o el patrón *“Factory”*, que ayudan a resolver cuestiones tecnológicas.

La tecnología en la que se basan sus componentes son los Java Beans, aunque se hizo un estudio para ver la posibilidad de migrar hacia componentes EJB (Enterprise Java Beans). Algunas empresas asociadas a IBM realizaron desarrollos basados en SanFrancisco. En 1998 sacaron una versión la 1.3. Este producto se enmarca dentro de unas de las estrategias realizadas por IBM, dirigida a orientar el desarrollo de aplicaciones de gestión (especialmente sobre sus servidores AS/400) hacia el lenguaje Java. [2]

Este proyecto no se consolidó, el proyecto fracasa y este fracaso básicamente fue a causa del auge de otras tecnologías. Sobre este fracaso se han desarrollados varias opiniones, muchos creen que este no se pudo desarrollar principalmente debido a la falta de conocimientos de la IBM y otros simplemente plantea que no supieron manejar dicho proyecto. [12] Lejos de lo que pueden plantear, la realidad es que este proyecto fracaso cuando comenzaba a desarrollarse.

CoGen

Los patrones de diseño aumentan el nivel de abstracción de las personas que diseñan software orientado a objeto. Sin embargo el mecanismo de implementación de los patrones de diseño se deja a los programadores. [13] A continuación se describe la arquitectura de una herramienta que automatiza la implementación de los patrones de diseño.

Es una herramienta de generación automática de código a partir de patrones de diseño. El usuario de la herramienta proporciona la información específica para un patrón dado, a partir de la cual, la herramienta genera el código del patrón automáticamente. La herramienta crea declaración de clases y definiciones que implementa el patrón, permitiéndole al usuario adicionar este código al resto de la aplicación, frecuentemente asegurando funcionalidades específicas de la aplicación.[13] La herramienta también incorpora una asistencia en línea, con el hipertexto de los patrones, esta versión en línea ofrece al usuario un acceso conveniente al material permitiéndoles seguir los enlaces entre los patrones y buscar la información rápidamente.

La herramienta despliega los aspectos de los patrones, intención, motivación, etc. en cada página, estas páginas a su vez tienen información adicional sobre otros enlaces que se relacionan con el patrón. Como se muestra en la figura 3

Figura 3. Página informativa de la herramienta.[13]

La página de generación de código viene inmediatamente después de las páginas anteriores y puede ser accedida desde otras páginas secuencialmente. Esta página le permite al usuario entrar información con la cual puede generar una implementación personalizada del patrón. Estas páginas están integradas completamente con las otras, referencia a participantes y a otros detalles mostrados en enlaces anteriores con respecto a discusión de los patrones. En la figura 4 se muestra dicha página.

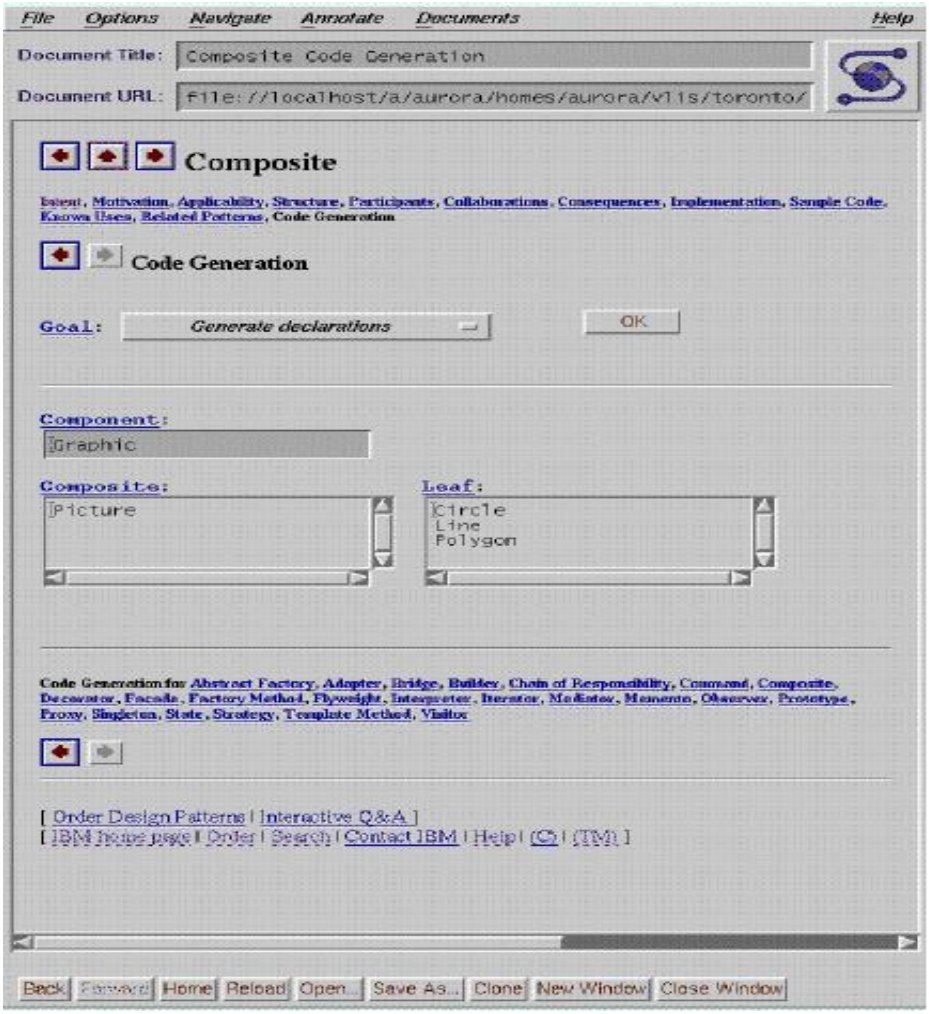

Figura 4. Página de generación de código. [13]

Su implementación se ha desarrollado dentro del entorno Web. Para ello, se han utilizado HTML y PERL. Sus autores le atribuyen las siguientes características:

- Se acomoda a las herramientas y aplicaciones existentes donde quiera que sea posible.
- Hace uso de especificaciones interpretadas para minimizar el cambio durante el tiempo de desarrollo.
- Divide la funcionalidad a fin de que los componentes clave puedan ser distribuidos, permitiendo así la ampliación del sistema sin afectar a los usuarios.[2]

Presenta los mismos inconvenientes que la herramienta anterior, añadiendo además otro: una vez generado el código del patrón, si el usuario añade modificaciones a este código, éstas se pierden al intentar generar de nuevo el código para el patrón.

Estas herramientas anteriormente expuestas se basan en un modelo específico desarrollado por sus desarrolladores, ninguna de estas herramientas se basan en la toma de decisiones para la aplicación de patrones ante situaciones específicas Para desarrollar el modelo propuesto en el objetivo del trabajo se necesita conocer un poco sobre las estrategias y algoritmos basados en la inteligencia artificial para la solución de problemas, en el próximo capítulo se hace alusión a algunos de estos algoritmos basados principalmente en la heurística.

Conclusiones del capítulo

En este capítulo se realizó un análisis del estado del arte sobre los libros, catálogos que abordan los patrones de diseño así como herramientas creadas para la aplicación de los mismos. Como resultado de este estudio se arriba a las siguientes conclusiones:

- Los distintos libros y catálogos de patrones consultados son una buena fuente de conocimientos para el estudio de los patrones. Con el estudio de los mismos se puede llegar mejorar los conceptos de diseño de software, al aplicar los patrones planteados. La mayoría de estos libros y catálogos analizados, los patrones que tocan son los previamente definidos en el catálogo de los cuatro, con esto se puede concluir que estos patrones son los más estudiados y aplicados por la mayoría de desarrolladores en el proceso de desarrollo de software pues estos patrones pueden ser aplicados independiente del lenguaje en que se desarrolle el software.

- Por lo que demuestran las búsquedas no hay muchas herramientas que permitan la aplicación de patrones. Y las existentes no garantizan la ayuda a la toma de decisiones. Pues si permiten a un diseño predefinido aplicar un determinado patrón pero este patrón será seleccionado previamente por el programador o diseñador y lo que se quiere es que la herramienta basada en la información que le dé el usuario, requisitos, tipo de aplicación etc., sea capaz de tomar la decisión de que patrón es posible aplicar en ante la situación dada y muestre al usuario dicha decisión.

Estrategias y algoritmos 2

Hasta el momento se han visto los diferentes enfoques de los patrones de diseño, estos han permitido analizar y estudiar a los patrones permitiendo definir sus características esenciales; debido fundamentalmente a la necesidad que todo programador, diseñador y arquitecto sea capaz de aplicar en sus diseños de software estos patrones de diseño, basándose en sus características es que se pretende construir un modelo que permita el proceso de ayuda a la toma de decisiones en situaciones donde sea necesario utilizar estos. Para la creación de este modelo se necesita la utilización de estrategias y algoritmos, así como técnicas estadísticas que permitan la realización de este proceso.

En este capítulo se realizará un estudio sobres algunas estrategias y algoritmos que usan inteligencia artificial para la solución de problemas. Las estrategias analizadas a continuación se basan en la búsqueda heurística de solución de problemas y en la clasificación sin aprendizaje.

Estrategias y algoritmos basados en inteligencia artificial

Algoritmos de Métodos de solución de problemas.

La solución de problemas es una forma de razonamiento muy compleja que requiere la generación y asimilación de nuevas estructuras de memoria a fin de contestar una interrogante. Es la actividad de encontrar una solución a un problema. En el contexto del procesamiento de la información el enfoque dado a la solución de problemas ha sido tratar de trazar la gráfica de la secuencia de eventos desde la formulación del problema hasta su solución final; o sea, tratar de comprender el proceso que interviene para derivar una solución. [14] En la inteligencia artificial hay diferentes métodos de solución de problemas, por ejemplo se tiene tres alternativas: aleatoria, a ciegas y con heurística. Con el siguiente ejemplo se ilustran estos. Supóngase que está en París sin un mapa y no habla francés ¿Cómo llegar hasta la torre Eiffel?.[14]

- Un método podía ser tomar aleatoriamente una calle esperando que más pronto que tarde se llegará a la torre. Esta búsqueda aleatoria puede llevar a encontrar la torre pero puede requerir una cantidad infinita de tiempo por la forma arbitraria en la cual se selecciono un camino y puede tomarse múltiple veces.

- Otra alternativa es seguir exhaustivamente cada calle de inicio a fin. Cuando se alcanza un final se busca una calle paralela y se sigue esta en dirección opuesta independientemente de si nos acercamos o alejamos del objetivo. Eventualmente esta variante consideraría todas las posiciones de nuestro espacio problema. Este tipo de búsqueda se llama a ciegas, pues no usa conocimiento de cuan cerca se está de la solución, para tomar un determinado camino.

- Alternativamente, se puede usar el conocimiento sobre la torre para mejorar la eficiencia de la búsqueda. Suponiendo que el extremo superior de la torre puede ser visto desde cualquier lugar del espacio problema, se puede tomar la calle que parezca que lleve a la dirección. Esta es llamada una búsqueda con heurística.

En esta sección se analizarán solo los métodos de solución de problemas con las estrategias de búsqueda a ciegas y con heurísticas.

Entre los métodos de búsqueda a ciegas se encuentran:

- Primero en anchura.
- Costo uniforme
- Primero en profundidad.

Los métodos de búsqueda heurística se encuentran:

- Primero el mejor.
- A*

A continuación se analizarán algunos conceptos sobre heurísticas y posteriormente se analizará cada uno de los anteriores algoritmos.

Los métodos de búsqueda heurísticas están orientados a reducir la cantidad de búsqueda requerida para encontrar una solución. Cuando un problema es presentado como un árbol de búsqueda el enfoque heurístico intenta reducir el tamaño del árbol cortando nodos pocos prometedores. [14]

Feigenbaum y Feldman definen la heurística como sigue: “Una heurística es una regla para engañar, simplificar o para cualquier otra clase de ardid el cual limita drásticamente la búsqueda de soluciones en grandes espacios de estados”. En esencia una heurística es simplemente un conjunto de reglas que evalúan la posibilidad de que una búsqueda va en la dirección correcta. Generalmente los métodos de

búsqueda heurísticas se basan en maximizar o minimizar algunos aspectos del problema. [14] Se puede decir que el objetivo principal de la búsqueda heurística es buscar una buena solución pero que esta no será necesariamente la mejor.

La Heurística no garantiza que siempre se tome la dirección de la búsqueda correcta, por eso este enfoque no es óptimo sino suficientemente bueno. Frecuentemente son mejores los métodos heurísticos que los métodos de búsquedas a ciegas. Las desventajas y limitaciones principales de la heurística son:

- La flexibilidad inherente de los métodos heurísticos puede conducir a errores o a manipulaciones fraudulentas.
- Ciertas heurísticas se pueden contradecir al aplicarse al mismo problema, lo cual genera confusión y hacen perder credibilidad a los métodos heurísticos.
- Soluciones óptimas no son identificadas. Las mejoras locales determinadas por la heurística pueden cortar el camino a soluciones mejores por la falta de una perspectiva global. La brecha entre la solución óptima y una generada por heurística puede ser grande.[14]

La heurística es una técnica que ayuda a mejor y aumentar la eficiencia de la ejecución de un proceso de búsqueda, siempre dejando un margen de error en la misma.[15] La aplicación de algoritmos heurísticos ofrece buenos tiempos de ejecución y buenas soluciones usualmente las óptimas.

Antes de pasar a ver cada unos de los algoritmos se necesita abordar un concepto sobre estos y es acerca de la admisibilidad de los mismos, se dice que un algoritmo es admisible cuando garantiza el hallazgo de una ruta óptima entre el nodo de inicio y el nodo meta, si es que ella existe. El teorema de admisibilidad es analizado, con el análisis del algoritmo A*, en los posteriores secciones [16]. La distancia aérea (equivalente a la distancia en línea recta entre dos puntos del mapa) es un ejemplo de heurística admisible.

Estas estrategias de búsqueda se evalúan atendiendo a cuatro aspectos:

- Completitud: Si garantiza o no encontrar la solución si existe.
- Complejidad del tiempo: Cantidad de tiempo que se requiere para encontrar una solución.
- Complejidad del espacio: Cantidad de memoria que se necesita para realizar la búsqueda.
- Optimalidad: Si se encontrará o no la mejor solución en caso de que existan varias.

Se dice que una estrategia de búsqueda es buena si es óptima y completa

A continuación se analizarán estos algoritmos, estos por sus características pueden ser utilizados para desarrollar el modelo que se propondrá en el próximo capítulo.

Primero en profundidad.

Este método siempre expande uno de los nodos del nivel más profundo del árbol. Solamente cuando la búsqueda alcanza un nodo muerto (nodo no objetivo que no se puede expandir) la búsqueda va a atrás y expande nodos de niveles inferiores.

Esta búsqueda tiene requerimientos de memoria modestos. Solamente almacena un camino de la raíz a un nodo hoja junto con los hermanos de los nodos de este camino a cada nivel. Para un espacio de estado con factor de ramificación b y profundidad máxima m se requiere solamente b*m nodos.

La principal ventaja de este método es que sus requerimientos de memoria son linealmente proporcionales a la profundidad del árbol. La cantidad de tiempo que consume es la misma que en primero a lo ancho.

Su principal problema es que puede tomar un camino equivocado que si es muy largo, quizás infinito, no permite encontrar una solución nunca. Muchos problemas tienen árboles de búsquedas muy profundos o incluso infinitos, en cuyo caso este método nunca se podrá recuperar de una selección equivocada en uno de los nodos cercanos a la raíz del árbol.

Otra situación en la que puede caer el método es encontrar una solución con un camino más largo que la solución óptima, es decir, el método no es ni completo, ni óptimo. Este método debe evitarse para árboles de búsquedas muy profundos.

Es un método apropiado cuando los estados objetivos se encuentran a la izquierda del árbol de búsqueda o cuando los caminos son cortos.

Primero en anchura.

Este algoritmo de búsqueda visita cada nodo del árbol por niveles, es decir, visita todos los nodos de un nivel antes de visitar los del siguiente.

En esta estrategia se expande primero el nodo raíz, luego todos los nodos generados se expanden y así sucesivamente con sus sucesores. En general todos los nodos a la profundidad *d* en el árbol de búsqueda son expandidos antes de los del nivel *d+1*. Es una estrategia de búsqueda sistemática porque considera

todos los caminos de longitud 1 primero, luego todos los de longitud 2, etc. Si existe una solución este método garantiza encontrarla y si hay varias soluciones siempre encontrará la menos profunda en el árbol de búsqueda. En términos del cuarto criterio este método es completo.[14]

Para analizar su costo en tiempo y memoria considérese un espacio de estado donde todo estado puede ser expandido en *b* nuevos estados, se dice que el factor de ramificación de estos estados (y del árbol de búsqueda) es *b.* Luego se tendrá *b* nodos en el primer nivel, b^2 en el segundo, hasta b^d en el nivel *d* donde se halla la solución, de modo que el número máximo de nodos expandidos antes de encontrar una solución es:

$1+ b + b^2 + b^3 + \ldots + b^d$

Luego la complejidad es exponencial $O(b^d)$. La complejidad del espacio es la misma que la de tiempo porque todos los nodos hojas del árbol tienen que ser mantenidos en memoria.

Con la búsqueda a lo ancho se asegura que una vez alcanzada una solución no existe otra ruta hacia la solución que tenga menor cantidad de pasos. Una desventaja de este algoritmo es que para alcanzar una solución de n pasos, debe haber explorado todo el espacio de estados hasta ese nivel.

Principales problemas del algoritmo:

- Requiere mucha memoria.
- Requiere mucho trabajo, especialmente si el camino más corto a una solución es muy largo.
- Los operadores irrelevantes (o redundantes) incrementan grandemente el número de nodos que deben explorarse.

La búsqueda en amplitud es particularmente inapropiada en situaciones donde existen muchos caminos que conducen a soluciones, pero cada uno de ellos es muy largo.

Algoritmo Best-First (Primero el mejor)

La búsqueda por el mejor nodo es una forma de combinar las ventajas de las búsquedas en profundidad y a lo ancho en un único método. En cada paso del proceso de búsqueda se selecciona el más prometedor de aquellos nodos que se han generado hasta el momento. Entonces este se expande usando los operadores para generar sus sucesores. Si uno de ellos es una solución se termina. Si no, todos esos nuevos nodos se añaden al conjunto de nodos generados hasta ese momento. Se selecciona de nuevo el nodo más prometedor y el proceso continúa. La selección del nodo a expandir es independiente de la

posición en que nos encontramos en el árbol de búsqueda y de la posición del nodo más prometedor. Lo que sucede usualmente es que se realiza un poco de búsqueda a profundidad mientras se explora una rama prometedora. En un momento dado esa rama comienza a ser menos prometedora que otras de más alto nivel que se han ignorado hasta ese momento.

El nombre de búsqueda primero el mejor (Best-First Search) es impreciso. Si realmente pudiéramos expandir el mejor nodo primero no sería una búsqueda sino un procedimiento directo al estado objetivo.

Este algoritmo es una estrategia de búsqueda avara o glotona prefiere tomar el bocado más grande del costo restante para alcanzar el objetivo sin preocuparse de si será el mejor. Por eso la búsqueda primero el mejor con esta heurística se denomina búsqueda golosa (greedy search).

La búsqueda golosa frecuentemente trabaja bastante bien. Ella tiende a encontrar soluciones rápidamente aunque no siempre encuentre la óptima. Esta búsqueda sufre de los mismos defectos que la búsqueda primero en profundidad, no es óptima y es incompleta. En el peor caso su complejidad en tiempo es $O(b^m)$ donde m es la máxima profundidad del espacio de búsqueda. Como mantiene todos los nodos en memoria su complejidad en espacio es la misma que respecto al tiempo. Estas complejidades se reducen sustancialmente con una buena función heurística.

Búsqueda de costo uniforme

La búsqueda primero a lo ancho encuentra el estado objetivo menos profundo, pero este no siempre es la solución de menos costo. La búsqueda con costo uniforme modifica la búsqueda primero a lo ancho expandiendo siempre el nodo de menor costo en lugar del nodo menos profundo usando una función g(n) que calcula el costo de un nodo. La búsqueda primero a lo ancho es justamente la búsqueda de costo uniforme cuando g(n) = Depth(n).

La búsqueda con costo uniforme encuentra la solución más barata bajo el requerimiento: El costo de un camino nunca decrece al alargarse el camino, es decir:

$g(\text{ sucesor}(n)) \geq g(n)\ \forall\ n.$

Si algún operador tiene costo negativo (si el costo de un nodo se calcula como la suma de los costos de los operadores que sirven para construirlo) sólo una búsqueda exhaustiva podría hallar la solución óptima. Si todos los operadores tienen un costo no negativo la búsqueda con costo uniforme puede encontrar el camino más barato sin explorar el árbol de búsqueda completamente.

Algoritmo A*

La variante de primero el mejor, de búsqueda heurística minimiza el costo estimado al objetivo y reduce el costo de la búsqueda considerablemente, pero no es ni óptima ni completa. Por otro lado, la búsqueda con costo uniforme, ella minimiza el costo del camino hasta el nodo n desde el nodo inicial; Ella es óptima y completa pero puede ser muy ineficiente.

Una buena alternativa es combinar ambas estrategias para tomar sus ventajas. Eso puede hacerse simplemente combinando las dos funciones de evaluación mediante la suma, así se obtiene: f(n) = g(n) + h(n).

Como g(n) da el costo del camino del nodo inicial al nodo n, y h(n) es el estimado del costo del camino más barato desde n hasta el nodo objetivo, entonces f(n) es el costo estimado de la solución más barata a través de n. Se puede probar que esta estrategia es completa y óptima siempre que la función h nunca sobrestime el costo de alcanzar el objetivo. En este caso h se denomina una heurística admisible. Las heurísticas admisibles son por naturaleza optimistas porque ellas piensan que el costo de resolver el problema es menor que lo que es realmente. Este optimismo se transfiere a la función f; si h es admisible f(n) nunca sobrestimará el costo real de la mejor solución a través de n.[14] La búsqueda primero el mejor que usa f como función de evaluación y una función h admisible se conoce como A*.

La relación de A* con las otras búsquedas es la siguiente:

A*	Otras búsquedas
A* con h = 0	Búsqueda de costo uniforme
A* con h = 0 y c(n,n') = 1	Búsqueda primero a lo ancho
A* con f(s) = 0 y f(n') = f(n) - 1	Búsqueda primero en profundidad

Otras consideraciones importantes acerca del algoritmo son:

- A* es óptimamente eficiente, ya que ningún otro algoritmo óptimo expande menos nodos que A*.
- A* es completo o convergente sobre grafos finitos localmente (grafos con un factor de ramificación finito) siempre que todo operador tenga un costo δ como mínimo (δ es una constante positiva). Ambas condiciones garantizan que no haya nodos con infinitos sucesores directos y que no haya un camino con costo finito pero una cantidad infinita de nodos.[14]

- A* usualmente se desborda en memoria antes de desbordarse en tiempo.

Sea $h^*(n)$ el costo de un camino óptimo desde n a un nodo objetivo. Un teorema sobre la admisibilidad de A* dice: Un algoritmo A* que usa una función heurística h tal que para todos los nodos n en el espacio de estado $h(n) \leq h^*(n)$ es admisible.[14]

Teorema de Admisibilidad

Si *h'* nunca sobrestima a *h* entonces A* es admisible. La única manera de garantizar que *h'* nunca sobrestime a *h* es haciéndolo cero, pero con ello estamos en la búsqueda primero en anchura, es decir el algoritmo es admisible, pero no eficiente. Pero existe un corolario que puede ser de utilidad en casos prácticos:

Corolario: Si h' muy rara vez sobrestima h por más de δ entonces el algoritmo A* rara vez encontrará una solución cuyo costo sea mayor más que en δ que el costo de la solución óptima.[17]

Este resultado es de un gran valor práctico pues aún desconociendo el valor exacto de h^* podemos encontrar una cota inferior de h^* y usarla como h en A*. Esto es suficiente garantía para que A* produzca una solución optima. Mientras más cerca esté h de h^* mayor eficiencia se obtiene en la búsqueda. Idealmente si conociéramos h^* y la usamos como h en A* este algoritmo encontrará una solución óptima directamente, sin retrocesos.

Sobre la complejidad de A*.

Para una función heurística monótona Nilsson mostró que A* encuentra el camino óptimo con una complejidad O(n), donde n es la cantidad de nodos posibles en el grafo. Sin embargo, para la mayoría de los problemas prácticos este estimado es demasiado grande.[14]

Una función heurística es casi exacta si una relación inducida por esta ordena el conjunto de todos los nodos en exactamente la misma forma que como lo hace h*. Para funciones casi exactas la complejidad de A* es O(M), donde M es el número de nodos en el camino óptimo (tamaño de la solución).[14]

A* con una función heurística monótona tiene una complejidad exponencial $O(l^m)$ donde l es el factor de ramificación.[14]. Una función heurística h es monótona si:

1. Para todos los estados n_i y n_j donde n_j es descendiente de n_i, $h(n_i) - h(n_j) \leq$ coste(n_i,n_j) coste(n_i,n_j) es el coste del arco que va desde el estado n_i hasta el estado n_j

2. La evaluación heurística del estado meta es cero, h(meta)=0.

De los anteriores algoritmos vistos, se analizo un poco más profundo el algoritmo A*, debido precisamente a que este algoritmo abarca todos los aspectos de los demás haciéndolo más potente y más óptimo. Este algoritmo por sus características podría ser una buena opción para ser utilizado en el desarrollo del modelo que se planteará posteriormente.

Este estudio de diferentes técnicas de inteligencia artificial para ser aplicadas en la ayuda a la toma de decisiones no se quedará solo en los algoritmos de métodos para la solución de problemas basados en heurística, en la próxima sección se estudiaran otras técnicas también de inteligencia artificial pero ahora basadas en la clasificación sin aprendizaje.

Algoritmos de clasificación sin aprendizaje

La clasificación se puede definir como la actividad de agrupar los elementos de información de acuerdo a atributos o propiedades comunes entre los mismos, consiste en establecer la existencia de clases o grupos en los datos. [18, 19]

La clasificación aborda dos líneas: clasificación sin aprendizaje y con aprendizaje.

La clasificación con aprendizaje se aplica a problemas donde el universo de objetos se agrupa en un número dado de clases, de las cuales se tiene información de cada una, una muestra de objetos que se sabe pertenecen a ella y el problema consiste dado un nuevo objeto poder establecer sus relaciones con cada una de dichas clases. [20]

La clasificación sin aprendizaje consiste en: dada las descripciones de un conjunto de objetos de un universo dado, hallar las relaciones intrínsecas de los mismos de modo tal que se pueda develar un conjunto de propiedades no evidentes a partir de sus descripciones. Es decir, se quiere conocer cómo se agrupan los objetos dados, pues precisamente en este problema de clasificación la cuestión más importante que se desconoce es cuales son los agrupamientos que se forman.[21] Resolver un problema de clasificación sin aprendizaje consiste en encontrar la estructura interna de un conjunto de descripciones de objetos en el espacio de representación. Esta estructura depende de una primera instancia, de la selección del propio espacio de representación y de la forma en que sus objetos se comparen, es decir del concepto de similaridad que se utilice. [21] Resumiendo se puede decir que de un problema de clasificación sin aprendizaje se sabe que:

- Por alguna razón puede que se conozca la cantidad de agrupamientos a formar y lo que no se sabe es cuales objetos están en cada grupo

- No se conoce en cuantas agrupaciones se encontrará el conjunto de objetos una vez definidos el espacio de representación y los conceptos de semejanza y la forma de usarlos.

En cualquiera de los dos casos el problema consiste en encontrar una vía, un procedimiento que permita conocer la estructura interna del conjunto de descripciones de objetos dados.

El problema analizado en esta tesis se rige por la segunda variante de la clasificación sin aprendizaje, pues para clasificar los patrones y llevarlos a los agrupamientos que se quieren según sus características y según sus relaciones unos con otros, no se conoce cuantos agrupamientos se pueden formar y tampoco se sabe que patrones pueden estar en un mismo grupo. Una de las técnicas utilizadas para realizar este agrupamiento es el clustering.

La técnica de clustering consiste en agrupar una colección dada de patrones no etiquetados en un conjunto de grupos. En este sentido, las etiquetas están asociadas con los grupos, pero las categorías se obtienen únicamente de las propiedades de los datos. Es conocida como clasificación no supervisada, pues no se tienen asignación de grupos a clases ya predefinidas, sino que los grupos se van creando de acuerdo a las características de los datos.

Los grupos o clusters, son un conjunto de objetos que comparten características similares y juegan un papel muy importante en la forma en que se analiza y describe el mundo que nos rodea. De forma natural, el humano se encarga de dividir objetos en grupos (clustering) y asignar objetos particulares a dichos grupos (clasificación). Clustering es una de las técnicas más útiles para descubrir conocimiento oculto en un conjunto de datos.[22]

A continuación se analizarán varios algoritmos de clasificación no supervisada que son la base de uno de los algoritmos propuestos para el desarrollo del modelo para la toma decisiones que se propone en el presente trabajo. Estos algoritmos son los siguientes:

- Algoritmo c-means.
- Algoritmo CLASS.
- Algoritmo Holotipo.

Algoritmo c-Means

Uno de los algoritmos más utilizados para la técnica de agrupamiento en la clasificación de aprendizaje no supervisado es el C-Means, se basa en minimizar el cuadrado de las distancias de todos los elementos

del agrupamiento al centro del mismo. Esta es una de las formas de maximizar la homogeneidad de cada uno de los agrupamientos, maximizando por ende la heterogeneidad entre los mismos.[21]

Paso 1. El algoritmo consiste en: se selecciona en la primera iteración la cantidad de centros (c) $z_1(1), z_2(1),..., z_c(1)$ para los agrupamientos a formar. La selección de estos primeros elementos, es decir la primera estructuración propuesta para la muestra de objetos, se realiza de manera arbitraria, aleatoria, o siguiendo el criterio de expertos

Paso 2. En el k-ésimo paso de iteración del algoritmo, se distribuyen los elementos de la muestra en los agrupamientos siguiendo el siguiente criterio:

$$O \in S_j(k) si \left\| I(O) - z_j(k) \right\| < \left\| I(O) - z_i(k) \right\| \qquad (1)$$

Para todo $i = 1,...,c, i \neq j;$ donde $S_j(k)$ denota al agrupamiento del cual $z_j(k)$ es el centro. La expresión (1) significa que un elemento del universo se ubica en el agrupamiento cuyo centro se encuentra más cercano a dicho elemento.

Paso 3. A partir de los resultados obtenidos en el paso anterior se calcula los nuevos centros $z_j(k+1), j = 1,...,c,$ de manera tal que la suma de los cuadrados de la distancias de cualquier punto de $S_j(k)$ al nuevo centro de ese agrupamiento sea mínima. Esto es, el nuevo agrupamiento con centro en $z_j(k+1)$ se determina de modo tal que el parámetro j_j sea minimizado.

$$j_j = \sum_{O \in S_j(k)} \left\| I(O) - z_j(k) \right\|^2, conj = 1,...,c \qquad (2)$$

El $z_j(k+1)$ que minimiza la expresión (2) es el valor medio del conjunto $S_j(k)$. De esta forma el centro del nuevo agrupamiento viene dado por la expresión:

$$z_j(k+1) = \frac{1}{N_j} \sum_{O \in S_j(k)} I(O) \text{ donde } N_j = \left| S_j(k) \right| \qquad (3)$$

El nombre del algoritmo, c-means, obedece a la manera en que se van calculando secuencialmente los centros de los nuevos agrupamientos.

Paso 4. Si $z_j(k+1) = z_j(k)$ para todo $j = 1,...,c$ termina el procedimiento con la propuesta de estructuración del universo, de lo contrario se regresa al paso 2.

Es importante que aclarar que el comportamiento del algoritmo, su aplicabilidad y efectividad práctica dependen de una serie de presupuestos que a continuación se enumeran:

- Se debe conocer la cantidad centros de agrupamientos a formar.
- Se tienen que seleccionar, entre los objetos del universo a estructurar, los centros siguiendo criterios de expertos, de manera aleatoria o por medio de alguna heurística.
- La expresión (1) garantiza que los agrupamientos elaborados formen una partición, luego sólo se obtendrán estructuraciones de este tipo.
- El algoritmo, como otros de este tipo y enfoque, presupone que homogeneidad (en el sentido del cumplimiento de las propiedades que caracterizan a los objetos de un mismo agrupamiento) es equivalente a cercanía (en el sentido de la distancia que se defina sobre el espacio de representación de los objetos (ERI)). Eso conlleva a una estrecha vinculación de dependencia entre la selección del ERI, la distancia definida y la calidad real de la solución propuesta.
- La expresión (3) puede llevar a un centro que no pertenezca a la muestra de estudio. Eso puede tener diferentes lecturas, algunas en las que no se le dé mucha importancia al asunto y otras en la que llegue a desconfiar totalmente de la estructuración propuesta. Una solución a este hecho puede venir dada por la decisión de tomar como centro al objeto de la muestra más cercano al centro virtual calculado; pero éste puede ser no único en la hiperesfera de centro en el centro virtual calculado y radio igual al mínimo de las distancias de éste a los objetos de la muestra dada. Esto conlleva a estructuraciones diferentes, ¿Cuál de ellas es la buscada? Por otro lado está suponiendo que dicho centro es un valor central, ya que lo determina mediante el promedio de los valores, y como ya sabemos esto es cierto siempre y cuando la distribución de datos en la muestra sea la menos simétrica.
- No es difícil apreciar que la velocidad con que se alcance la solución estará en dependencia de la geometría de los datos, siendo más factible alcanzar respuestas rápidas y relativamente confiables en los casos en que las descripciones de los objetos queden como empaquetadas en el ERI respecto a la distancia seleccionada.

- El algoritmo no establece qué hacer cuando un objeto equidiste de más de uno de los centros en una iteración cualquiera. En términos de la expresión (1) no se puede decidir la ubicación de dicho elemento.
- No hay un teorema que garantice en el caso general la convergencia del algoritmo.

Algoritmo CLASS

A continuación se trata el algoritmo CLASS elaborado por la especialista soviética Susana Valerianovna Sirotinskaia. Este algoritmo fue elaborado bajo las motivaciones de la prospección geológica y forma parte de una extensa colección de trabajos dedicados al desarrollo de los métodos lógicos del análisis de la información geológica, una línea de investigación aplicada en las que mayor éxito ha alcanzado los métodos lógico-combinatorios. Este algoritmo usa como concepto fundamental uno análogo al de conjunto β_0-compacto. [21]

Descripción del algoritmo

Paso 1. Se calcula los valores de los objetos: $T(O_i, O_j)$, para todos los pares posibles de MI (conjunto de objetos o muestra inicial) es decir, se forma MS (matriz de semejanza).

Paso 2. Se calcula β_0. En el caso del original CLASS, Sirotinskaia usa esta variante:

La magnitud $\beta_0 \in \Delta$ la denominaremos umbral de semejanza y puede ser calculada de la siguiente forma:

$$\beta_0 = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \beta(I(O_i), I(O_j))$$

Estos dos primeros pasos del algoritmo descansan sobre el proceso de formalización del concepto de analogía implícitos en el problema en cuestión, es decir, después de ejecutados los dos primeros pasos se tiene la matriz MS y el umbral de semejanza, con lo cual se puede determinar si dos descripciones son o no β_0- semejantes.

Paso 3. Se obtiene $\beta_i^* = \max\{T(O_i, O_j)\}$. Aquí i recorre las filas y j las columnas de la matriz MS, $i = 1,...,m,\ j \neq i$.

Paso 4. La magnitud β_i^* si se compara con β_0 Si $\beta_i^* < \beta_0$ entonces el objeto O_i se considera aislado y conforma un conjunto unitario. Si $\beta_i^* \geq \beta_0$ se va al paso 5.

Paso 5. Se determina los índices $t = t_i^*$ tales que $\beta(I(O_i), I(O_t)) = \beta_i^*$.

Paso 6. Para cada fila $i, i = 1,...,m$ se forman r vectores del siguiente tipo: $B_s = \left(t_s^*, i_1^s, ..., i_{1_s}^s\right)$

$s = 1,...,r$ donde r es el número de valores diferentes de columnas $t_i^* y, i_1^s, ..., i_l^s$, los índices t_s^*.

Paso 7. El vector B_1, empezando por el Segundo elemento, es decir, t_{1s}^1, se compara con los elementos $t_2^*, ..., t_r^*$ de los vectores $B_{2,....,} B_r$. Si para algún vector $B_{v,} 2 \leq v \leq r, i_1^1 = t_v^*$, entonces se forma de nuevo vector $B_1^{\ 1}, = B_1 \cup B_v$ que tiene la siguiente forma:

$B_1^{\ 1} = \left(t_1^*, i_1^1, ..., i_{1_s}^s, i_1^v, ..., i_{l_v}^v\right)$.

Paso 8. El vector $B_1^{\ 1}$ (o al B_1 si es que en el paso 7 no se incluyó elemento alguno) se pone al final de la lista y se reenumera la misma (pasando ahora a encabezarla el vector B_2 si es que este no fue eliminado en el paso 7). Ahora se aplica el paso 7 para el nuevo primer vector de la lista. Los pasos 7 y 8 se repiten hasta obtener r^* vectores $B_s^{\ *}$ para los cuales $t_i^* \neq i_k^u$; $s, k = 1,...,l_k$. Los elementos de los vectores así obtenidos, serán los índices de los objetos de MI que conforman los subconjuntos compactos buscados A_s. En el caso de que todos los vectores $B_1,...,B_r$ se unan ($r^* = 1$), esto querrá decir que todo el conjunto MI forma un compacto y el procedimiento de estructuración de MI termina. Sin embargo, en el caso en que $r^* > 1$ se va al paso 9

Paso 9. Se calcula las medidas de semejanza entre todos los pares posibles de subconjuntos $A_1,..., A_s$ y se conforma una nueva matriz de semejanza (ahora entre subconjuntos compactos de MI) que denotaremos MS' formada por los valores

$$D_{sv} = \frac{1}{m_s m_v} \sum_{t=1}^{m_s} \sum_{q=1}^{m_v} \beta\left(I(O_t), I(O_q)\right)$$

Donde $O_t \in A_s, O_q \in A_v, m_s = |A_s|, m_v = |A_v|$ y $\beta\left(I(O_t), I(O_q)\right)$ se calculo en el paso 1.

Paso 10. En la matriz de semejanza de compactos MS' se obtiene $D = \max_{\substack{v=1,\ldots,r \\ s \neq v}}$

Después se repiten los pasos 4 y 8 para estos elementos. Como resultado de estas operaciones vamos a obtener nuevas agrupaciones de objetos (un nivel superior al anterior, más débil si se quiere) para los cuales también se pueden repetir los pasos 9 al 10.

La estructuración (jerárquica) del conjunto de objetos de MI en grupos de semejanza se considerará concluida si en el paso 4 para cualquier $s = 1,\ldots,r^*, D_s^* < \beta_0$ o sin el paso 8 todo los grupos se unen.

En resumen el algoritmo permite:

- Resolver el problema del agrupamiento de los objetos MI sobre la base de los criterios de semejanza definidos, en particular, cuando los objetos están descritos en términos de un gran número (más de 100) rasgos.
- No es necesario previamente establecer el número de agrupamiento en los que hay que estructurar a MI.
- La estructuración de MI no depende del orden en que se escojan los objetos para agruparlos. Dado β_0, la agrupación es única a cada nivel.
- El método nos permite obtener una estructuración jerárquica de MI
- Los resultados expresándose en forma de grafo reflejan la dirección del cambio de las propiedades de los objetos de uno a otro.

Más adelante se verá otro algoritmo, diseñado fundamentalmente con el objetivo de resolver el problema de reconocimiento de patrones a partir de una matriz de aprendizaje con una sola clase (el problema tiene por supuesto más de una) y que en su primera parte puede realizar funciones análogas al CLASS, con la diferencia que el holotipo, al cual se hace referencia, usa como concepto básico el de la componente conexa.

Algoritmo Holotipo

El algoritmo holotipo fue elaborado por Yuri Alexandrovich Voronin, Greta Nikolaevna karataeva y Eugenia Naumovna Cheremesina. El algoritmo fue inicialmente desarrollado para resolver problemas de reconocimiento de patrones con aprendizaje, número de clase $l \geq 2$ y se extendió al caso $l = 1$.[21]

La idea general del holotipo se resumen desde el punto de vista geométrico y haciendo uso del concepto de componente conexa, a construir un cubrimiento por hiperesferas de una cierta región n-dimensional de forma compleja. En el centro de cada hiperesfera se ubicará un objeto (denominado holotipo) el cual en la media, será el objeto que más se parece a los restantes del mismo grupo, de la hiperesfera que se construye.

El algoritmo permite:

- Estructurar la MA dada, agrupando sus objetos al estilo que lo hace el CLASS pero en base a los componentes conexa en lugar de los conjuntos compactos.
- Construir una regla de la solución sobre la base del sistema de hiperesferas que construye, que nos permite decidir si un objeto pertenece a la clase dada en MA.
- Establecer un orden entre los objetos que el algoritmo considera que no pertenecen a la clase dada, de modo tal que permitir establecer el grado en el que dichos objetos se alejan de las descripciones que caracterizan a la muestra de las clases dada.

Descripción del algoritmo

Paso 1. Se construye una matriz de semejanza para cada uno de los rasgos $x_1,...,x_n$ en términos de los cuales se describe los objetos admisibles. Como criterio de comparación de los valores de las variables, los iniciadores del algoritmo consideraron:

$$C_k\left(O_i, O_j\right) = 1 - \frac{\left|x_k\left(O_i - x_k\left(O_j\right)\right)\right|}{\Delta x_k}$$

Siendo $\Delta x_k = \max_{O \in MA}\left\{x_k\left(O\right)\right\} - \min_{O \in MA}\left\{x_k\left(O\right)\right\}$, para el caso de rasgos aritméticos y

$C_k\left(O_i, O_j\right) = \begin{cases} 1 & si \quad x_k(O_i) = x_k\left(O_j\right) \\ 0 & en \quad otro \quad caso \end{cases}$ para la variante de rasgos lógicos, $k = 1,...,n; i, j = 1,...,m$.

En general no tenemos que restringirnos a un criterio particular, por muy socorrido que este sea en un área de conocimientos dados. Entre otras razones porque aquí pueden predominar criterios matemáticos apriorísticos en el momento de la modelación matemática de dichos problemas.

Obviamente las matrices $MS(x_k)$ calculadas en este paso son cuadradas, de orden $m \times m$, simétricas, 1 en la diagonal principal y valores del intervalo $[0,1]$ en el resto.

Paso 2. Como quiera que los rasgos $x_1,...,x_n$ pueden tener valores asociados que reflejan su heterogeneidad informacional, consideremos $\delta_k, k = 1,...,n,$ valores de pesos asociados a cada x_k. En cualquier caso, se construye la matriz de semejanza $MS,$ haciendo

$$MS = \sum_{k=1}^{n} \delta_k MS(x_k)$$

Paso 3. Determinación del umbral β_0 para la agrupación de objetos dentro del $MA.$ El valor del umbral se determina por cualquier variante de estas:

$$\beta_0 = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \beta\left(I(O_i), I(O_j)\right)$$

$$\beta_0 = \frac{1}{m} \sum_{i=1}^{m} \max_{\substack{j=1...m \\ i \neq j}} \left\{\beta\left(I(O_i), I(O_j)\right)\right\}$$

$$\beta_0 = \min_{\substack{i=1...m-1 \\ i \neq j}} \left\{ \min_{j=i+1...m} \left\{\beta\left(I(O_i), I(O_j)\right)\right\} \right\}$$

Paso 4. Determinación de los subgrupos dentro de la clase dada en MA haciendo uso del concepto de componentes conexa. Sean $G_{1,...,}G_r$ los subgrupos formados por $r \geq 1$.

Conclusiones del capítulo

En este capítulo se realizó un análisis sobre las distintas técnicas y estrategias basados en inteligencias artificiales para la solución de problemas, como métodos a aplicar en los sistemas de ayuda a la toma decisiones. Como resultado de este estudio se arriba a las siguientes conclusiones:

- Los diferentes algoritmos analizados, tanto los basados en la heurística como los de clasificación sin aprendizaje son una buena vía para ser aplicados para la creación del modelo que permita la ayuda a la toma de decisiones.
- La búsqueda heurística garantiza disminuir la gama de soluciones posibles a un problema
- La clasificación sin aprendizaje permite clasificar información permitiendo agrupar la misma por sus rasgos propios, garantizando así la realización de la toma de decisiones.
- Cada técnica tiene sus características propias, analizando estas características se ha decidido escoger la técnica de la clasificación sin aprendizaje para ser aplicada en el desarrollo del modelo de ayuda a la toma de decisiones, pues esta es la que más se ajusta al problema a resolver, permitiendo clasificar los patrones de diseño, esta clasificación se realizara a partir de las características propias de los mismos permitiendo agruparlos hasta clustering.

Modelo para la selección de patrones de Arquitectura. 3

En este capítulo se presenta, se realiza una caracterización de los patrones de diseño descritos en el libro de los cuatro y en base a esta caracterización se elabora una encuesta como herramienta de captura de información relevante para la arquitectura de un sistema, y esta información capturada será el punto de entrada al modelo desarrollado, el mismo consta de dos algoritmos que serán analizados en este capítulo.

Características que distinguen la aplicación de determinado patrón.

Cada patrón debido a la intención y problemática que resuelven, tienen características que lo identifican y lo hacen único pero a su vez también presentan otras características que lo hacen relacionarse con otros patrones, estas peculiaridades de los mismos es la que permite dada determinada situación poder decidir que patrones usar según las características expuestas. A continuación se realiza un análisis de las características propias de cada patrón así como su similitud y diferencias con otros patrones de diseño.

En todos los sistemas o aplicaciones desarrolladas, en el proceso de su implementación siempre hay creación, representación y composición de objetos; en muchos casos los sistemas llegan hacer poco flexibles debido precisamente a este proceso; para tratar este tema del proceso instanciación es que se definen los patrones de diseño creacionales. Algunas características comunes de estos patrones son:

- Ayudar a hacer los sistemas independientes de como sus objetos son creados, compuestos y representados.
- Encapsulan todo el conocimiento sobre las clases concretas que los sistemas usan.
- Ocultan el proceso de creación de las instancias de esas clases.

Debido a esto los patrones de creación dan mucha flexibilidad en cuanto lo que se crea, quien lo crea, como lo crea y cuando.

Hasta ahora se han visto estos patrones de forma general es decir por sus aspectos comunes pero no todos a pesar de estar en el mismo grupo tienen las mismas características, también presentan características específicas de cada uno de ellos. A continuación se presenta una **¡Error! No se encuentra el origen de la referencia.** que evidencias las características de estos patrones.

Tabla 2. Resumen de las características específicas de los patrones creacionales.

Patrones	Características específicas
Factory method y abstract factory	Factory method: deja a las clases definir su dominio de objetos. Es decir le permite a las clases definir sus subclases correspondientes. Se utiliza para crear cualquier tipo de objeto donde los mismos pueden ser simples o compuestos. Abstract factory: trabaja en la creación de familias de objetos permitiendo hacer los sistemas independientes de como sus productos se crean, componen y representan, resultando fácil cambiar de familia de productos. Crear un objeto que sea la composición de muchos otros, que los hacen parte de este objeto padre, es decir el objeto compuesto representa la familia de los demás objetos que lo componen, un ejemplo claro de la utilización de este patrón es en los objetos conexión de ADO.net, este presenta una interfaz IDbConnection de la cual heredan oracleConection y sqlConnection, las cuales tienen cada una de ellas una serie de objetos que son específicos a cada una; una vez creada la clase oracleConection o sqlConnection automáticamente los objetos que se deriven de ellas (oracleCommand, oracleTrasaction o sqlCommand, sqlTrasaction) se crearán, pues estas clases representan la familia de dichos objetos.
Builder y factory method	No se encarga simplemente de devolver un objeto como el factory method sino de construir una composición de objetos enteras por compleja o sencilla que sea, permitiendo separar la construcción y la representación de un objeto complejo, permitiendo que el mismo proceso de construcción sirva para crear diferentes representaciones. El caso más habitual es el de construir una interfaz de usuario, pero no se limita únicamente a componentes visuales.
Singleton	Su diferencia radica en que permite crear objetos que tengan una sola instancia global a la misma, por ejemplo cuando se quiere acceder por un solo punto a una base de datos o se quiere acceder a una impresora desde la red o se quiere que un objeto sea accedido desde múltiples

	formas.
Prototype	Este proceso es diferente a los demás pues la creación se realiza duplicando al objeto, es decir clonando una instancia creada previamente. Se duplica el objeto incluyendo su estado actual. Por ejemplo se utiliza para realizar la operación de copiar objeto, pues para realizar esta operación se necesita realizarle una copia al mismo.

Estas diferencias no solo ocurren en este grupo de patrones, la misma situación se repite en los patrones estructurales y de comportamiento. A continuación se abordará este tema.

Como características o aspectos generales de los patrones estructurales se encuentran las siguientes:

- Forman grandes estructuras a partir de la composición de objetos y clases.
- Las clases de estos patrones usan herencia para componer interfaces e implementaciones.
- Describen formas de componer objetos para que realicen nuevas funcionalidades.

Una vez vistas las características que identifican al grupo de patrones de estructuras, se verán las características específicas para cada uno de los patrones que se encuentran en este grupo, junto con sus semejanzas y diferencias. Estas características son resumidas en la **¡Error! No se encuentra el origen de la referencia.**.

Tabla 3. Resumen de las características específicas de los patrones estructurales.

Patrones	Semejanzas	Diferencias
Decorator y composite	Utilizan la composición recursiva para organizar un número abierto de objetos.	El decorator permite agregar responsabilidades a los objetos sin la utilización de subclases; evita la explosión de subclases que puede levantarse de intentar cubrir cada combinación de responsabilidades estáticamente. Por ejemplo, un textview y al cual le puedes agregar la funcionalidad de scrollbar o

		ponerle borde negro. Composite tiene un intento diferente. Se enfoca en estructurar las clases para que elementos diferentes sean tratados uniformemente y puedan tratarse múltiples objetos como unos solo, su enfoque no está en el embellecimiento y si en la representación. Por ejemplo tienes un gráfico que está compuesto por letras, líneas y rectángulos y todos estos objetos necesitan ser tratados de la misma forma y no como objetos aislados. Nota: Estas características son distintas pero complementarias. Por consiguientes estos dos patrones frecuentemente se usan juntos.
Decorator y proxy	Ambos describen cómo proporcionar un nivel de indireccionamiento a un objeto. La implantación de ambos mantiene una referencia a otros objetos a los cuales ellos reenvían las peticiones.	El Proxy a diferencia del decorador, no se preocupa por adjuntar o descartar propiedades dinámicamente y no es diseñado por composición recursiva. Su intención es proporcionar un estándar para sujetos para cuando sea inoportuno o indeseable acceder a ellos directamente. Por ejemplo: Objetos que se encuentren en máquinas diferentes, cargar imágenes o ficheros grandes o se tienen objetos que consumen mucha memoria.
Adapter y bridge	Ambos promueven flexibilidad proporcionando un nivel de indireccionamiento a otro objeto.	El adaptador enfocas sus responsabilidades en resolver incompatibilidades entre dos interfaces. No se enfoca en como esas interfaces son

	Involucran re-envió de peticiones, demandas a objetos desde una interfaz a otra.	implementadas, ni considera como ellos pueden variar independientemente, es una manera de hacer que dos clases independientemente diseñadas trabajen juntas sin tener que re-implementar una o la otra. Por ejemplo se tiene un editor de dibujo que tiene, lineshape, polygonshape, textshape. Los dos primeros son fáciles de implementar pero el tercero no, pues este muestra y edita texto sin embargo una interfaz toolkit, provee una clase potente para esto (textview) pero el textshape no puede rehusar esta clase pues tiene una interfaz diferente. Por lo que se necesita la unión de ambas para un mejor funcionamiento Bridge por otro lado desvincula las abstracciones de las implementaciones; proporciona interfaces estables a clientes que varíen la implementación de las clases. Se pondrá un ejemplo para aclara la idea anterior. Se tiene una interfaz que muestra información basada en cálculo, es necesario tener esta interfaz desacoplada de la implementación pues así si la implementación varía, la interfaz no se afecta.

En la tabla anterior se quedó por analizar dos patrones de este grupo, que abarca los patrones estructurales, no se analizaron juntos con los demás pues estos no presentan características de semejanzas con los demás a no ser las características que identifican al grupo. A continuación se presenta la tabla 4 que muestra las características específicas para estos patrones.

Tabla 4. Resumen de las características específicas de los patrones estructurales.

patrones	Características específicas
Flyweight	Define una estructura pero para compartir objetos. Los objetos son compartidos por lo menos por dos razones: eficacia y consistencia, objetos cuyo costo de almacenamiento sea muy alto por lo que crear un gran número de ellos consume mucha memoria.
Facade	Permite unificar una interfaz para un conjunto de interfaces en un subsistema, minimizando la comunicación y dependencia entre subsistemas. Por ejemplo Exciten muchas dependencias entre clientes y clases de implementación de una abstracción. Por esa razón se introduce una fachada para desacoplar el subsistema desde el cliente y otros subsistemas.

Todo sistema propicia la comunicación e interacción entre objetos pero si esta comunicación no se define de la mejor forma es el sistema pierde reusabilidad, flexibilidad; para mejorar esta definición es que surgen los patrones de comportamiento garantizando un bajo acoplamiento entre estos objetos. A continuación se describirán los patrones de comportamiento.

Las características generales de los patrones de comportamiento son las siguientes:

- Conciernen los algoritmos y la asignación de responsabilidades entre los objetos.
- No solo describen patrones de objetos o clases sino patrones de comunicación entre ellos.
- Caracterizan un flujo de control complejo, que es difícil seguir en tiempo de ejecución.
- Se concentran solo en la forma que los objetos son interconectados.

Las características específicas de estos patrones se encuentran relacionadas en la **¡Error! No se encuentra el origen de la referencia.**5, así como sus semejanzas y diferencias.

Tabla 5. Resumen de las características específicas de los patrones de comportamiento.

Patrones	Semejanzas	Diferencias
Strategy, state, mediator e iterator	Encapsulan la variación (Cuando un aspecto de un programa cambia frecuentemente, estos patrones definen un objeto que encapsula los aspectos, entonces otras partes del programan pueden colaborar con el objeto siempre que ellos dependan de ese aspecto.) Describen aspectos de un programa que pueden cambiar	Strategy: encapsula un algoritmo y los hace intercambiables, permitiendo variar el algoritmo independientemente del cliente que lo use; facilitando especificar y cambiar el algoritmo usado en un objeto. Por ejemplo Existen varios algoritmos para dividir un flujo de texto en líneas (**linebreaking**), escribir cada algoritmo en las clases que los necesite no es recomendable. Por lo que se definen clases que encapsulen cada algoritmo de linebreaking diferente, siendo llamado cada uno de ellos strategy State: encapsula un comportamiento estado-dependiente. Permitir a un objeto modificar su comportamiento a medida que su estado interno va cambiando, dando así la impresión de que el objeto "cambia de clase". Por ejemplo una clase TCPConnection tiene varios estados y cuando esta recibe una petición responde en dependencia del estado en que se encuentra. Mediator: encapsula el protocolo entre dos objetos, es decir encapsula como un conjunto de objetos interactúa, evitando el acoplamiento entre ellos. Se puede usar

		mediator cuando se tiene un cuadro de dialogo que tiene muchas dependencias con sus widgets (botones, menús, campos de entrada). El objeto mediator se encarga de encapsular las interconexiones de esos objetos (botones, menús, campos de entrada). Iterator: encapsula la forma de acceder a un conjunto de elementos secuénciales, permitiendo no dar a conocer su representación interna. Se utiliza para recorrer colecciones de elementos.
Chain of responsability y mediator	Permiten la pérdida de acoplamiento	A diferencia del mediator que se utiliza para encapsular la interacción de un conjunto de objetos el Chain of responsability: proporciona a más de un objeto la capacidad de atender una petición, para así evitar el acoplamiento con el objeto que hace la petición. Se forma con estos objetos una cadena, en la cual cada objeto o satisface la petición o la pasa al siguiente. Un ejemplo es que en múltiples ocasiones, un cliente necesita que se realice una función, pero o no conoce al servidor concreto de esa función o es conveniente que no lo conozca para evitar un gran acoplamiento entre cliente y servidor, y este patrón garantiza que se cumpla esto.
Command y memento	Definen objetos que actúan como recuerdos que fueron pasados o invocados tiempo	En el command el recuerdo representa una petición. Permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor

	atrás	real de la misma. Por ejemplo en una interfaz toolkit que incluye objetos como botones y menús, llevan a cabo peticiones en respuesta a una salida de usuario, pero el toolkit no puede implementar la petición explícitamente en el botón o menú pues solo la aplicación que use el toolkit conoce como hacerlo. Por lo que esta implementación se le deja al objeto command, que será el encargado de realizarla el botón o el menú solo tendrán que llamar a este objeto. En el memento, el recuerdo es el estado interno, y el memento representa el estado interno de un objeto en un momento específico por ejemplo el estado de una operación. Ejemplo: En un editor gráfico se tienen elementos como cajas o recuadros que se pueden conectar entre sí por medio de flechas o líneas discontinuas. Existe la posibilidad de mover de posición todo (o partes de) un dibujo, siendo requisito indispensable que las relaciones entre los distintos elementos del mismo se mantengan, a pesar de variar su situación en el documento. Más aún es deseable que un traslado de elementos de un dibujo pueda ser deshecho fácilmente y con total eficiencia, esto es, que todo quede como al principio. Como el patrón memento permite guardar el estado interno, estas acciones pueden realizarse con la aplicación de este patrón.

Mediator y observer	Permiten la comunicación entre objetos	El observer distribuye comunicación, es decir: define dependencia de objetos de uno a muchos, así cuando un objeto cambia su estado, todas sus dependencias son notificadas y actualizadas automáticamente. Por ejemplo varias toolkit para representar datos numéricos pero de diferentes formas, en una tabla, gráfica de barras, y es necesario que si los datos cambian se actualicen todo lo que dependa de ellos. En este caso lo que se hace es introducir un objeto observe que capture los cambios y actualice las dependencias. Mediator encapsula la comunicación entre objetos. El observer promete más reusabilidad que el mediator. Por otro lado, es más fácil de entender el flujo de comunicación en el Mediador que en el Observador.
Template method e interpreter	Distribuir comportamiento entre las clases	Template method: es una definición abstracta de un algoritmo. Este define algoritmos paso a paso, cada paso invoca una operación abstracta o primitiva. Se puede usar en las aplicaciones de flujos de trabajo donde hay muchas interfaces comunes y el comportamiento se repite una y otra vez. Interpreter: representa una gramática como una jerarquía de la clase. Dado un lenguaje, define una representación para su gramática junto con un intérprete que

		usa dicha representación para interpretar sentencias en ese lenguaje.
Visitor		Permite representar una operación que está pensada para ser aplicada sobre los elementos de una estructura de objetos, permitiendo así definir y añadir un nuevo comportamiento sin la necesidad de cambiar las clases de los elementos de la estructura de objetos. Se usa cuando una "Estructura De Objetos" contiene muchas clases con diferentes interfaces y se quieren añadir operaciones a dichos objetos en función de la interfaz.

Hasta aquí se han dado todas las características que identifican a cada uno de los patrones de diseños. Una vez identificadas las características de los patrones, se necesita de un modelo matemático que trabaje con ellas para lograr la identificación de patrones en situaciones específicas. En el próximo epígrafe se desarrollara este modelo teniendo como punto de entrada estas características vistas.

Modelo matemático para la identificación de patrones.

Una vez identificadas las características principales que describen los patrones estamos en condiciones de crear un modelo que permita la toma de decisiones para la identificación de situaciones donde se debe aplicar un patrón de diseño.

Para desarrollar este modelo se partirá de la información obtenida en el proceso de captura de requerimientos donde se obtendrá información relevante para la arquitectura. El modelo consta de dos variantes de algoritmos que son la clave del desarrollo del mismo. Estos algoritmos no se realizan paralelamente, estos pueden ser usados indistintamente a gusto del que los vaya a implementar, siempre y cuando se tenga los datos que estos llevan. Estos algoritmos permiten trabajar con la información relevante obtenida de la encuesta realizada en el proceso de captura de información de forma tal que le da una respuesta al usuario de los patrones que puede utilizar en el desarrollo de su software. A continuación se darán dos variantes de algoritmos que se pueden seguir y llegar a la selección de los patrones.

Variante I: Basada en la captura de información y en la relación entre los patrones.

Variante II: Algoritmo de clasificación por el método difuso no supervisado

Estos algoritmos de forma general tienen sus diferencias y similitudes. La Variante I a partir de las características arquitectónicas del sistema a desarrollar, identificadas durante el levantamiento de requisitos, determina la relevancia independiente de cada patrón para la resolución del problema completo. Calcula además un valor de relevancia de cada patrón teniendo en cuenta la correlación de este patrón con otros patrones relevantes para la situación específica. Finalmente establece una combinación lineal de los dos valores de relevancia para dar como resultado el grado de aplicabilidad de cada patrón en la resolución del problema planteado. La Variante II: Este algoritmo tiene como base los algoritmos de aprendizaje no supervisado e incorpora elementos de la lógica borrosa para su funcionamiento. Básicamente se construyen clusters de patrones a partir de la similitud entre ellos y la situación específica que se presenta. La solución que provee este algoritmo no es un patrón aislado sino un conjunto de patrones que deben ser usados.

El modelo consta de la siguiente estructura, expuesta en la figura 5:

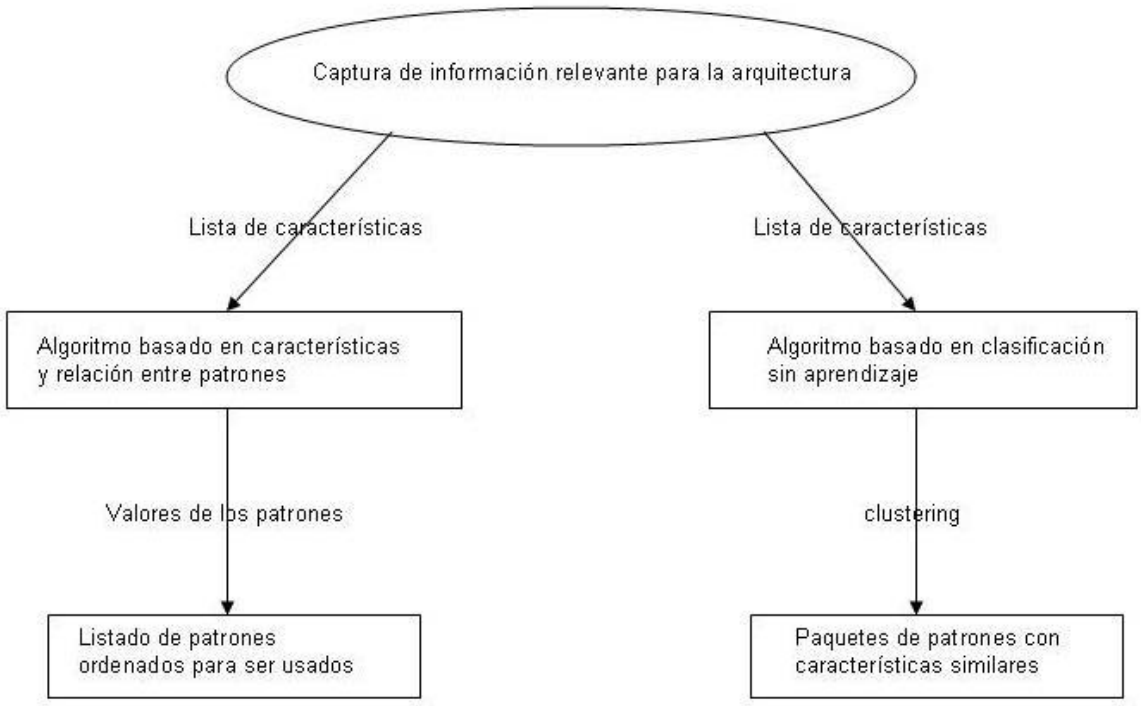

Figura 5. Estructura del modelo para la toma de decisiones en situaciones donde se apliquen los patrones de diseño.

Notaciones

M : Matriz patrón contra características, que indica el grado de presencia de la característica en un patrón, es decir se tiene un conjunto de características que pueden presentarse en determinado patrón o patrones, la frecuencia de aparición de las mismas es diferentes en cada patrón por lo que a esta frecuencia se le asigna un valor y dicho valor es el que se almacenará en la matriz.

C_j : Característica que puede presentar en un patrón en una situación en específico, la cual se encuentra expuesta en la herramienta utilizada en el proceso de captura de información (ver anexo 1).

P_i : Patrón que se esté tratando.

R : Matriz patrón contra patrón que indica en la medida que un patrón se relaciona con otro. El valor almacenado en esta matriz determina el grado de relación de dos patrones para ser utilizados juntos, es decir si uno está presente en qué medida puede estar presente el otro.

m : Cantidad de características.

n : Cantidad de patrones.

V_i : Vector que almacena el valor asignado por el encuestado a cada una de las características.

Vp : Vector que almacena los valores de cada patrón calculados a partir del valor de las características y de la matriz que presenta el grado de presencia de una característica en un patrón.

Vp_{acum} : Vector que almacena los valores de cada patrón pero calculados a partir de la relación de los patrones, la medida de esta relación se encuentra almacenado en la matriz patrón vs. patrón.

Vf : Vector que almacena los valores finales de los patrones.

$B(P_i, P_j)$: Distancia entre los patrones P_i , y P_j distancia que está determinada por la similitud entre dichos patrones.

μA(c): Función de membresía correspondiente a los términos lingüísticos. Esta función describe el grado de membresía del valor real c al conjunto A.

Proceso de Captura de Información relevante para decisiones de arquitectura

Identificar la funcionalidad y las cualidades que un producto o sistema software ha de satisfacer, es el punto de partida obligado de todo proyecto de desarrollo. Si los desarrolladores no conocen de forma precisa el problema a resolver, es probable que la solución que se obtenga no sea la esperada por los usuarios. La Ingeniería de Requisitos es la parte de la Ingeniería del Software que aborda el problema de la identificación y definición de los servicios y de las restricciones operativas del sistema a desarrollar. [23] La Ingeniería de Requisitos comprende cinco pasos para su desarrollo los cuales son:

- Captura de requisitos
- Análisis de los requisitos y negociación.
- Especificación de requisitos
- Gestión de requisitos.
- Validación de requisitos

La captura de requisitos es una de las actividades más importantes organizadas en el proceso de ingeniería de requerimientos. Una buena captura de requisitos influye de forma positiva en las decisiones que se llevarán a cabo para la realización de la arquitectura de un software y por tanto esto garantizará una buena obtención del producto.

La captura de requisitos se basa en varias técnicas para su desarrollo, las técnicas tradicionales incluyen el uso de cuestionarios y encuestas, entrevistas y análisis de información existente, otra técnica muy común es la tormenta de ideas. Estas técnicas permiten que el proceso de captura de requisitos sea más eficiente, además permiten obtener información relevante para ser utilizada posteriormente en el proceso de diseño y tomar decisiones de arquitectura.

Para desarrollar un mejor proceso de captura de requisitos y utilizando una de estas técnicas se desarrolló una encuesta de arquitectura (ver anexo 1) que permitirá identificar características específicas que el software deberá cumplir. En esta encuesta el encuestado deberá darle un valor de 0 a 10 a las características, cuyo valor en este intervalo estará dado por la medida en que cada característica corresponda con el software a desarrollar. La información obtenida en esta encuesta será muy importante para el posterior desarrollo del software y ayudará a tomar ciertas decisiones de arquitectura en el proceso de diseño. La herramienta puede ser aplicada a diferentes personas, como por ejemplo a stakeholders

que se vinculan con el proceso de automatización, según a la persona que se le aplicará se le asignará un grado de experticidad para tener idea de cuan buena es el resultado de la aplicación de la herramienta.

Una vez analizado el proceso de captura de información se pasará al analizar los algoritmos propuestos.

Variante I

Esta variante parte de la herramienta aplicada en el paso de captura de información, a partir de esta herramienta se creará la lista de características con los valores especificados por el usuario, en dependencia de los valores de las características se calculará la relevancia del patrón para el problema específico. Concluido el proceso de puntuación de los patrones se actualizará la evaluación de los mismos según su correlación, es decir cada patrón puede estar relacionado con otros y la puntuación de uno influirá sobre la puntuación de los que estén relacionados con él. Para concluir se ordenará la lista de patrones según la evaluación acumulada, y se obtendrá una lista con los patrones organizados según los más propensos a utilizar.

Algoritmo V1

Precondiciones

Antes de efectuar este algoritmo se dispone de una matriz M de patrones contra características donde cada celda $M_{i,j} \in [0,1]$ indica el grado de relación del patrón P_i con la característica específica C_j, un valor cercano a cero indicará una baja relación entre el patrón y la característica, un valor cercano a 1 indicará una alta relación entre patrón y característica.

Formato de la matriz M (La matriz con datos se encuentra expuesta en los anexos (ver anexo 2) está representada la transpuesta de la matriz):

	C_1	C_2	C_3	C_4	.	C_m
P_1						
P_2						
P_3						
P_4						

Donde $M_{i,j}$ es el valor de la característica C_j en el patrón P_i; tal que $n < m$.

n : Cantidad total de patrones.

m : Cantidad total de características.

.

P_n

Se dispone también de la matriz R de patrones contra patrones donde cada celda $R_{i,j} \in [0,1]$, indican el grado de relación entre los patrones, un valor cercano a cero indicará una baja relación entre los patrones, un valor cercano a 1 indicará una alta relación entre ellos.

Formato de la matriz R (La matriz con datos se encuentra expuesta en los anexos (ver anexo 3) está representada la transpuesta de la matriz):

P_1 P_2 P_3 P_4 . P_n

P_1

Donde $R_{i,j}$ es el valor de

P_2 relación de P_i en el P_j .

P_3

P_4

.

P_n

Estas dos matrices se construyen de forma prescriptiva y dependen de los patrones que se consideren a aplicar, para el caso que se está analizando, las matrices que se formaron fueron con respecto a los patrones de diseños descritos en el libro de GOF.

Descripción del algoritmo

Paso 1. Crear la lista de características a partir de la herramienta aplicada en el paso de Captura de información (ver anexo1).

Paso 2. Actualizar el valor de cada característica con el valor que especificó el usuario en el proceso de captura de la información. Es decir a cada característica se le asignará el valor correspondiente creándose el vector $V_i \in [0,m]$, donde este tendrá los valores de las características.

```
Repetir hasta cantidad de características
  Lista Características[i] ←Valor
Fin Repetir
```

Paso 3. Crear matriz de cambio Mc para almacenar el valor que se calculará tomando el valor de las características V_i identificado en el proceso de captura de información. Las celdas de la matriz Mc recibirán el valor correspondiente a la multiplicación del valor M_{ij}, matriz patrón característica por el valor acumulado de la característica j correspondiente, esta multiplicación se realiza de la siguiente forma: cada elemento de la lista se multiplicará por cada celda de una fila de la matriz y este proceso se repetirá para cada fila, ver el algoritmo siguiente para una mejor comprensión:

```
Cantidad de patrones = j
Repetir hasta Cantidad de patrones
  Repetir hasta Cantidad de características(i)
    Mc_{i,j} ← M_{i,j} * V_i
  Fin Repetir
Fin Repetir
```

Este paso tiene un orden de complejidad de $O(n,m)$ pues se realizan dos ciclos anidados cuya máxima iteración es $n \times m$.

Paso 4. Calcular el valor acumulado del patrón para lo cual se sumarán todos los valores de las celdas correspondientes a la fila del patrón específico. El vector Vp guardará el valor acumulado de todos los patrones.

```
Repetir hasta cantidad de patrones
```

$$Vp_i \leftarrow \sum_{j=1}^{m} Mc_{i,m}$$

```
Fin Repetir
```

Paso 5. Normalizar el vector que almacena el valor de los patrones, calculados a través del valor de las características y de su presencia en los patrones, es decir llevar los valores a una escala entre 0 y 1.

$$Vp_i \leftarrow \frac{Vp_i}{\max(Vp_i)}$$

Al culminar este paso tenemos en el vector Vp la relevancia de los patrones tomada de forma aislada sin considerar la interrelación entre ellos.

En los siguientes pasos se calculará la relevancia de los patrones pero considerando la interrelación entre ellos.

Paso 6. Crear matriz de cambio Rc para almacenar el valor que se calculará a partir de la relación de los patrones entre si y de su valor analizados de forma independiente. Para calcular este valor se tomará la matriz R, las celdas de la matriz Rc recibirán el valor correspondiente a la multiplicación del valor R_{ij} por el valor del vector Vp, esta multiplicación se realiza de la siguiente forma: cada elemento del vector se multiplicará por cada celda de una fila de la matriz y este proceso se repetirá para cada fila, ver el siguiente algoritmo para una mejor comprensión.

```
Cantidad de patrones = j=i

Repetir hasta cantidad de patrones

  Repetir hasta cantidad de patrones
```

$$Rc_{i,j} \leftarrow R_{i,j} * Vp$$

```
  Fin Repetir
Fin Repetir
```

En este paso se realizan dos ciclos anidados con la mimas cantidad de iteraciones en ambos casos, por lo que tiene una complejidad de $O(n^2)$.

Paso 7. Calcular el valor acumulado del patrón dependiendo de influencias mutua de los patrones. Se calculará sumando todos los valores de la celda correspondientes a la columna de un patrón, guardándose en un vector Vp_{acum} .

```
Repetir hasta cantidad de patrones
```

$$V_{p_{acum\ i}} \leftarrow \sum_{n=1}^{j} Rc_{i,n}$$

```
Fin Repetir
```

Paso 8. Normalizar el vector que almacena el valor de los patrones, calculados a partir de la correlación entre los mismos, es decir llevar los valores a una escala entre 0 y 1.

$$Vp_{acum_i} \leftarrow \frac{Vp_{acum_i}}{\max(Vp_{acum})}$$

Paso 9. Calcular el valor final del valor de los patrones, dependiendo este valor, del valor del patrón calculado por sí solo y del valor del patrón calculado con la influencia de otros patrones. Antes de realizar el cálculo se debe normalizar los vectores

$Vf \leftarrow \alpha Vp_n + \beta Vp_{acum\,n}$, donde $\alpha + \beta = 1$ y $\alpha, \beta \in [0,1]$

Los valores α, β se definen en dependencia a como se quiera calcular el valor, por ejemplo si se quiere calcular el valor sin tener en cuenta la interrelación entre los patrones se le da a β el valor de 0 y a α el valor de 1, si se quiere calcular teniendo en cuenta solo esta relación pues se le da el valor de cero a α y uno a β .

Paso 10. Ordenar la lista de patrones dependiendo de su valor almacenado en el vector Vf . Este ordenamiento se realizará de mayor a menor garantizando obtener la lista con un orden de prioridad de los patrones, permitiendo este orden garantizar obtener en qué medida se puede aplicar más un patrón u otro, en el problema planteado. El algoritmo de ordenamiento que se utilizará será el QuickSort, debido a su eficacia y rapidez en dar una solución.

Este algoritmo tiene una complejidad de $O(n \log n)$

Paso 11. Devolver la lista de los patrones organizados.

Finalmente el algoritmo descrito tiene como máxima cuota de complejidad: $O(n, m)$

Diagrama del algoritmo

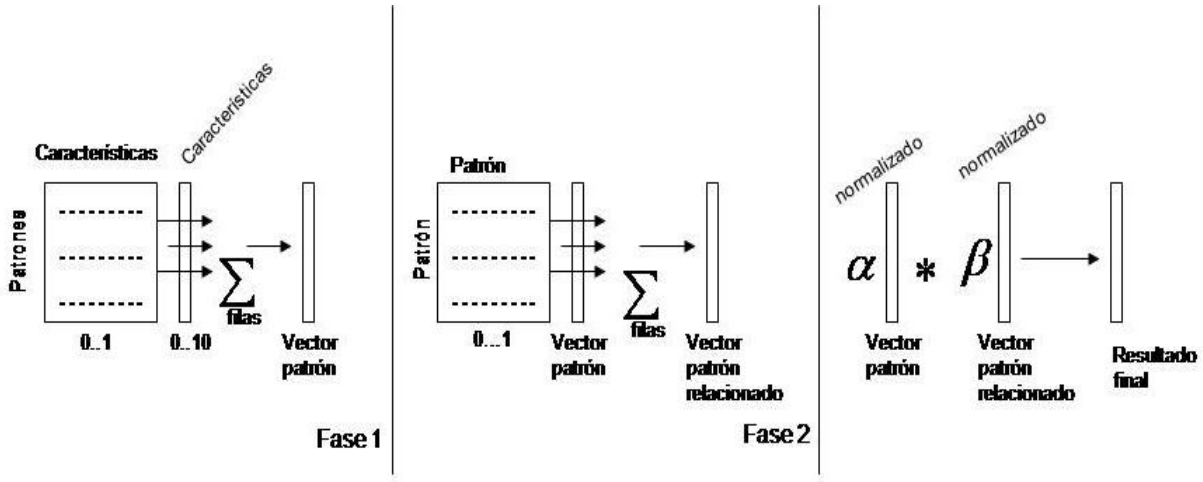

Figura 6. Diagrama del algoritmo presentado.

Breve análisis del algoritmo

El presente algoritmo tiene potencialidades y debilidades, una de sus potencialidades es que el mismo no desecha ninguna solución pues da la misma en base a todos los patrones solo que organizados en función del grado de utilización y deja al usuario escoger por su propia decisión. Su principal debilidad radica en que a medida que aumenta el número de características y de patrones crece la complejidad computacional.

Variante II

Esta variante consiste en aplicar clustering a partir del conjunto de patrones y sus características, para esto se utilizó un Algoritmo de clasificación no supervisado, este es creado partiendo de las bases de otros algoritmos clásicos clasificación sin aprendizaje, como son el Class, el Holotipo, el C-Means entre otros, tratados en el pasado capítulos 2. El algoritmo parte de la construcción de una matriz que almacena el valor de una característica que presenta el patrón, cuyo valor parte del proceso de captura de información, posteriormente se definen de forma prescriptivas las variables lingüísticas y las funciones de pertenencias asociadas a los valores de las características, para determinar en qué medida una característica está presente en un patrón determinado. Teniendo estas medida se pasa a la formación de clustering los cuales en un principio estarán formados cada uno por un solo patrón y posteriormente se unirán por el grado de semejanza, cuyo valor de semejanza estará determinado por las variables lingüísticas y sus respectivas funciones de pertenencias. Una vez concluido este proceso se tendrá una serie de clustering que representarán los posibles patrones a utilizar en una determinada situación.

Algoritmo V2

Determinación de clusters de objetos ClusPat.

Paso 1. Construir la matriz inicial M_{ij} partiendo de los datos entrados por el usuario.

Formato de M_{ij}

$C_1 \quad C_2 \quad C_3 \quad C_4 \quad \cdot \quad C_m$

P_1

P_2

P_3

P_4

.

P_n

Donde P_iC_j es el valor de la característica C_j en el patrón P_i; tal que $n < m$.

n : Cantidad total de patrones.

m : Cantidad total de características.

Paso 2. Construir las variables lingüísticas para cada uno de los atributos continuos.

Las variables lingüísticas se pueden definir de varias formas, de forma automática, buscando un epsilon o simplemente de forma prescriptiva, la forma prescriptiva será la utilizada en dicho algoritmo.

De forma prescriptiva se define las variables lingüísticas como: bajo, medio y alto, estos términos significan que:

- Bajo: La característica tiene un bajo nivel de identificar el patrón.
- Medio: Presentación media de la característica en el patrón.
- Alta: La característica tiene un elevado grado de presencia en el patrón.

Teniendo las variables lingüísticas definidas se pasa a definir de forma prescriptiva las funciones de pertenencias asociadas a dichas variables. Como el valor de las características se encuentra en el intervalo de 0 a 10, las funciones quedan definidas a través del siguiente gráfico:

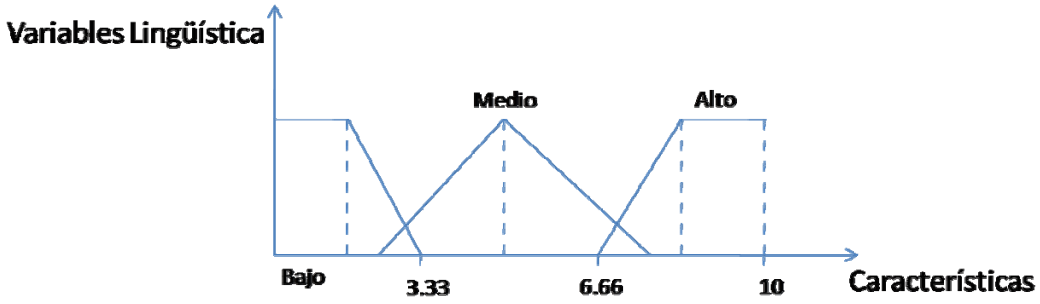

Figura 7. Función de pertenencia asociada a las variables lingüísticas.

En forma de recomendaciones se puede sugerir construir las variables lingüísticas a través de la definición de un epsilon, cuyo proceso se describe a continuación:

Para definir las variables lingüísticas primeramente es necesario construir las funciones de pertenencia asociadas a las variables lingüísticas. Como primer paso para construir dichas funciones es necesario determinar los intervalos discretos en que podemos agrupar los valores de los atributos. La estrategia tradicional más sencilla para construir dichos intervalos es definir un epsilon y agrupar los datos mientras las distancias entre estos sean

menor que el epsilon especificado. El problema entonces se convierte en encontrar el epsilon apropiado para cada problema y dominio de la variable en cuestión. Se propone una solución alternativa que permite agrupar los valores con independencia del problema en cuestión que se trate.

Dado el conjunto original de datos o valores numéricos de un mismo atributo ordenarlos

Calcular un valor de umbral β_0 como se muestra $\beta_0 = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left| a_i{}^c - a_j{}^c \right|$ donde n es la cantidad de patrones, y por tanto la cantidad de posibles valores de la característica c. Los valores a_i y a_j representan valores posibles de c.

Paso 3. Calcular el valor del umbral que se utilizará como criterio límite para la unión de dos clusters:

$H = \frac{2}{m(m-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} B(P_i, P_j)$ $0 \leq H \leq 1$

El cálculo de este umbral parte del cálculo de las distancias entre los patrones. La distancia entre los patrones P_i y P_j se denota como $B(P_i, P_j)$ su cálculo se muestra en la Ecuación 1, donde esta es la función de semejanza de los patrones.

Ecuación 1

$B(P_i, P_j) = \sum_{k=1}^{m} sim(c_k{}^{Pi}, c_k{}^{Pj})$ siendo m la cantidad de características a tener en cuenta, $c_k{}^{Pi}$ la característica C_k del patrón P_i

Similitud entre atributos

Considerando que los atributos son numéricos teniendo en cuenta un enfoque borroso durante la comparación. Vale señalar que siguiendo el enfoque borroso para establecer la comparación entre dos valores reales se establece primeramente una asociación entre el valor real y un término lingüístico definido de antemano y aclarando que si el atributo hubiera sido simbólico su valor coincidiría con el término lingüístico. La función de comparación entre atributos toma finalmente la forma de la ecuación 2.

Ecuación 2

$$sim(c_a, q_a) = \begin{cases} 1.0 & \alpha = \beta \\ 1 - \dfrac{|\mu_\alpha(c_a) - \mu_\alpha(q_a)| + |\mu_\beta(c_a) - \mu_\beta(q_a)|}{2} \end{cases}$$

Donde $\alpha = \arg\max(\mu_A(c_a))$ y $\beta = \arg\max(\mu_A(q_a))$ (principio de máxima membrecía). Es decir α y β son los términos lingüísticos donde la función de pertenencia toma su máximo valor para c_a y q_a respectivamente. Donde $\mu A(q_a)$ representa la función de pertenencia del conjunto borroso A con atributos numéricos.

Para mejor comprensión se debe señalar que el valor de semejanza entre objetos cumple que: $0 \le B(P_i, P_j) \le 1$.

En este paso se realizan dos sumatorias donde una llega hasta n, cantidad de patrones y la otra hasta n-1, por lo que la máxima cuota de orden de complejidad es de $O(n^2)$.

Paso 4. Construir la matriz de semejanza o matriz de cambio (**MS**) donde las filas y columnas serán agrupaciones o clusters de patrones. Se denota $T_i k$ al cluster k en la iteración i.

Formato de MS

	T_1	T_2	T_3	T_4	.	T_m
T_1	1					
T_2		1				
T_3			1			
T_4				1		
.					1	
T_n						1

Inicialmente en la iteración 1 existirán tantos clusters como patrones, formándose un cluster por cada patrón que exista en MI, o sea n = m y el cluster T_1k contendrá al patrón k. Cuando los clusters contienen un solo patrón este es considerado el holotipo del cluster. El elemento i, j de la matriz MS representa la distancia entre el cluster P_ps, y el cluster P_pr, donde s, r ∈ [1,m] y *p* representa al número de la iteración. Para calcular la distancia entre los clusters P_ps, P_pr se determinan los holotipos de los mismos y se usa la función de semejanza entre los patrones descrita en la ecuación 1. Nótese que la diagonal principal de esta matriz estará formada solamente por 1 indicando la máxima semejanza entre clusters.

Paso 5. Agrupar los clusters más semejantes. Para esto, se determinar en la matriz MS, el mayor valor de semejanza entre clusters (*maxval),* este valor se determina a partir de la distancia entre los holotipos de los clusters, que no es más que la distancia entre dos patrones descrita en la ecuación 1. Para cada par de clusters cuyo grado de semejanza sea *maxval*:

```
Si maxval  > H

 entonces Crear un nuevo cluster a partir de la unión del  par de
clusters con semejanza maxval

sino Terminar
```

Para la determinación del holotipo de un clusters, si este está formado por un solo patrón en un inicio, como se había planteado dicho patrón sería el holotipo del clusters, el holotipo no es más que el patrón que es simétrico a todo los demás. Para el caso en que el clusters este formado por más de un patrón se trata el clusters desde el punto de vista geométrico como una hiperesfera r-dimensional, donde el holotipo sería el patrón más cercano al centro de la hiperesfera (ver gráfico 2).

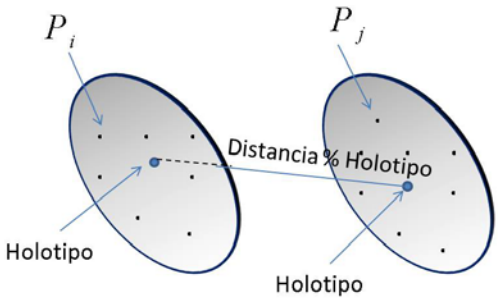

Figura 8. Representación geométrica del cluster.

En este caso el holotipo será el objeto que minimiza la ecuación 3:

Ecuación 3

$$LMD(P_i) = \sum_{j=1}^{l} B(P_i, P_j), donde \quad l = |T_p k| \text{ donde } P_i,\ P_j \text{ pertenecen al cluster } T_p k$$

Estos dos últimos pasos se estarán repitiendo hasta que se cumpla la condición de parada, que pueden ser las siguientes:

- Que se cumpla un número de iteraciones preestablecido.
- Que el número de clusters formados hasta el momento sea igual a número preestablecido como parámetro.
- Chequear que no se puedan formar más clusters a partir del análisis del umbral.

Estos dos últimos pasos tienen un de orden de complejidad de $O(n\ )$ pues se realiza un ciclo para recorrer las matriz, cuyo máximo valor que alcanza la iteración es de n.

Finalmente el algoritmo descrito tiene como máxima cuota de complejidad: $O(n^2)$

Una vez concluido este algoritmo se obtendrán diferentes grupos de patrones con características específicas que determinaron el agrupamiento de los mismos para poder ser usado en la problemática en que se apliquen. Dándole una decisión al desarrollador sobre cual grupo de patrones poder aplicar.

Análisis de los algoritmos

Una vez planteado los dos algoritmos por los que está integrado el modelo propuesto se realizará una pequeña comparación de los mismos. Para realizar esta comparación se tendrán en cuenta los siguientes puntos:

- Complejidad de los algoritmos.
- Respecto a la claridad.
- Respecto a la eficiencia.

Comparación:

1. Complejidad de los algoritmos: Como ya se había analizado anteriormente, el primer algoritmo tiene un orden de complejidad de $O(n,m)$ y el segundo de $O(n^2)$, lo cual demuestra que el primer algoritmo tiene un orden de complejidad mucho mayor que el segundo.
2. Respecto a la claridad: Respecto a la claridad, habiendo visto ambos algoritmos se puede decir que el primero tiene mayor claridad a la hora de aplicarlo pues todo se resume a las simples operaciones matemáticas de suma y multiplicación mientras que el segundo ya se necesita de algunos conocimientos de Inteligencia Artificial para poder entenderlo y así poder aplicarlo.
3. Respecto a la eficiencia: En cuanto a la eficiencia cada uno tiene sus particularidades, pues por ejemplo el primero en base a los datos obtenidos en el proceso de captura de información y a la relación de unos patrones con otros, brinda una solución, pero dicha solución se basa sólo en cálculos estadísticos mientras que el segundo sólo utiliza los datos recogidos en el proceso de captura de información pero el proceso que aplica para obtener la solución es muy utilizado y reconocido por expertos.

A partir de esta comparación de los dos algoritmos planteados se selecciono el primero para ser implementado con el objetivo de analizar su funcionamiento. Para realizar este análisis se tomaron algunos ejemplos de sistemas realizados y otros que están en proceso.

Ucidrez:

Primeramente se tomó el sistema Ucidrez creado en el curso 2003-2004, al aplicarle la herramienta de captura de información se obtuvieron los siguientes resultados, relacionados en la siguiente tabla cuya

primera columna corresponde con el número de las características que se ofrecen en la encuesta y la segunda con el valor de la característica dados por el encuestado.

N. Caract	Valor
1	10
2	10
3	10
4	10
5	10
6	0
7	0
8	0
9	0
10	0

N. Caract	Valor
11	5
12	2
13	10
14	10
15	0
16	10
17	0
18	0
19	0
20	0

N. Caract	Valor
21	0
22	0
23	0
24	0
25	0
26	6
27	0
28	0
29	0
30	0

N. Caract	Valor
31	10
32	10
33	10
34	10
35	10
36	10
37	10
38	10
39	10
40	10

N. Caract	Valor
41	0
42	0
43	0
44	0

Los patrones utilizados en el sistema Ucidrez fueron los siguientes: Factory Method, Strategy, Singleton, Command, State, Flyweight.

Al procesar esta información a través del algoritmo se obtuvo como resultado que, según las características que presenta el sistema y las funcionalidades que cumple, debería utilizar principalmente los siguientes patrones de diseño, Factory Method, Flyweight, Statregy, Singleton, State, Command, Abstract factory, Proxy, Observer.

Como se puede observar la lista de patrones obtenida no está lejos de los patrones que se usaron. Al comunicarle a uno de los desarrolladores el resultado del algoritmo, informó que algunos de los patrones que había ofrecido el algoritmo no se usaron debido a su falta de experiencia, posteriormente al desarrollo del sistema hicieron un estudio más profundo llegando a la conclusión que debían haber aplicado estos otros patrones también.

Por tanto, los resultados obtenidos al aplicar el algoritmo a este proyecto fueron satisfactorios, dando un grado de confianza del mismo.

Proyecto Energía:

El algoritmo también se aplicó al proyecto Energía. A continuación se realiza el análisis del mismo:

Resultados obtenidos en la encuesta:

N. Caract	Valor
1	9
2	10
3	1
4	0
5	5
6	9
7	10
8	9
9	10
10	7

N. Caract	Valor
11	4
12	6
13	3
14	10
15	10
16	0
17	0
18	0
19	0
20	8

N. Caract	Valor
21	8
22	7
23	7
24	8
25	0
26	8
27	0
28	4
29	10
30	10

N. Caract	Valor
31	10
32	10
33	0
34	0
35	10
36	8
37	0
38	0
39	5
40	5

N. Caract	Valor
41	0
42	8
43	8
44	8

Los patrones utilizados en el proyecto Energía son los siguientes: Singleton, Facade, Observer.

Al procesar la información se obtuvo que los patrones que se deben usar según las características, son los siguientes, Factory Method, Facade, Memento, State, Singleton, Command, Proxy, Prototype.

Con el análisis de estos resultados, se puede observar que coinciden los patrones obtenidos con los que se aplicaron pero el algoritmo ofrece más opciones, lo cual puede ser por varias razones, que el algoritmo no sea suficientemente eficiente o que los desarrolladores del sistema de Energía en su momento no contaban con los conocimientos suficientes para no aplicar dichos patrones.

Proyecto ONE:

En otro proyecto que se aplicó el algoritmo fue en ONE, proyecto que se encuentra en desarrollo en estos momentos. Después de aplicar la encuesta de Arquitectura se obtuvieron los siguientes resultados:

N. Caract	Valor
1	9
2	10
3	1
4	0
5	5
6	9
7	10
8	9
9	10
10	7

N. Caract	Valor
11	4
12	6
13	3
14	10
15	10
16	0
17	0
18	0
19	0
20	8

N. Caract	Valor
21	8
22	7
23	7
24	8
25	0
26	8
27	0
28	4
29	10
30	10

N. Caract	Valor
31	10
32	10
33	0
34	0
35	10
36	8
37	0
38	0
39	5
40	5

N. Caract	Valor
41	0
42	8
43	8
44	8

Los patrones utilizados hasta el momento son los siguientes: Singleton y Facade. Al aplicar el algoritmo se obtiene la siguiente lista de patrones: Facade, Factory Method, Singlenton, Builder, Template, State, Iterator. Se puede observar que los dos patrones utilizados hasta el momento en el proyecto son tenidos en cuenta por el algoritmo.

Proyecto Registros y Notaría

Finalmente se presentan los resultados de aplicar el algoritmo al Módulo de Mercantil del proyecto Registros y Notaría. Los Resultados de la encuesta fueron los siguientes:

N. Caract	Valor
1	5
2	6
3	10
4	6
5	5
6	0
7	0
8	0
9	10
10	0

N. Caract	Valor
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0

N. Caract	Valor
21	0
22	3
23	0
24	0
25	0
26	1
27	0
28	0
29	0
30	0

N. Caract	Valor
31	10
32	0
33	0
34	0
35	0
36	0
37	0
38	0
39	0
40	0

N. Caract	Valor
41	0
42	10
43	10
44	10

Los patrones obtenidos con la aplicación del algoritmo resultaron ser: Facade, Factory Method, Statregy, Command, Template, Mediator. Al encuestar a los desarrolladores del proyecto se percibió que los patrones utilizados en este Módulo fueron Strategy, Template, Command, Facade. Por tanto la solución brindada por el algoritmo es correcta.

A partir del análisis realizado en cada uno de los proyectos mencionados se llega a la conclusión que el algoritmo Variante 1 presenta una eficiencia con un alto grado de validez por lo que puede ser utilizado en el proceso de toma de decisiones en el desarrollo de los diseños de arquitectura de los productos software.

A modo de conclusión se puede ver que con la solución obtenida en los algoritmos se mejora la calidad del diseño, pues con ella el diseñador, desarrollador o arquitecto puede llegar a aplicar los patrones de diseño sin tener incluso muchos conocimientos sobre ellos y al ser los mismos soluciones ya probadas disminuye la introducción de errores en el proceso de diseño, mejorando así la flexibilidad del software y el proceso de gestión de cambio, pues es más fácil hacer un cambio a un software que tiene un modelo de diseño bien estructurado, entendible y sobre todo flexible que sobre un modelo que no es claro y no es lo suficientemente flexible.

Conclusiones del capítulo

En este capítulo se le dio cumplimiento al último objetivo propuesto en la tesis: Elaborar un modelo que permita la identificación y caracterización de situaciones donde se requiera la aplicación de los patrones de desarrollo de software.

En el capítulo se realizó una caracterización de los patrones de diseños; para la realización de esta caracterización se dividieron los patrones de diseños en los tres grupos que se tratan en el libro de los cuatro y a partir de esto se sacaron sus semejanzas y diferencias permitiendo buscar la relación entre los mismos. Pues gracias a esta relación se puede determinar grupos de patrones a ser utilizados en las situaciones donde se hagan necesarios. Para un mejor entendimiento de estas características, se relacionó la característica con ejemplos prácticos que se dan en la vida diaria de un programador. Esta caracterización es la base para el desarrollo del modelo que permita la ayuda a la toma de decisiones.

Como segunda parte de este capítulo se propusieron dos algoritmos, uno de ellos basados en una técnica de inteligencia artificial: clasificación sin aprendizaje aplicando clustering. Estos dos algoritmos parten de la identificación de las características de cada patrón, y basándose en las mismas y en la relación entre los patrones es que permiten realizar el proceso a la toma de decisiones en situaciones específicas. Cada algoritmo tiene sus peculiaridades y no hay ninguno superior al otro, la diferencia entre ambos es que siguen técnicas diferentes para llegar al resultado.

Conclusiones

Se realizó un estudio del estado del arte sobre los patrones de diseño, las distintas herramientas que permiten la aplicación de estos patrones en los modelos de diseños desarrollados, y también se realizó un estudio sobre las distintas técnicas y estrategias basadas en inteligencia artificial que pueden ser utilizadas en la ayuda a la toma de decisiones. Este estudio es la base para la creación del modelo en el capítulo tres. A partir de este estudio y análisis se propusieron objetivos que fueron cumplidos y se arriba a las siguientes conclusiones:

- Se desarrollo un modelo que permite la ayuda a la toma de decisiones en situaciones específicas donde se apliquen los patrones de diseño, finalmente se concluye en:

  - La caracterización hecha a cada patrón permite identificar los patrones de diseño de una forma más asequible para los desarrolladores, diseñadores, arquitecto o cualquiera que desee usar los mismos, además que permite relacionar unos patrones con otros, permitiendo siempre buscar el vinculo entre ellos y de esta forma tener los patrones agrupados de una forma que sea mucho mejor trabajar con ellos.

  - Los algoritmos propuestos permiten realizar el proceso de ayuda a la toma de decisiones basándose en la caracterización de los patrones de diseño y de rasgos distintivos de cada situación a programar.

  - La unión de la caracterización de los patrones de diseño con el desarrollo de los algoritmos es lo que conforma el modelo propuesto en la tesis.

Recomendaciones

- Incorporar otras clasificaciones de patrones por ejemplo patrones de análisis, de implementación y organización.
- Mejorar el instrumento desarrollado para la identificación de características y posibles patrones en un problema, no pensar solo en un instrumento encuesta sino en un instrumento entrevista y entonces como entrevista presentar posibles preguntas.
- Buscar e incorporar otras variantes de algoritmos que utilicen inteligencia artificial y valorar su aplicación.
- Realizar a partir del modelo un sistema que permita la ayuda a la toma de decisiones en situaciones donde se apliquen determinado patrón.
- Crear una base de datos con datos reales de diferentes proyectos, con los patrones utilizados y con información del proceso de captura de información relevante para la arquitectura.
- Mejorar los algoritmos permitiendo que estos aprendan a través de una base de conocimientos.

Referencias bibliográficas:

1. Liebener, L. and M.A. Rossi, *plinker. Relaciones entre los patrones de diseño*, in *Facultad de Ciencias Exactas*. 2003, Universidad Nacional del Centro de la Provincia de Buenos Aires: Buenos Aires.
2. Martínez, J.S., *Una Arquitectura para una Herramienta de Patrones de Diseño*, in *Dpto. de Informática y Sistemas*. 1999, Universidad de Murcia. .
3. *Patrones de Diseño en Aplicaciones Web con java J2EE*. http://www.programacion.com/java/tutorial/patrones/
4. Devis, R., *Patrones de Diseño*. http://www.arrakis.es/~devis/patrones.ppt
5. Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1995.
6. Cooper, J.W., *Introduction to Design Patterns in C#*. 2002.
7. *IBM. Rational Software*. http://www.neovalia.es/imagenes/ficheros/ibm/RATIONAL.doc
8. *DEV396 Essentials of Rational Software Architect*. 2005.
9. *SanFrancisco performance: A case study in performance of large-scale Java applications*. http://researchweb.watson.ibm.com/journal/sj/391/christ.html
10. *Las frameworks San Francisco se hacen realidad.* http://researchweb.watson.ibm.com/journal/sj/391/christ.html
11. Monkchips, J.G.s. (2005) *Time for IBM to restart the San Francisco Project?* http://redmonk.com/jgovernor/2005/11/07/time-for-ibm-to-restart-the-san-francisco-project/
12. Budinsky, F.J., et al., *Automatic code generation from design patterns*. 1996. http://www.research.ibm.com/journal/sj/352/budinsky.html
13. Pérez, D.R.B., *Curso de Métodos de Solución de Problemas para la Inteligencia Artificial*, U.C.d.L. Villas, Editor. 1998.
14. *Búsqueda Huerística*. http://gda.utp.edu.co/pub/ia/Presentaciones/IA8.ppt#263,6,7

15. Becke., C.v.d. (2000) *Admisibilidad*. http://club.telepolis.com/ohcop/admisibi.html

16. Arteaga, R. and J.C. Armijos (1998) *Búsqueda heurística*. http://www.monografias.com/trabajos/iartificial/pagina3_2.htm

17. Hernández, A.G. (2004) *Aprendizaje Automático: Clasificación.* http://www.uv.mx/aguerra/teaching/MIA/MachineLearning/clase02.pdf

18. Montero, Y.H. and F.J.M. Fernández (2004) *Sistemas de Clasificación de Información*. http://www.nosolousabilidad.com/articulos/sistemas_clasificacion.htm.

19. Rodríguez, H.D., et al. *Sistema para la Clasificación basado en Técnicas de Reconocimiento de Patrones*. http://www.uo.edu.cu/fac/fie/info/articulos/TELEC2000P107.PDF.

20. Ruiz-Shulcloper, D.J. and M.J.F. Martínez-Trinidad, *Clasificación sin aprendizaje y con aprendizaje parcila. (Enfoque lógico-combinatorio)*. 1995, Centro de investigación y estudios avanzados. Instituto Politécnico nacional de Mexico.

21. Valadez, E.H., *Algoritmo de clustering basado en entropía para descubrir grupos en atributos de tipo mixto*. 2006: Mexico. http://www.cs.cinvestav.mx/Estudiantes/TesisGraduados/2006/tesisEdnaHernandez.pdf.

22. Escalona, M.J. and N. Koch (2002) *Ingeniería de Requisitos en Aplicaciones para la Web – Un estudio comparativo*. http://lsiweb.lsi.us.es/docs/informes/LSI-2002-4.pdf

23. *Patrón de diseños: Singleton.* http://www.lsi.us.es/docencia/get.php?id=608.

24. *Introducción a la Inteligencia Artificial.* http://dmi.uib.es/~abasolo/intart/1-introduccion.html#1.2.

25. *Reconocimiento de patrones. h*ttp://usuarios.lycos.es/ealonsop/archivos/ia/SB2_RP.ppt.

26 *Aprendizaje y Adaptación.* http://ccrma.stanford.edu/~juanig/articles/charlAndes/Aprendizaje_y_Adaptacion.html

27. *Ingeniería de Requisitos*. http://www-gris.det.uvigo.es/~jose/doctorado/re/sld001.htm

28. *Ingeniería de Requisitos.* http://fuerteventura.ls.fi.upm.es/~anunez/ficheros/Requisitos.pdf

29. Marqués José M. *Ingeniería del Software I.* http://www.infor.uva.es/~jmmc/ingsoft/isrequis.html

30. Molpeceres Alberto. *Diseño de software con patrones.* http://www.programacion.com/java/articulo/joa_patrones3/#joa_patrones3_prototipo

31. García PerezSchofield Baltasar, *Desarrollo rapido de aplicaciones: Patrones de diseño*. http://trevinca.ei.uvigo.es/~jgarcia/cdRAD/transpas-patrones.pdf

32. León Welicki, *Patrones y Antipatrones: una Introducción - Parte II,* http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3317.asp

Anexos

Anexo 1: Encuesta de arquitectura

Encuesta a realizar.

1. Se necesita dejar a las clases definir su dominio de objetos. Es decir se les permite a las clases definir sus subclases correspondientes.
2. Proceso de creación de objetos.
3. Presentamos clases que difieren en su comportamiento. Es decir se tiene un proceso donde las entradas y las salidas serán las mismas y lo que puede variar es la forma de hacer este proceso.
4. Permitir variantes de un mismo algoritmo.
5. Presentamos uso de múltiples sentencias condicionales. Se tienen varias actividades que según la condición que se cumpla se realizarán una u otra.
6. Necesitamos agregar funcionalidades a un objeto que no necesariamente debe tenerlas por defecto. Se tiene un concepto con varias funcionalidades pero no se quiere que las mismas estén siempre presentes, se necesita que se puedan agregar en tiempo de ejecución, evitando así que estén presentes en todo momento.
7. Se Necesita trabajar con componentes simples y compuestos de manera uniforme. Se tienen elementos bases y otros que son derivados de los mismos y se tienen procesos que trabajan con ellos sin interesarle esta característica de los mismos.
8. Construcción de objetos complejos a partir de objetos simples y similares entre sí.
9. Acceder secuencialmente a una colección de objetos sin exponer su estructura interna. Se tiene un gran número de elementos de un mismo tipo y se necesita acceder a ellos para realizar cualquier tipo de operación.
10. Define la estructura de un algoritmo, permitiendo a las subclases que lo usan redefinir algunos pasos. Es decir se tiene un proceso dividido por pasos, que nunca van a cambiar y lo que puede variar es la forma de realizar dichos pasos.

11. Proporcionar una interfaz para la creación de familias de objetos. Se tienen elementos que son los que agrupan a otra serie de elementos, formándose como una familia.
12. Proporcionar una librería de clases de productos. Se quiere proporcionar un paquete de elementos que todos pertenezcan a un mismo grupo o familia.
13. La clase solo tiene una instancia y proveer un punto global de acceso a la misma. Se tiene un elemento que es único por lo que solo puede ser usado por uno a la vez.
14. Controlar el acceso a un recurso físico único. Por ejemplo se tiene una única impresora.
15. Se necesita cargar imágenes grandes o ficheros grandes.
16. Acceder a un objeto remoto.
17. Proveer un sustituto a un objeto y controlar el acceso a él. Se quiere controlar el acceso a determinado elemento.
18. Permitir que clases con interfaces incompatibles puedan trabajar de conjunto. Se tienen elementos de diferentes contextos pero a su vez estos elementos pueden trabajar juntos.
19. Crear una clase reusable que coopere con clases inesperadas.
20. Desacoplar una abstracción desde su implementación tal que puedan variar independientemente. Por ejemplo se tiene que mostrar una información basada en cálculos, los cuales pueden cambiar en cuanto a la forma de obtenerlos pero esto es ajeno al cliente.
21. Se necesita crear diferentes representaciones a partir de un mismo proceso de construcción.
22. Tenemos un conjunto de objetos cuya comunicación entre los mismo es muy compleja. Se tiene un grupo de elementos que se comunican entre sí.
23. Interdependencias entre objetos poco estructurados y difíciles de entender. Mucha dependencia entre elementos.
24. Tenemos objetos con muchas referencias a otros.
25. Se necesita encapsular el comportamiento de todo un conjunto de objetos en un solo objeto.
26. Tenemos un objeto con varias dependencias y se necesita que cuando el objeto cambie, cambien todas sus dependencias.

27. Una petición es atendida por más de un objeto.
28. Cadena de objetos que satisfacen una petición o la pasan al siguiente.
29. Se necesita Capturar y exteriorizar el estado interno de un objeto. Por alguna razón se necesita guardar el estado interno de un elemento.
30. Se desea facilitar el hacer y deshacer de determinadas operaciones
31. Separar el código de la interfaz usuario de las acciones. Se tiene un proceso donde se envían solicitudes sin conocer exactamente la operación solicitada ni el receptor de la solicitud.
32. Se necesita clonar instancias de objetos creadas previamente. Duplicar elementos.
33. Se necesita añadir/eliminar productos concretos /clases dinámicamente. Es decir se quiere configurar una aplicación en tiempo de ejecución.
34. Se necesita crear una instancia de una clase pero es un proceso muy complejo y requiera mucho tiempo por lo que se crea una copia de la misma.
35. Se tiene un objeto que su comportamiento de depende del estado del mismo.
36. Existen operaciones con largas sentencias condicionales con múltiples ramas que dependen del estado del objeto.
37. Añadir a los objetos de una estructura operaciones muy diferentes y sin relación alguna entre ellas, sin necesidad de hacerlo a nivel individual.
38. Objeto compartido que puede ser usado en diferentes contextos simultáneamente, de forma transparente, indistinguible. Se tiene un elemento que puede ser compartido para ser utilizado por otros elementos.
39. Los costes de almacenamiento son prohibitivos como consecuencia a que la aplicación tiene gran numero de objetos.
40. Eliminar o reducir la redundancia entre objetos que contienen información idéntica.
41. Definir una representación para la gramática de un lenguaje junto con un intérprete que usa dicha representación para interpretar sentencias en el mismo.
42. Se necesita una interfaz unificada para un conjunto de interfaces en un subsistema.

43. Se necesita minimizar la comunicación y dependencia entre subsistemas.

44. Exciten muchas dependencias entre clientes y clases de implementación de una abstracción.

Anexo 2: Matriz patrón vs. Características.

Nota: Las columnas representan a los patrones y las filas a las características representadas en la encuesta. Los patrones están distribuidos en el siguiente orden: factory method, statregy, decorator, composite, Iterator, template, abstract factory, singleton, proxy, adapter, bridge, builder, mediator, observer, chain of responsability, memento, command, prototype, state, visitor, flyweight, interpreter, facade respectivamente.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
19	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Anexo 3: Matriz Patrón vs. Patrón.

Nota: Los patrones están distribuidos en el siguiente orden: factory method, statregy, decorator, composite, Iterator, template, abstract factory, singleton, proxy, adapter, bridge, builder, mediator, observer, chain of responsability, memento, command, prototype, state, visitor, flyweight, interpreter, facade respectivamente, tanto para las columnas como para las filas. El orden de la relación es tomada filas-columnas.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	1	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
16	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
20	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
21	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1