Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

División de Ciencias Básicas e Ingeniería

"Método-Asistente para la toma de decisiones de diseño de arquitecturas de software (MATDDS)"

Tesis que presenta:

Lic. Sandra Méndez Luna

Para obtener el grado de

Maestra en Ciencias

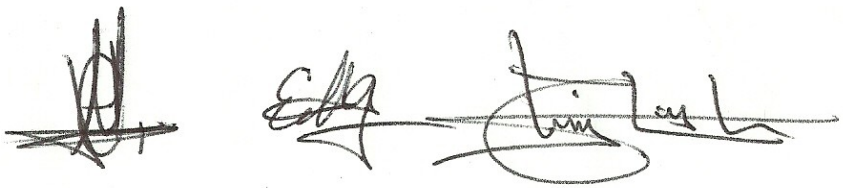
(Ciencias y Tecnologías de la Información)

Asesor: Dr. Humberto Cervantes Maceda

Jurado Calificador:

Presidente:	Ing. Luis Fernando Castro Careaga
Secretario:	Dr. Humberto Cervantes Maceda
Vocal:	M. èn C. Edith Valencia Martínez

19 de Marzo del 2012



Agradecimientos

Agradezco a todos aquellos que estuvieron presentes a lo largo de la realización de este sueño.

A el Dr. Humberto Cervantes Maceda por haber aceptado ser mi asesor de tesis y por el tiempo que invirtió en orientarme y dirigirme para culminar este proyecto.

Agradezco a la M. En C. Edith Valencia Martínez y al Ing. Luis Fernando Castro Careaga por haber aceptado formar parte del jurado revisor y calificador para la defensa de mi tesis, por sus comentarios y sugerencias. También quiero extender mi agradecimiento a la Dra. Perla Ines Velasco Elizondo por sus observaciones y sugerencias acerca de mi tesis.

A la Universidad Autónoma Metropolitana por haber permitido mi desarrollo académico a nivel Licenciatura y Posgrado. También a los profesores de la maestría que contribuyeron a mi desarrollo académico.

Agradezco a también a los amigos y compañero que estuvieron conmigo, especialmente a Karina Marcela Contreras Hernández, María Esther Sosa Rodríguez, Erick Andrey Serratos Álvarez, Israel Hernández Merchand, Pablo Hernández Durán e Israel de Olmos Ramírez quienes siempre me apoyaron en todos los aspectos.

Agradezco a mi familia que me apoyo siempre de todas las maneras posibles, especialmente mi tío Cuauhtémoc Méndez Salvador por todos sus consejos y enseñanzas, a mi tío Joaquín Méndez Salvador por su apoyo el último año, a mi madre María Luisa Luna Trejo por toda su compresión y amor, y a mi padre Arturo Méndez Salvador quien sé que seguirá conmigo donde quiera que esté.

Por último agradezco a CONACyT por el apoyo económico que me otorgó en mi estadía en esta maestría.

Resumen

La propuesta que se presenta en este trabajo de tesis describe un enfoque al problema de la selección de patrones durante el diseño de la arquitectura de software. Se propone la utilización de un programa de software (Asistente-MATDDS), basado en un método obtenido como parte de este trabajo, el cual utiliza un algoritmo de búsqueda para asistir al arquitecto en la toma de decisiones de diseño respecto a la selección de patrones. La información que usa el algoritmo es sobre los atributos de calidad que se deben satisfacer al realizar el diseño de la arquitectura, ofreciendo una lista de patrones que el arquitecto puede aplicar. Como resultado de este estudio se presenta también una evaluación preliminar del trabajo realizado, ésto a través de dos ejemplo desarrollados con la ayuda de esta herramienta.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

La industria del desarrollo de software ha ido creciendo con el transcurso del tiempo y de la misma manera la dificultad de crear sistemas de software, ya que cada vez éstos se vuelven más complejos. Para poder realizar el desarrollo de software, las empresas especializadas aplican procesos y métodos, que les ayudan a desarrollar sistemas en tiempo, costo y calidad [6]. Parte de la complejidad mencionada se ve reflejada en las arquitecturas que se diseñan para estos sistemas.

1.1 Contexto

El contexto de este proyecto es el desarrollo de sistemas de software y las metodologías que se usan para hacerlo. El desarrollo de sistemas de software se realiza a través de varias actividades que están relacionadas con cuatro fases:

- *Fase de Requerimientos*. En esta primera fase se obtienen los datos necesarios para poder desarrollar el sistema [6] (captura, análisis y especificación de requerimientos). Los requerimientos son condiciones o capacidades a las cuales se debe ajustar el sistema.
- *Fase de Diseño de Alto Nivel*. Aquí se lleva a cabo el diseño de la arquitectura del sistema. La arquitectura de software es la estructura o estructuras del sistema que contienen elementos o componentes de software, los cuales tiene propiedades y relaciones entre si [10]
- *Fase de Construcción*. En esta fase se diseñan, codifican y prueban todos aquellos elementos obtenidos en la fase anterior, teniendo como base la arquitectura obtenida.
- *Fase de Integración y Pruebas*. Se validan los elementos construidos y se integran entre si para formar el software que deberá estar basado en los requerimientos obtenidos, diseñados e implementados en las fases anteriores.

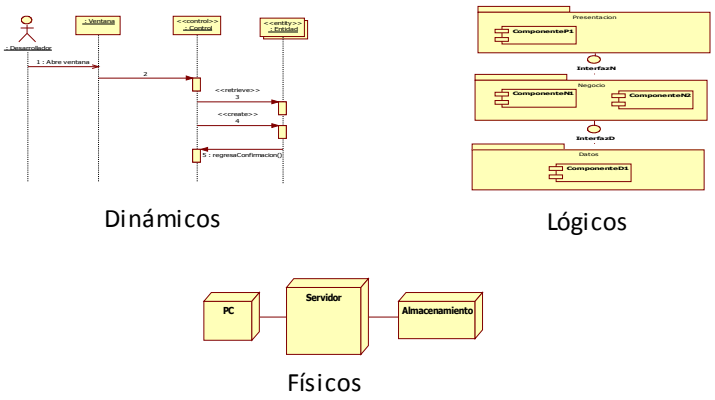


Figura 1: Elementos y sus relaciones que forman las estructuras representados por diagramas

Idealmente, al final del desarrollo, se obtiene un software probado y construido que satisface las necesidades de los clientes y del grupo de desarrollo.

Dentro de las primeras fases del desarrollo del sistema se realiza el desarrollo de la arquitectura. La arquitectura proporciona una visión esencial del sistema, ya que identifica sus partes o elementos y cómo se relacionan entre si para dar un servicio o cubrir una necesidad. Los elementos que pueden estar representados por las estructuras son de distintos tipos, por ejemplo *Elementos Dinámicos* (objetos, hilos), *Elementos Lógicos* (clases componentes), *Elementos Físicos* (nodos, directorios). En la figura 1 se muestran diagramas en notación UML (Unified Modeling Language) que representan estructuras conformadas por los elementos mencionados.

Este conjunto de elementos que conforman a la arquitectura es el que ayuda a satisfacer los requerimientos y atributos de calidad que debe cumplir un sistema (que sea seguro, modificable, etc). También la arquitectura ayuda a cumplir con las restricciones que se le imponen al sistema. A este conjunto de requerimientos, atributos de calidad y restricciones se les llama drivers arquitectónicos.

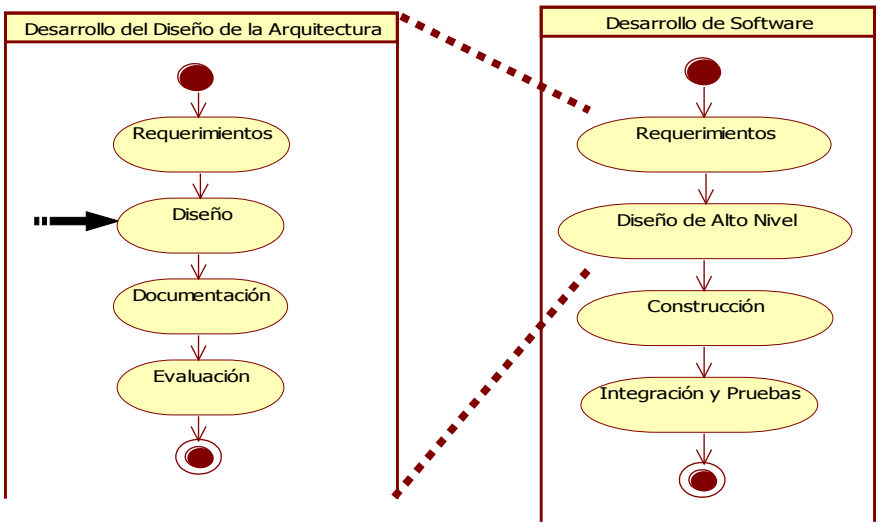


Figura 2: El Ciclo de Desarrollo de la Arquitectura ocurre dentro de fases tempranas del Desarrollo de Software

De forma conceptual, el desarrollo de la arquitectura se lleva a cabo en cuatro etapas o fases, que en conjunto forman el ciclo de vida del desarrollo de la arquitectura. Este ciclo de vida ocurre como parte de las dos primeras fases del Desarrollo de Software, como se muestra en la figura 2.

Las fases del ciclo de vida del desarrollo de la arquitectura son:

- Requerimientos – Se refiere a la fase de recolección, análisis y asignación de prioridades de requerimientos que influyen en la arquitectura o también llamados drivers arquitectónicos.
- Diseño – Esta fase se refiere a la de diseño de la arquitectura del sistema, a través de la toma de decisiones de diseño.
- Documentación – En esta fase se hace la documentación del diseño de la arquitectura a través de vistas, las cuales plasman la estructuras obtenidas del diseño.
- Evaluación – En esta fase se evalúa la arquitectura diseñada y documentada en las fases anteriores, con la finalidad de saber si el diseño satisface los drivers arquitectónicos.

En general, se tienen dos niveles de detalle para el diseño: el diseño de alto nivel y el diseño detallado. En el diseño de alto nivel se identifican los componentes que va a tener el sistema. En el diseño detallado se describe cada componente internamente, de tal forma que se pueda hacer su construcción de forma individual. El diseño detallado se realiza en la *fase de Construcción del Desarrollo de Software*. En la figura 3 se ejemplifica el diseño de alto nivel con la construcción de dos componente, y el diseño detallado con la descripción interna de uno de ellos.

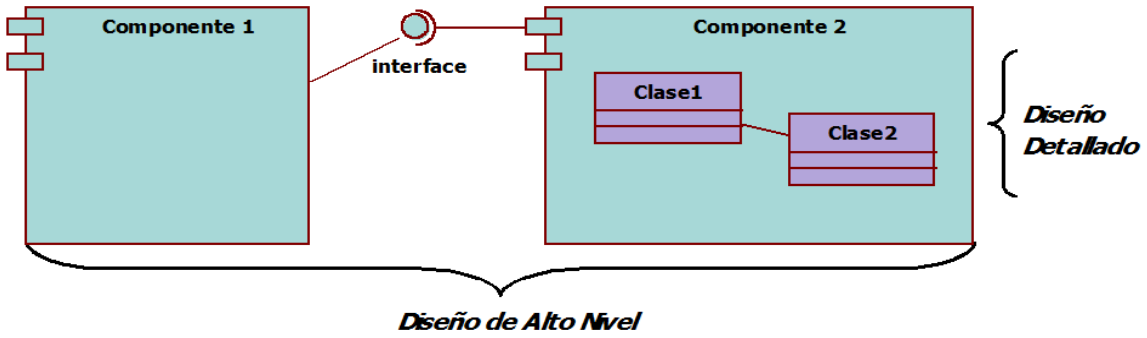


Figura 3: Diseño de Alto Nivel y Diseño Detallado

En la fase de diseño del ciclo de vida del desarrollo de la arquitectura se hace la descomposición del sistema en elementos o componentes y se analizan sus comportamientos y las relaciones entre ellos (estructuras). Generalmente la creación de estas estructuras involucra soluciones conceptuales a problemas que son recurrentes en el diseño llamadas patrones [18], así como tácticas de diseño las cuales buscan controlar la respuesta de un atributo de calidad en particular [10]. Los patrones y las tácticas son decisiones de diseño que sirven de guía para satisfacer los drivers arquitectónicos.

El diseño de la arquitectura se hace de manera iterativa, para obtener un mayor refinamiento del diseño en cada iteración. Existen varios métodos o procesos que sirven de guía en el diseño, como son ADD (Attribute-Driven Design) [8], ACDM (The Architecture Centric Development Method) [14] o RUP (The Rational Unified Process) [11]. Todos estos conceptos se detallarán más adelante.

1.2 Problemática

Una fase importante en el ciclo de vida del desarrollo de la arquitectura es la de diseño y aunque para realizar esta tarea existen métodos que permiten realizar las actividades de forma sistemática, éstos no guían en el cómo hacer la toma de decisión para elegir los patrones y tácticas de diseño que satisfagan a los drivers arquitectónicos.

Las decisiones de diseño que guían la arquitectura son tomadas por el Arquitecto y se hacen con base en su experiencia adquirida a través de los años, en torno a los patrones y las tácticas de diseño. Si no se tiene la experiencia necesaria, en particular con respecto a los patrones, esta toma de decisión es muy complicada debido a que existe una gran cantidad de ellos que influyen de alguna manera sobre los drivers arquitectónicos. La influencia de un patrón sobre un driver puede ser positiva, negativa o nula, esto es, que satisfaga, perjudique o no afecten a algún driver arquitectónico en particular. Esta influencia es la que determina si se aplica un patrón o no. A este problema se le suma que existen pocos arquitectos para hacer desarrollo de software y aún menos con la experiencia necesaria para hacer buenas tomas de decisiones con respecto a los patrones.

La consecuencia de hacer una mala toma de decisiones de diseño es no obtener una arquitectura que satisfaga los drivers arquitectónicos y por lo tanto no obtener un sistema que cumpla con las necesidades requeridas por clientes y desarrolladores.

1.3 Propuesta

Este proyecto busca proponer una solución a la problemática anterior, por medio del uso de un Método (MATDDS – Método-Asistente para la Toma de Decisiones de Diseño de arquitecturas de Software) que ayude a aquel arquitecto sin experiencia en el uso, específicamente, de patrones. El Método (MATDDS) busca ayudar al arquitecto proporcionando sugerencias sobre posibles decisiones de diseño, por medio de una serie de pasos a seguir. Para ello, el Método (MATDDS) utiliza información de catálogos de patrones y un algoritmo que permite identificar un conjunto de patrones que ayuden a satisfacer los drivers arquitectónicos del sistema, específicamente la satisfacción de atributos de calidad involucrados en el desarrollo. Los detalles de la propuesta se muestran en el *Capítulo III* de ésta tesis.

1.4 Objetivos

Esta tesis tiene un objetivo general del cual se desprenden algunos objetivos específicos.

1.4.1. Objetivo General

- Crear un método que ayude a arquitectos a hacer toma de decisiones de diseño en la fase de Diseño en el Ciclo de Desarrollo de la Arquitectura de Software con respecto a los patrones (MATDDS).

1.4.2. Objetivos Específicos

- Identificar la influencia que tiene cada patrón sobre varias categorías de atributo de calidad.
- Encontrar una estrategia para poder determinar el conjunto de patrones que satisfagan los atributos de calidad de un Sistema de Software dado.
- Desarrollar dicha estrategia para obtener un método que pueda ser aplicado.
- Crear una herramienta (Asistente) que facilite la aplicación del método obtenido y aplicarla a un ejemplo, de tal forma que se pueda hacer un análisis de resultados.
- Evaluar los resultados obtenidos.

1.5 Metodología

El proyecto que dio como resultado la escritura de esta tesis se realizó en varias etapas, bajo ciertas condiciones dando como resultado algunos alcances, todos estos explicados a continuación:

1.5.1. Etapas

Las etapas para el desarrollo de este proyecto fueron las siguientes:

- Investigación del Marco Teórico y del Estado del Arte – En esta etapa se llevó a cabo la revisión e investigación de los elementos básicos para entender la realización del diseño de la arquitectura, así como los métodos utilizados para hacer el diseño y los trabajos relacionados con el tema.
- Identificación del Problema – Se hizo un análisis tanto del marco teórico como del estado del arte, para determinar nichos de oportunidad.
- Desarrollo del Algoritmo – Se consideraron varias opciones de algoritmos que ayudaran a resolver el problema identificado, de tal forma que el resultado fuera aceptable.
- Desarrollo del Método y Asistente MATDDS – Analizando los elementos obtenidos, se describió un método para auxiliar en la toma de decisiones de diseño, posteriormente creando un Asistente con base en el método obtenido.
- Evaluación y Análisis de Resultados – Con el Método MATDDS desarrollado se realizaron pruebas sobre dos ejemplos, para establecer los alcances del método.

1.5.2. Condiciones y Alcances del Proyecto

Este proyecto se llevó a cabo bajo ciertas condiciones para tener ciertos alcances, los cuales se mencionan a continuación:

Condiciones

- Este proyecto sólo se enfocó en los atributos de calidad del conjunto de drivers arquitectónicos. Se toma en cuenta sólo éstos ya que son los considerados para cuantificar la calidad del sistema, aspecto que es muy importante tanto para el grupo de desarrollo (arquitecto y desarrolladores) como para el cliente.
- Sólo se tomaron seis categorías de atributos de calidad, los sugeridos por el SEI [8] (Software Engineering Institute), Modificabilidad, Disponibilidad, Desempeño, Usabilidad, Seguridad y Facilidad de Pruebas.
- De las decisiones de diseño, el enfoque fue sólo sobre los patrones, en particular aquellos descritos en

el catálogo escrito por Bushmann [17]. La razón de ésto fue porque los patrones están relacionados con los atributos de calidad, es decir, por medio de los patrones se pueden satisfacer éstos.

Alcances

- Se obtuvo un método que sirve para tomar de decisiones, al realizar el diseño de la arquitectura.
- Se logró utilizar el Método MATDDS como parte de un método para realizar diseño de arquitecturas.

1.6 Estructura de la Tesis

En el *Capitulo II* se definen detalladamente conceptos esenciales para este proyecto, se presenta también los trabajos previos, tanto de métodos de diseño como de herramientas que ayudan a llevar a cabo la toma de decisiones y por último se muestra la comparación de los métodos y herramientas expuestas. En el *Capitulo III* se retoma la problemática para explicarla con mayor detalle, así como la propuesta para solucionar el problema y se da una explicación de cómo se llegó a tal propuesta. En el Capítulo IV se exponen los resultados obtenidos al aplicar la solución propuesta en dos ejemplos de diseño de arquitectura de software y se hace un análisis de los resultados obtenidos. Finalmente se exponen las conclusiones del trabajo de tesis, basadas en los resultados obtenidos y se hace una propuesta del trabajo futuro que se puede realizar con base en este proyecto de tesis.

Capítulo 2

Marco Teórico y Estado del Arte

Este capítulo define con mayor detalle algunos de los conceptos que ya se mencionaron anteriormente, así como el marco teórico para este proyecto. El marco teórico incluye una breve descripción de algunos métodos para llevar a cabo el diseño de la arquitectura, así como una comparación entre ellos. También incluye el trabajo relacionado respecto a herramientas auxiliares en la toma de decisiones en el diseño de la arquitectura.

Entrando en el contexto del proyecto, referente al desarrollo de la arquitectura, a continuación se definirán algunos conceptos necesarios para la total comprensión tanto de la problemática como de la propuesta.

2.1 Ciclo de Vida del Desarrollo de la Arquitectura

Como ya se mencionó, la arquitectura se lleva a cabo a través de cuatro fases, que en conjunto se denominan Ciclo de Vida del Desarrollo de la Arquitectura. Estas fase se describen a continuación.

2.1.1. Fase de Requerimientos

En esta fase se hace la de recolección y análisis de requerimientos que influyen en la arquitectura o drivers arquitectónicos; es el punto de partida para hacer el diseño de la arquitectura. En esta fase se le asigna una prioridad a los drivers arquitectónicos. La forma en que se hace esta asignación de prioridad es por medio del análisis de escenarios. Un escenario es una situación real que representa y define a un driver, pueden existir varios escenarios para un driver particular. Existen algunas técnicas o métodos que ayudan a llevar a cabo la recolección y análisis de los requerimientos, como son el taller QAW desarrollado por el SEI [8] o el modelo de

clasificación *FURPS+* [3], desarrollado por Hewlett-Packard, los cuales se explican con un poco de detalle en el *Apéndice A*.

2.1.2. Fase de Diseño

Esta fase se enfoca al diseño de la arquitectura del sistema, a través de la toma de decisiones de diseño. En donde se ven involucrados los resultados obtenidos del análisis de requerimientos de la fase anterior. De esta fase se hablará con detalle más adelante.

2.1.3. Fase de Documentación

Una vez que se tiene el diseño de la arquitectura es necesario hacer su documentación para plasmar el por qué se tomaron las decisiones de diseño y de esta forma tener un mayor entendimiento de la arquitectura. La documentación es la forma de comunicar estas decisiones de diseño, no solo a los clientes sino también a los desarrolladores, siendo estos últimos los encargados de hacer la construcción de la arquitectura. La documentación se hace a través de vistas, en las cuales se muestra un conjunto de elementos arquitectónicos y del resto del sistema. Cada una de las vistas buscan alcanzar distintos objetivos y tienen distintos usos, esto es, presentan distintos tipos de información.

Existen distintos tipos de vistas como son:

- Vista Lógica – Muestra la estructura estática del sistema.
- Vista de Desarrollo – Muestra el sistema desde el punto de vista de los programadores, esto es, la organización estática del software en su entorno de desarrollo [7].
- Vista Física – Muestra las asignaciones del software en el hardware y refleja el aspecto distribuido del sistema [7].
- Vista de Procesos o Dinámica – Muestra hilos de control, procesos de ejecución y la comunicación entre ellos.

La documentación no solo sirve para tener una buena comunicación entre el arquitecto y los involucrados (clientes y desarrolladores), también es útil para dar mantenimiento, hacer modificaciones o permitir la evolución del sistema de software. Para documentar la arquitectura se utilizan distintos métodos auxiliares, como son "Views and Beyond" desarrollado por el SEI [8] o "The 4+1 Views" desarrollado por Philippe Kruchten [7], así como las plantillas ofrecidas por RUP [11] y OpenUP [12].

2.1.4. Fase de Evaluación

Una vez que se tiene la documentación de la arquitectura, es necesario evaluar y verificar que el diseño cubra las necesidades del cliente, esto es, que satisfaga los drivers arquitectónicos. Esta fase permite identificar

defectos de manera temprana y mitigar riegos, ya que aún no se ha hecho la implementación de la arquitectura.

El proceso de evaluación se lleva a cabo analizando la documentación realizada en la fase anterior. Para lograr este objetivo se usan métodos tales como *ATAM* (*Architecture Tradeoff Analysis Method), ARID (Active Reviews for Intermediate Designs)* [15]*, CBAM (Cost Benefit Analysis Method*) los tres desarrollados por SEI [8] o el método ACDM (The Architecture Centric Development Method) que cuenta con una sección de revisión de la arquitectura [14].

El ciclo de desarrollo de la arquitectura se completa al evaluar el diseño y concluir que se satisfacen los drivers arquitectónicos, por lo tanto están satisfechas las necesidades de la o las personas que requieren y desarrollan el sistema de software.

2.2 Drivers Arquitectónicos

Los drivers arquitectónicos son un conjunto de requerimientos de software relevantes para la creación del diseño de la arquitectura. Dentro de estos requerimientos de software están los requerimientos funcionales primarios, restricciones y atributos de calidad, todos estos derivados de las necesidades de los involucrados.

La figura 4 describe el orden que se sigue desde que se obtienen los *objetivos de negocio* que son las necesidades de los involucrados, de los cuales se van a derivar los requerimientos arquitecturalmente importantes, que serán los *drivers arquitectónicos*. Los drivers servirán de entrada para hacer el *diseño de la arquitectura*, y finalmente obtener la *arquitectura del sistema de software*. Todos estos conceptos se describirán a continuación.

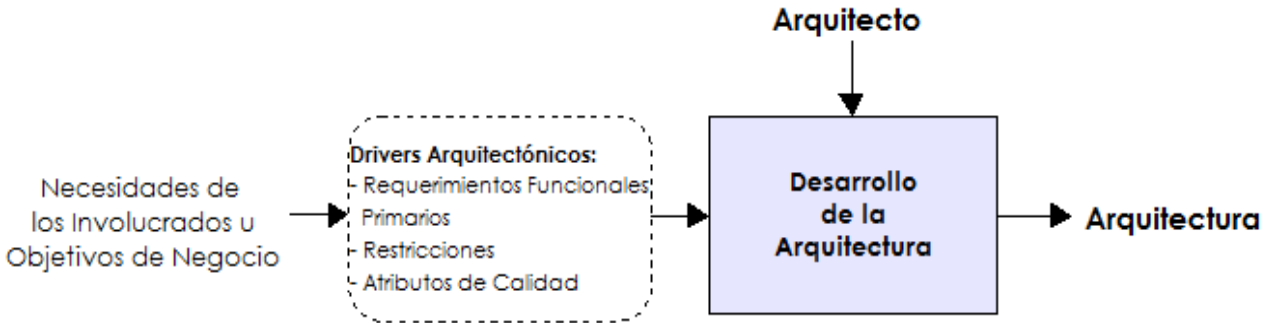


Figura 4: Drivers Arquitectónicos como entradas para el Desarrollo de la Arquitectura

2.2.1. Objetivos de negocio

Los objetivos de negocio son la justificación del porque es analizado y construido un sistema de software, esto es, la necesidad de los involucrados de satisfacer actividades relacionadas con su negocio [4]. Existe una relación entre los objetivos de negocio y la arquitectura de software, se dice que la arquitectura es el puente entre los objetivos de negocio y el sistema, es decir, el sistema debe cumplir con las necesidades de los involucrados (ya sea el cliente o el grupo de desarrollo) a través de la arquitectura del sistema de software. Entender los objetivos es fundamental para el desarrollo del sistema ya que es de éstos de donde se obtienen los requerimientos de software.

2.2.2. Requerimientos de Software

Un requerimiento de software, como se mencionó anteriormente, es una condición o capacidad a la cual el sistema se debe de ajustar. Se derivan directamente de las necesidades del usuario o bien se indica en algún contrato, estándar, especificación o algún otro documento formal [11]. Los requerimientos son colectados en la fase más temprana del desarrollo de software.

Existen dos categorías de requerimientos, los llamados requerimientos funcionales y los requerimientos no-funcionales. Los requerimientos funcionales son los que definen el comportamiento básico del sistema que soportan las metas, tareas y actividades que requiere el usuario [2].

Los requerimientos no-funcionales no describen un comportamiento del sistema, sino son propiedades emergentes o limitaciones. Éstos pueden ser *Atributos de Calidad* o *Restricciones* respectivamente. Los llamados Atributos de Calidad se relacionan con la calidad que debe tener el sistema mientras que las Restricciones son limitantes impuestas sobre el sistema o restricciones operativas. De cada uno de los requerimiento mencionados anteriormente se hablará a continuación.

2.2.2.1 Requerimientos Funcionales Primarios

Dentro de los requerimientos funcionales existe un sub-conjunto que se considera como arquitecturalmente significativo o que influye de alguna forma en la arquitectura, a este grupo se le llama *Requerimientos Funcionales Primarios*. Estos requerimientos son seleccionados por ser esenciales para el funcionamiento del sistema, por su importancia para el negocio y porque afectan de alguna forma la arquitectura del sistema que se va a desarrollar (interactúan con mucho elementos arquitectónicos), dejando de lado todos aquellos que no la afectan.

2.2.2.2 Atributos de Calidad

Los atributos de calidad son características o propiedades medibles del sistema, que afectan el grado de satisfacción de los usuarios y los desarrolladores con respecto al sistema. Están relacionados con la calidad que

se requiere del sistema. Para este proyecto se tomarán en cuenta las categorías de atributos de calidad sugeridos por el SEI [10], como se mencionó anteriormente:

- Modificabilidad (modifiability) – El costo de realizar un cambio, tomando en cuenta qué se va a cambiar y cuando y quien va a realizar el cambio.
- Disponibilidad (availability) – Tiene que ver con las fallas del sistema y las consecuencias asociadas a estas fallas.
- Desempeño (performance) – Se refiere al tiempo de respuesta del sistema cuando ocurre un evento como interrupciones, mensajes, solicitudes del usuario, etc.
- Usabilidad (usability) – Concierne a la facilidad de usar el sistema, por ejemplo la facilidad de aprender las características, aprender a usar el sistema de una forma eficiente, minimizar el impacto de errores en su manejo, qué tan fácil se adapta el sistema a las necesidades del usuario y la satisfacción que tiene el usuario para con el sistema.
- Seguridad (security) – Es la habilidad que tiene el sistema de resistir usos no autorizados o ataques, sin dejar de proveer sus servicios a usuarios legítimos.
- Facilidad de Pruebas (testability) – Se refiere a la facilidad de mostrar las fallas del sistema a través de pruebas y de esta forma poder solucionarlas.

2.2.2.3 Restricciones

Las restricciones son características que limitan el diseño y el desarrollo de la arquitectura de alguna forma. Estas restricciones son impuestas por la organización que requiere el sistema o por la organización que lo desarrolla. Pueden ser de distintos tipos como son las restricciones a nivel de implementación, sistema operativo o manejador de la base de datos que se va a utilizar, llamadas restricciones técnicas. También pueden ser restricciones de negocio, como el tener una fecha límite de entrega.

Como ya se mencionó, el conjunto de requerimientos tanto funcionales como no-funcionales conforman los drivers arquitectónicos que ayudan a realizar el diseño de la arquitectura.

2.3 Diseño de la Arquitectura

Como se mencionó anteriormente, la finalidad de la arquitectura es satisfacer los drivers arquitectónicos y de esta forma obtener estructuras que guíen el desarrollo de software. La forma de lograr esta satisfacción es dividiendo el sistema en elementos y repartiendo o asignando las necesidades o drivers arquitectónicos en los distintos elementos del sistema. Para no invertir más tiempo en la búsqueda de soluciones, se aplican patrones y tácticas

de diseño, que son soluciones ya probadas. Los patrones y tácticas de diseño darán como resultado las estructuras que conforman la arquitectura de software.

2.3.1. Patrones

Un patrón es una solución conceptual a un problema recurrente en el contexto del diseño de la arquitectura de software. Los patrones definen elementos o estructuras fundamentales para el sistema, los tipos de relaciones que existen entre ellos y un conjunto de restricciones de cómo pueden ser usadas estas estructuras [10]. También proveen un vocabulario común con el cual los arquitectos y desarrolladores se pueden comunicar de manera sencilla.

Existen diferentes niveles de granularidad de patrones, como son los patrones de diseño o los patrones arquitectónicos. Los patrones de diseño tienen un nivel de detalle fino, generalmente es el diseño dentro de los componentes (grano fino). Los patrones arquitectónicos se dice que son de alto nivel ya que el nivel de detalle es de grano grueso. Debido a que existen una gran cantidad de patrones tanto de diseño como arquitectónicos, algunos expertos en el tema han elaborado catálogos que los describen al igual que las relaciones entre ellos y la forma en que se pueden usar. Uno de estos catálogos es el llamado Pattern-Oriented Software Architecture (POSA) de Frank Buschmann, Kevlin Henney y Douglas C. Schmidt [17]. Este catálogo muestra los patrones en mapas, donde cada patrón es un nodo y está relacionado con uno o más patrones a través de aristas, las cuales representan las necesidades que tiene el usuario. De esta forma se plasman en este catálogo alrededor de 116 patrones. Al unir cada uno de estos mapas se puede obtener un grafo de patrones relacionados a través de necesidades de usuarios como se muestra en la figura 5.

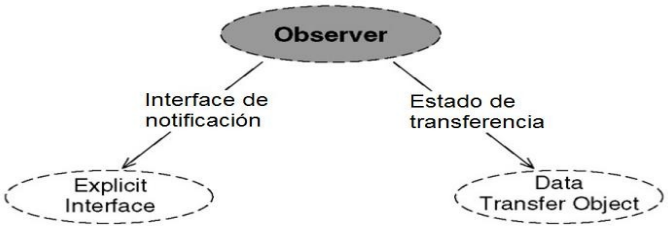


Figura 5: Forma en que se representan los patrones en el catálogo POSA

El beneficio de tener catálogos de patrones, es poder usar los patrones plasmados en estos con la confianza de que han sido hechos por personas expertas. Aún así es necesario saber usar los patrones correctamente, para ésto se requiere nuevamente experiencia, en este caso la experiencia de los arquitectos. En muchas ocasiones, esta experiencia no es suficiente para garantizar el éxito del proyecto, ya que no es tan fácil determinar el patrón a aplicar para satisfacer los drivers arquitectónicos.

2.3.2. Tácticas de Diseño

Al igual que los patrones, las tácticas son decisiones de diseño. Éstas buscan controlar la respuesta de un atributo de calidad [10]. Estas tácticas se encuentran plasmadas en el libro Software Architecture in Practice [10] de Len Bass, Paul Clements y Rick Kazman, donde al igual que los catálogos de patrones, se da una descripción de ellas. Este catálogo especifica la forma en que una táctica ayuda a satisfacer un atributo de calidad específico, ésto se debe a que las tácticas están asociadas a un atributo de calidad a diferencia de los patrones que pueden satisfacer uno o varios de ellos. A continuación se listan algunas Tácticas de Diseño correspondiente a los atributos de calidad sugeridos por el SEI:

- Tácticas de Disponibilidad – Ping/echo, Latido (heartbeat), excepciones, voto, redundancia activa, etc. En la figura 6 se muestran las tácticas que se utilizan para obtener disponibilidad en un sistema.
- Tácticas de Modificabilidad – Mantener coherencia semántica, anticipar cambios esperados, generalizar el módulo con el que se trabaja, encapsulamiento, polimorfismo, etc.
- Tácticas de Desempeño – Aumento de eficiencia computacional, administrar cantidades de eventos, control de frecuencia de muestreo, limitar tiempos de ejecución, introducir concurrencia, etc.
- Tácticas de Seguridad – Autentificación de usuarios, mantenimiento de confidencialidad de datos, mantenimiento de integridad, detección de intrusos, restauración de estado, etc.
- Tácticas de Facilidad de Pruebas – Separación de interfaz e implementación, interfaces de pruebas especializadas, monitores integrados, etc.
- Tácticas de Usabilidad – Estas tácticas se dividen en las tácticas en tiempo de ejecución y tácticas en tiempo de diseño.

Las tácticas y los patrones están relacionados, un patrón puede englobar a varias tácticas y una vez instanciado un patrón puede ser modificado mediante el uso de tácticas.

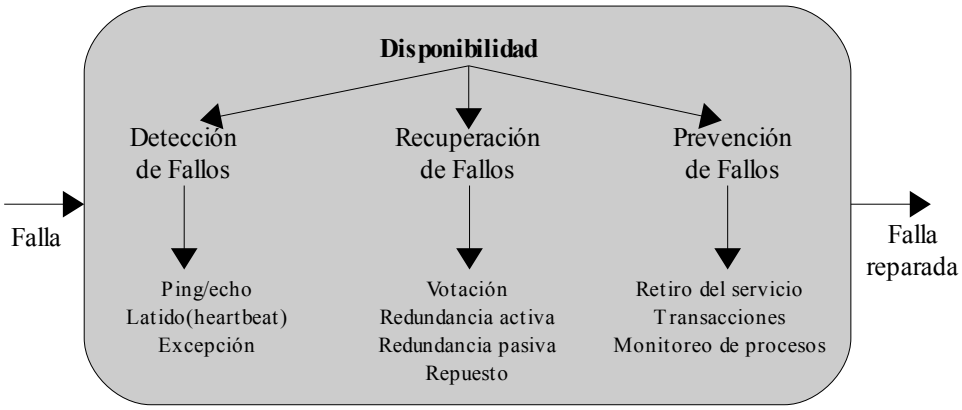


Figura 6: Tácticas de Diseño que atacan el atributos de calidad de Disponibilidad

Como ya se mencionó, estas decisiones de diseño se llevan a cabo en la fase de Diseño del Ciclo de Vida del Desarrollo de la Arquitectura, fase para la cual existen algunos métodos de ayuda, los cuales se describirán a continuación.

2.4 Métodos de Diseño

Existen distintos métodos para llevar a cabo el diseño de la arquitectura de un sistema de software, como son *ADD* (*Attribute-Driven Design*) [8], enfocado exclusivamente al diseño de la arquitectura; *ACDM* (*The Architecture Centric Development Method*) [14] y *RUP* (*The Rational Unified Process*) [11], que no se enfocan específicamente al diseño pero incorporan tareas que lo guían.

2.4.1. ADD

ADD se basa en drivers arquitectónicos, especialmente en los atributos de calidad, para hacer el diseño de la arquitectura. Este método hace una descomposición del sistema en elementos más pequeños, a través de varias iteraciones. En la primera iteración se toma el sistema como un solo elemento. Los componentes o elementos resultantes de la descomposición se volverán a descomponer cada uno de ellos, de ser necesario. En cada iteración del método se van aplicando tácticas y patrones para satisfacer los drivers.

ADD sigue un enfoque llamado "Plan, Do, Check" [16].

- Plan – Los atributos de calidad y las restricciones son consideradas para seleccionar qué elementos se usarán en la arquitectura.
- Do – Los elementos son instanciados para satisfacer los atributos de calidad así como los requerimientos funcionales.
- Check – El diseño resultante es analizado para determinar si lo requerimientos son alcanzados.

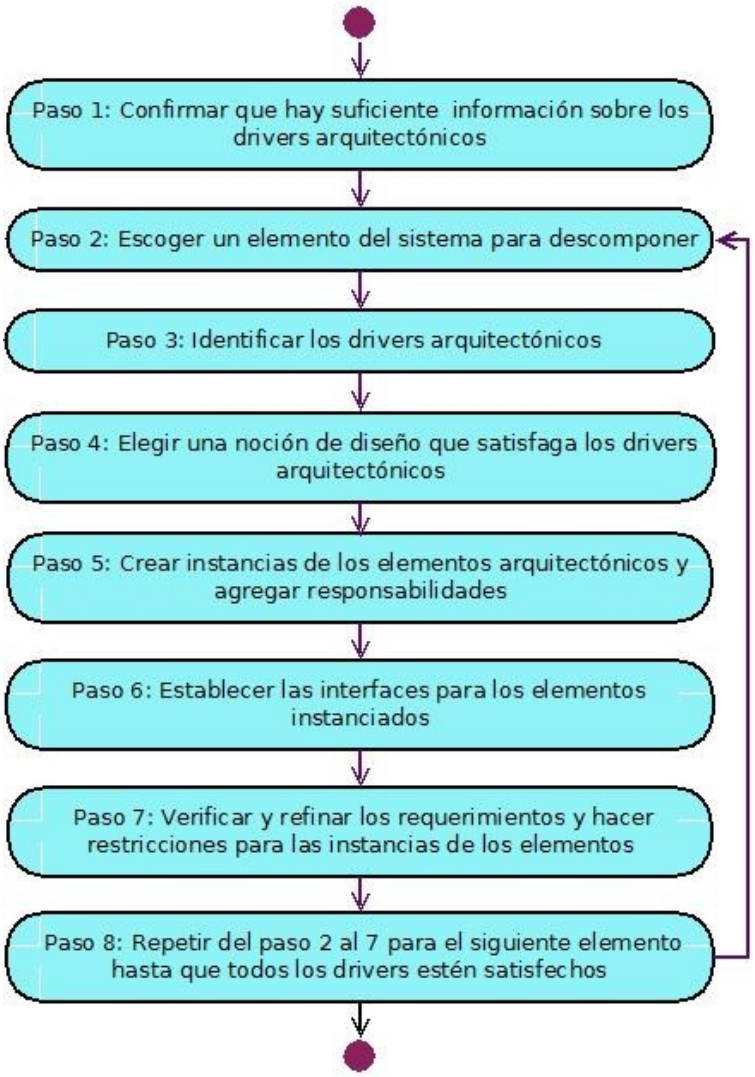


Figura 7: Pasos de ADD

ADD consta de ocho pasos a través de los cuales se va haciendo el diseño de la arquitectura en cada iteración, como se muestra en la figura 7.

Paso 1: **Confirmar que hay suficiente información sobre los requerimientos**

En este paso se confirma que se tiene suficiente información sobre los drivers arquitectónicos y se verifica que éstos estén priorizados de acuerdo a las necesidades de los involucrados y los objetivos de negocio, de una manera decreciente con respecto a la importancia, o sea, del de mayor al de menor importancia. Toda esta información ayuda al arquitecto a tomar las decisiones de diseño en los pasos siguientes, como son la selección de patrones y tácticas de diseño para alcanzar los requerimientos del sistema [13]. Durante este paso no se toman decisiones de diseño.

Paso 2: **Escoger un elemento del sistema para descomponer**

Se selecciona el elemento con el cual se trabajará en los siguientes pasos del método, esta selección se puede hacer de dos formas.

A) Primer Caso: Si es la primera iteración del método, el único elemento que se tiene para descomponer es el sistema en si y todos los requerimientos se le asignan al sistema.

B) Segundo Caso: Ya se ha llegado anteriormente al paso dos, esto indica que el sistema se ha dividido en dos o más elementos a los cuales se les han asignado requerimientos. De estos elementos se debe seleccionar uno para que se trabaje con él en los siguientes pasos. La forma de hacer esta selección es basada en los siguientes aspectos.

- Conocimiento de la arquitectura actual: Si es el único elemento que se puede seleccionar, como en el caso de que sea el último elemento que se quiera descomponer o el número de dependencias que tiene con los demás elementos del sistema.
- Riesgo y dificultad: Qué tan difícil será alcanzar los requerimientos asociados al elemento o cómo se van a alcanzar los requerimientos asociados al él o los riesgos relacionados con alcanzar los requerimientos relacionados con el elemento.
- Criterio de negocio: El papel que tiene el elemento en el desarrollo incremental del sistema y en las liberaciones de funcionalidad o si el elemento se va a construir o se va a comprar o el impacto que tendrá en el mercado, etc.
- Criterio de la organización: El impacto que tendrá el elemento en los recursos ya utilizados, el nivel de habilidad de los desarrolladores o que alguna persona con autoridad seleccione el elemento.

Durante este paso no se toman decisiones de diseño.

Paso 3: **Identificar los drivers arquitectónicos candidatos**

En este paso ya se tiene seleccionado un elemento con requerimientos priorizados asociados a él. A estos requerimientos se les asigna una nueva prioridad, que se representa con parejas de letras, las cuales indican el nivel de importancia, ya sea alto (H), medio (M) o bajo (L), donde la primera letra representa la importancia para los involucrados y la segunda el impacto sobre la arquitectura.

(H, H), (H, M), (H, L), (M, H), (M, M), (M, L), (L, H), (L, M), (L, L)

De esta prioridad se seleccionan los requerimientos que tenga nivel más alto en las parejas. Pueden ser de uno a cinco requerimientos, los cuales serán candidatos a drivers arquitectónicos para ese elemento. Durante este paso no se toman decisiones de diseño.

Paso 4: **Elegir una noción de diseño que satisfaga los drivers arquitectónicos**

Se entiende por noción de diseño un patrón o una táctica de diseño. Se deben seleccionar los principales elementos que aparecen en la arquitectura y que tienen que ver con el elemento que se va a descomponer, también se determina el tipo de relaciones entre ellos. Este paso se lleva a cabo a través de los siguientes puntos:

1. Identificar los problemas de diseño asociados a los drivers arquitectónicos candidatos.
2. Para cada problema de diseño se crea una lista de patrones que se ocupen de ellos. Los patrones se seleccionan tomando en cuenta el conocimiento, habilidades y experiencia que se tiene con respecto a ellos. En caso de que un driver arquitectónico se refiera a más de un atributo de calidad, se aplicará una o múltiples tácticas. Para cada patrón se identifican parámetros y valores.
3. Seleccionar de la lista de patrones los que sean más apropiados para satisfacer los drivers arquitectónicos candidatos, creando una matriz para evaluar ventajas y desventajas, teniendo en cuenta que tan bien se pueden combinar los patrones seleccionados.
4. Considerar los patrones identificados y decidir la relación entre ellos, la combinación de los patrones puede dar como resultado un nuevo patrón.
5. Describir los patrones que se seleccionaron por medio de vistas arquitectónicas, sin necesidad de tener mucho detalle como cuando se esta documentando la arquitectura.
6. Evaluar y resolver inconsistencias en el diseño conceptual, esto se hace tomando en cuenta los drivers arquitectónicos para ver si todos se consideraron. Si no se satisfacen los drivers, se analizan los patrones y tácticas alternativos para aplicarse de ser necesario. Evaluar el actual elemento que se diseñó contra los demás elementos de la arquitectura y resolver inconsistencias.

En este paso sí se toman decisiones de diseño, como son las funcionalidades que tienen los elementos, las relaciones entre ellos, etc.

Paso 5: **Crear instancias de los elementos arquitecturales y asignar responsabilidades**

Las responsabilidades de los elementos instanciados son asignadas de acuerdo a los requerimientos funcionales asociados a los drivers arquitectónicos. Las responsabilidades asociadas al elemento padre o elemento original se distribuyen entre los distintos elementos hijos o instancias. Al final de este paso todos los requerimientos asociados al elemento padre deben de estar representados como una secuencia de responsabilidades en los elementos hijo. En este paso sí se toman decisiones de diseño.

Paso 6: **Establecer las interfaces para los elementos instanciados**

En este paso se definen los servicios y propiedades requeridas y provistas por el elemento que se está diseñando, estas propiedades y servicios se refieren a las interfaces del elemento con otros elementos del diseño. En este paso sí se toman decisiones de diseño como las interfaces externas del sistema, las interfaces entre las

aplicaciones, etc.

Paso 7: **Verificar y refinar los requerimientos y hacer restricciones para las instancias de los elementos**

Se verifica que la descomposición del elemento cubra los requerimientos funcionales, atributos de calidad y restricciones de diseño (drivers arquitectónicos). Las responsabilidades asignadas a cada elemento hijo se transforman en requerimientos para el elemento en particular. Se refinan los atributos de calidad de ser necesario para cada elemento hijo. Durante este paso no se toman decisiones de diseño.

Paso 8: **Repetir el paso 2 hasta el *7* para el siguiente elemento del sistema que se desee descomponer**

Al terminar el paso 7 se tiene un elemento padre con elementos hijos que tienen ahora responsabilidades, una descripción de interface y requerimientos funcionales, atributos de calidad y restricciones de diseño. Ahora se puede seleccionar el siguiente elemento a descomponer. Se hacen tantas iteraciones como sea necesario hasta que estén satisfechos los drivers arquitectónicos.

La forma en que trabaja ADD permite que se obtenga el diseño de la arquitectura a través de la toma de decisiones, pero no indica la forma en que se puede hacer esta toma de decisión. El método refiere en el paso 4 que se debe de tener experiencia para determinar el patrón o patrones que se utilicen.

2.4.2. ACDM

ACDM (The Architecture Centric Development Method) es un método que cuenta con varias etapas tanto para obtención de drivers arquitectónicos, planteamiento de alcances, creación de una arquitectura conceptual (diseño de la arquitectura), etc, cubriendo con éstas el Ciclo de Vida del Desarrollo de la Arquitectura. En la figura 8 se muestra un diagrama de actividad de las etapas de ACDM, las cuales se describen brevemente en el Apéndice A de ésta tesis.

En este caso, el enfoque será sobre la creación de una Arquitectura Conceptual, ya que es aquí donde se hace el diseño de la arquitectura.

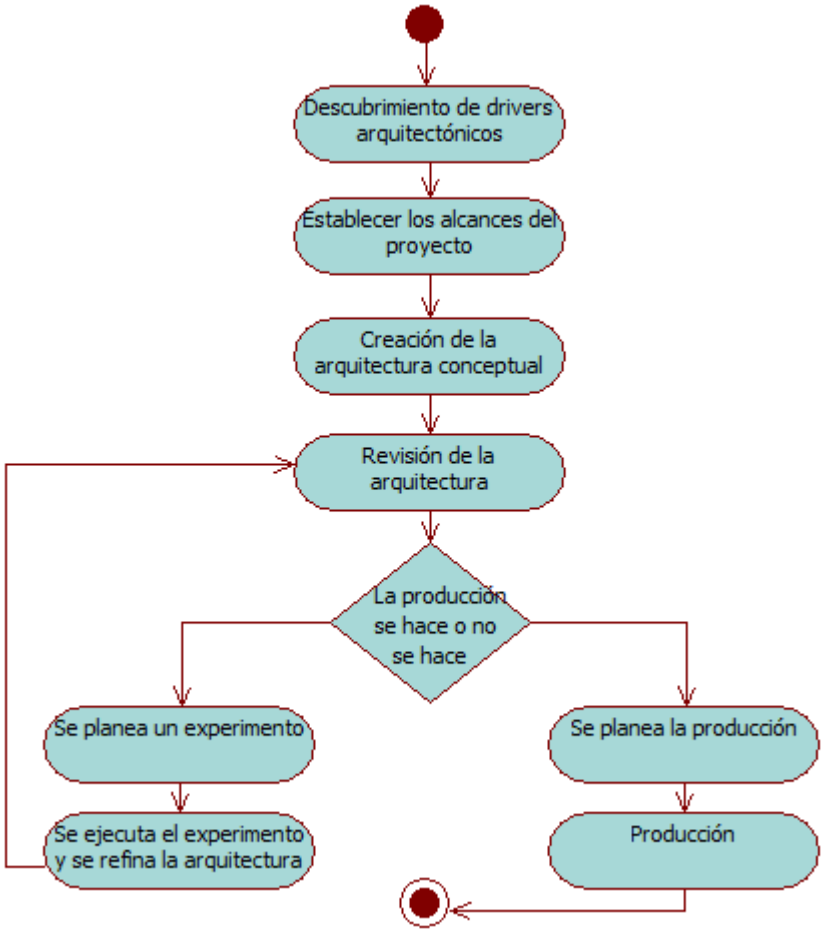


Figura 8: Etapas de ACDM - método iterativo para hacer el desarrollo de la arquitectura

Creación de una Arquitectura Conceptual

Se crea una arquitectura inicial, a la cual se le llama *conceptual* debido a que no se ha construido y sólo es el diseño. La arquitectura incluye una vista de ejecución (run-time view), una vista de código o de desarrollo (code view) y una vista física (physical view). En esta etapa se usan los drivers arquitectónicos como base para crear la Arquitectura Conceptual, que se tomará como un plano para la construcción el sistema. La idea principal de ACDM es que no se lleve mucho tiempo en obtener los drivers arquitectónicos, ni en la creación de la arquitectura. La arquitectura se hace en varias iteraciones hasta que se considere adecuada para el proyecto. Tomando en cuenta que ésta es sólo un borrador, no se le invierte mucho tiempo en cada iteración.

Para desarrollar la arquitectura conceptual se consideran los siguientes aspectos:

- Crear un Diagrama de Contexto

- Hacer una Partición del sistema
    - Selección de un patrón inicial – empieza la descomposición del sistema
- Crear Estructuras y Vistas
- Crear interfaces

Crear un Diagrama de Contexto

Estos diagramas establecen la parte interna y externa del sistema sobre la cual se va a desarrollar. Estas partes pueden ser impuestas por las restricciones de diseño y tienen que ver con dispositivos externos, redes u otros sistemas. Básicamente se muestran los límites del sistema, así como componentes internos y externos del sistema con los cuales interactúa.

Hacer una Partición del Sistema

Se divide el sistema para tener partes más pequeñas y trabajar con ellas. Con base en los drivers arquitectónicos obtenidos en la etapa 1 de *Descubrimiento de drivers arquitectónicos* y refinados en la etapa 2 de *Establecer alcances del proyecto* se hace la partición del sistema. Las restricciones pueden también dictar particiones, sin embargo la mayor influencia en las particiones es determinada por los atributos de calidad, que están clasificados de mayor a menor importancia y dificultad. Para hacer estas particiones, los arquitectos se basan particularmente en la experiencia adquirida por años, por lo tanto cada arquitecto puede tener una perspectiva diferente.

Los sistemas se diseñan a partir de un conjunto de patrones. Para empezar a hacer la arquitectura conceptual o diseño se selecciona un patrón que proveerá de una partición o descomposición inicial. La selección del patrón debe ser basada en el contexto de los objetivos de negocio y los atributos de calidad, esta decisión debe ser documentada. El o los elementos resultantes serán también particionado en iteraciones posteriores de esta etapa (Creación de una Arquitectura Conceptual), teniendo cada vez más detalle para la arquitectura.

Crear Estructuras y Vistas

La forma de documentar la decisiones de diseño que se toman es por medio de vistas que representan las estructuras. Las vistas usadas son:

Vista en tiempo de ejecución o dinámica – Representa elementos en tiempo de ejecución por ejemplo procesos.

Vista de código o vista lógica – Representa código orientado a los elementos, por ejemplo componentes.

Vista física – Representa elementos físicos como son los nodos (hardware).

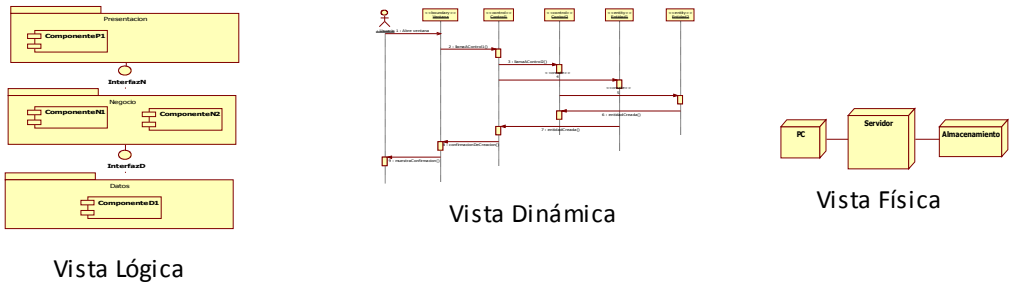


Figura 9: Vistas expresadas en diagramas de UML

Se deben de considerar como mínimo las tres vistas mencionadas anteriormente, ya que estas ayudarán a entender las propiedades del sistema y a que el sistema sea construido.

Crear Interfaces

Las interfaces son acuerdos entre los elementos, que definen la forma en que se van a relacionar, ésto quiere decir que definen los servicios y datos que necesitan y proveen los elementos.

Después de la descomposición inicial se procede a descomponer los elementos resultantes sucesivamente. Se considera que la arquitectura conceptual está completada cuando se tienen los siguientes puntos cubiertos:

- El sistema debe estar particionado en elementos y los elementos deben tener relaciones entre ellos. También cada elemento debe de tener una representación en las tres vistas mencionadas anteriormente, con las cuales se describe la arquitectura del sistema.
- Las responsabilidades de cada elemento deben de estar definidas y documentadas.
- Los datos y servicios que brinda cada elemento deben de estar definidos y documentados.
- Los datos y servicios requeridos por cada elemento deben de estar definidos y documentados.
- Las interfaces de cada elemento deben de estar definidas.

Con esto se asegura que se tiene la arquitectura conceptual. La finalidad para hacer esta arquitectura es, asegurar que se cubren los atributos de calidad del sistema.

El aspecto que sería equivalente al paso 4 de ADD en este método es el de *Hacer una Partición del Sistema*, debido a que es esta sección donde se determina la forma en que se va a dividir el sistema pero además se aplican patrones. Al igual que en ADD, en ACDM se refiere que para hacer esta toma de decisión es necesario la experiencia del arquitecto.

2.4.3. RUP

RUP (The Rational Unified Process) es un proceso de desarrollo de software que se lleva a cabo en iteraciones, de acuerdo a las necesidades del proyecto. Esta compuesto de cuatro fases, una de incepción o inicio, una de elaboración, una de construcción y finalmente una de transición. En cada una de estas fases se pueden realizar iteraciones, según se necesiten. Dentro de cada iteración se realizan actividades correspondientes a distintas disciplinas que en RUP son descritas a través de flujos de trabajo, donde un flujo de trabajo es una secuencia de actividades que dan resultados observables [11]. El esquema que maneja RUP es el siguiente:

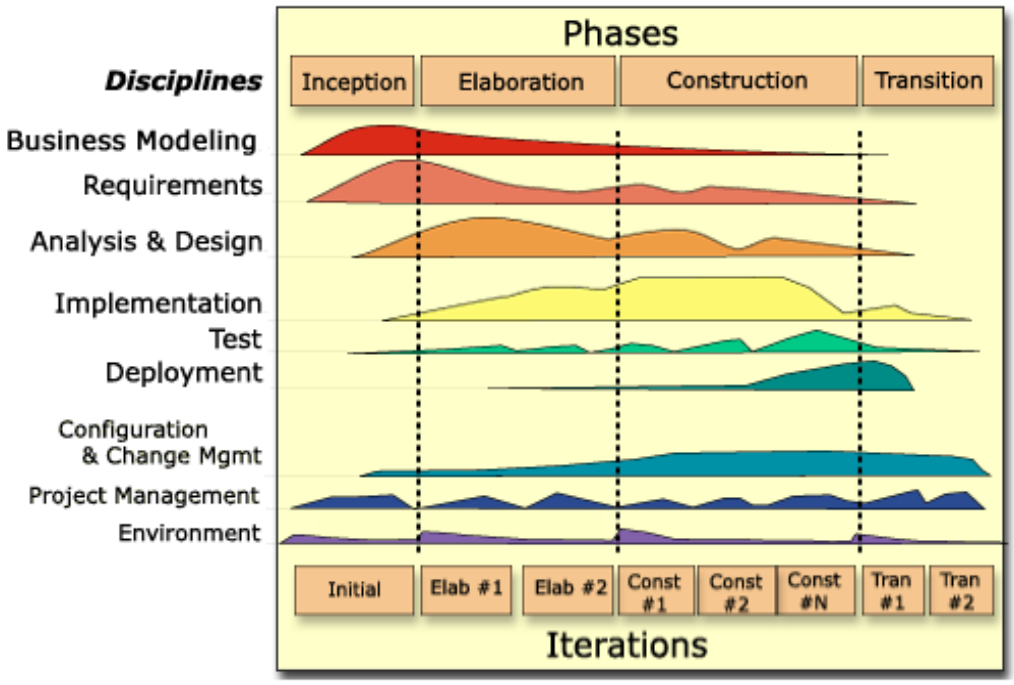


Figura 10: Estructura del proceso de desarrollo RUP

- Fase de Inicio. En la fase inicial se define lo que se va a desarrollar, quien y como se va a hacer, así como los entregables, donde entregable es todo aquel material que va realizando a través de cada fase. Se inicia la documentación de los entregables que van a apoyar a la planeación y construcción de la arquitectura. En esta fase se pueden identificar algunas actividades así como algunos otros elementos que se relacionan con la arquitectura, como es el flujo de trabajo llamado *Realizar la Síntesis de la Arquitectura* que se describirá más adelante.
- Fase de Elaboración. El objetivo de esta fase es obtener una arquitectura estable que sirva como base para el diseño e implementación del sistema en la fase de Construcción. Para esta fase los flujos relacionados con la arquitectura son *Definición de una Arquitectura Candidata* y *Refinar la Arquitectura* que se describirán más adelante.

- Fase de Construcción. En esta fase se trabaja con la implementación del sistema basado ya en la arquitectura obtenida en la fase anterior. En este punto la arquitectura debe estar totalmente refinada y probada.
- Fase de Transición. En esta fase se hace entrega del sistema al usuario final. Aquí es donde se prueba que todo lo hecho anteriormente haya sido correcto, por lo tanto la arquitectura, por ser la base del sistema tendrá que haber sido totalmente probada e implementada.

Este proceso usa, para guiar el desarrollo, el paradigma orientado a objetos, por lo tanto usa las clases de las entidades del dominio del problema para pasar a objetos en memoria.

Disciplinas

Una disciplina es un conjunto de actividades de acuerdo a lo que se desarrolla en cada una de ellas. Cada fase cuenta con las mismas disciplinas, pero no todas las fases se desarrollan de la misma manera o con las mismas actividades. De la misma forma, no en todas las disciplinas se trabaja con la arquitectura del sistema. Las disciplinas donde se trabaja con la arquitectura se detallaran a continuación.

Disciplina de Análisis y Diseño

La disciplina donde se trabaja con la arquitectura es la de *Análisis y Diseño*. Lo que se busca es tener una arquitectura sólida para el sistema. Dentro de esta disciplina se encuentran flujos de trabajo donde se desarrolla la arquitectura. Los flujos de interés para analizar son los siguientes:

- Realizar una Síntesis de la Arquitectura (opcional)
- Definir una Arquitectura Candidata
- Refinar la Arquitectura

Realizar una Síntesis de la Arquitectura

El propósito de este flujo de trabajo es probar que el proyecto es factible, esto se hace a través de la búsqueda de una solución arquitectónica para el sistema. Para lograr este propósito se realizan varias actividades:

Análisis de la Arquitectura - Define una Arquitectura Prueba que servirá para determinar si el proyecto es factible o no. Es sólo un borrador o bosquejo de la arquitectura.

Construcción la Arquitectura Prueba - Esta actividad sirve para sintetizar o simplificar alguna solución que cumpla con los requerimientos críticos para la arquitectura. Estos requerimientos son aquellos arquitecturalmente significativos.

Evaluación de la Arquitectura Prueba – En esta actividad se evalúa que la arquitectura cubra los requerimientos arquitectónicos críticos y puedan ser alcanzados por la solución propuesta, esto es, que sean satisfechos.

Definir la Arquitectura Candidata

Los objetivos principales de este flujo son, hacer un esbozo de la arquitectura, definir un conjunto inicial de elementos significativos para la arquitectura y mecanismos de análisis. Este flujo tiene dos actividades:

Análisis de la Arquitectura - Define una arquitectura candidata basado en la experiencia con proyectos similares, también define los patrones necesarios para la arquitectura candidata. Al haber definido los patrones, se crean las vistas necesarias para su representación:

- Vista de Casos de Uso
- Vista Lógica
- Vista de Procesos
- Vista Despliegue
- Vista de Implementación
- Vista de Datos

Análisis de Casos de Uso - Esta actividad muestra las responsabilidades, atributos y las relaciones entre las clases en el sistema, basados en los Casos de Uso. Un Caso de Uso es una técnica para representar escenarios de las acciones que realizará el sistema [6], estas acciones son los requerimientos funcionales de importancia para la arquitectura.

Refinar la Arquitectura

Completa la arquitectura que se obtuvo del flujo de trabajo anterior. Permite que se haga una transición entre las actividades de análisis a las actividades de diseño. Mantiene la consistencia e integridad de la arquitectura en caso de que ya se haya tenido algún diseño anterior. Las actividades que se realizan en este flujo son las siguientes:

Identificar los Mecanismos de Diseño - Refina los mecanismos de análisis que se usan dentro de los mecanismos de diseño, basados en las limitaciones del entorno de aplicación.

Identificar los Elementos de Diseño - Analiza las interacciones de las clases para poder identificar los elementos del modelo de diseño.

Incorporar los Elementos de Diseño Existentes - Analiza las interacciones entre las clases de análisis para encontrar interfaces y define la arquitectura incorporando elementos si es posible.

Estructurar el Modelo de Implementación o Aplicación - Establece la estructura en que la implementación va a residir y asigna responsabilidades de los subsistemas y sus contenidos.

Descripción de la Arquitectura en Tiempo de Ejecución - Se identifican procesos y sus mecanismos de comunicación, así como sus ciclos de vida.

Descripción de la Distribución - Describe como están distribuidos los nodos físicamente en el sistema.

Revisión de la Arquitectura - Descubre cualquier riesgo, fallas o desperfectos que no se haya tomado en cuenta

en la fase de inicio o en alguna iteración pasada. De la misma forma sirve para evaluar los atributos de calidad e identificar posibilidades de reuso de componentes y elementos.

Esta es la ultima actividad de este flujo, y de los flujos relacionados con la arquitectura en la fase de elaboración de RUP. Este flujo de trabajo sirve básicamente para analizar los resultados de la arquitectura ejecutable y finalmente refinarla.

La parte que sería equivalente al paso 4 de ADD descrita en este proceso es la correspondiente a la actividad de *Análisis de la Arquitectura*, del flujo de trabajo de Definir una Arquitectura Candidata de la fase de Elaboración. En esta actividad se hace la toma de decisión con base en la experiencia con proyectos similares.

Para los fines de éste proyecto, se hace un análisis y una comparación entre el método ADD del SEI, la creación de la Arquitectura Conceptual de ACDM y el trabajo realizado en los dos flujos de trabajo dedicados al diseño de la arquitectura de RUP llamados Definir una Arquitectura Candidata y Refinar la Arquitectura. Esta comparación se presenta a continuación.

2.4.4. Tabla Comparativa de los Métodos

Para poder entender las diferencias entre los métodos estudiados anteriormente, se muestra la siguiente tabla comparativa (tabla 1).

	ADD – Atribute-Driven Design del SEI	**Arquitectura Conceptual de ACDM**	**Definir una Arquitectura Candidata y Refinar la Arquitectura de RUP**
Base para realizar el diseño de la Arquitectura	Requerimientos funcionales arquitecturalmente importantes, atributos de calidad y restricciones (drivers arquitectónicos)	Requerimientos funcionales arquitecturalmente importantes, atributos de calidad y restricciones (drivers arquitectónicos)	Requerimientos funcionales, arquitecturalmente importantes, atributos de calidad y restricciones (drivers arquitectónicos)
Forma de hacer el diseño de la Arquitectura	Descomposición iterativa del sistema en elementos, empezando por el sistema visto como un sólo elemento	Descomposición iterativa del sistema empezando por la selección de un patrón inicial	Descomposición iterativa del sistema a través del flujo de trabajo *Realizar una Síntesis de la Arquitectura* y de su actividad *Análisis de la Arquitectura*
Aspectos que se toman en cuenta para dar prioridad a los requerimientos, atributos de calidad y restricciones	De mayor a menor importancia para los involucrados específicamente en el paso 3 – *Identificar los Drivers Arquitectónicos*	De mayor a menor importancia y dificultad	La priorización es obtenida con base en documentos que se utilizan como entrada al llegar a la disciplina de *Análisis y Diseño*, tales como el *Documento de Visión* y *Modelo de Casos de Uso*
La nociones de diseño se hacen con base en ...	Experiencia del arquitecto	Experiencia del arquitecto	Se basa en la documentación ofrecida a través de los distintos flujos de trabajo y sobre todo en la experiencia del arquitecto con sistemas similares
Forma de documentar	A través de vistas	A través de vistas: Vista dinámica Vista de código o lógica Vista física	A través de vistas y documentos: Vista de Casos de Uso Vista Lógica Vista de Procesos Vista de Despliegue Vista de Implementación Vista de Datos Documento de *Arquitectura de Software*
Verificación y validación de las decisiones tomadas	Se verifica que la descomposición del sistema satisfaga los requerimientos funcionales, atributos de calidad y restricciones de diseño (drivers arquitectónicos), en el paso 7	Se verifica que estén cubiertos los siguientes aspectos que debe tener el sistema: Que esté particionado en elementos y que estos elementos tengan relaciones entre ellos Que las responsabilidades de cada elemento estén definidas y documentadas Que los datos y servicios que brinda y requiere cada elemento estén definidos y documentados Que las interfaces de cada elemento estén definidas	La verificación se realiza en el flujo de trabajo *Refinar la Arquitectura* en la actividad *Revisión de la Arquitectura*

Tabla 1: Comparación entre los distintos métodos de diseño de la arquitectura de software

La mayor diferencia entre estos métodos es la forma de documentar los resultados, ya que mientras ADD y ACDM utilizan principalmente las vistas, RUP utiliza aparte documentos escritos como se menciona en la tabla 1. También se observa que los métodos cuentan con aspectos muy similares, como la descomposición del sistema en partes más pequeñas. Otro aspecto compartido es el hecho de que se apoyan en el uso de patrones para hacer el diseño de la arquitectura. Como ya se mencionó, para utilizar estos patrones se requiere experiencia ya que ninguno de los métodos da una guía específica para hacer esta toma de decisiones de diseño. Existen algunas herramientas y métodos que ayudan a hacer esa toma de decisiones de diseño. De algunos de ellos se hablará a continuación.

2.5 Estado del Arte

Hoy en día existen distintas herramientas que ayudan a realizar la toma de decisiones de diseño a través de la búsqueda y selección de soluciones de diseño. Éstas pueden ser desde sistemas que propongan como solución un diseño de arquitectura completo, hasta sistemas o métodos que propongan soluciones en cada iteración de algún método de diseño. A este conjunto de soluciones de diseño que se obtienen se les conoce como conocimiento arquitectural [19], o sea, el cómo el sistema fue diseñado y las razones por las cuales fue diseñado de tal forma. Este conocimiento se puede almacenar en repositorios, bases de datos, catálogos, etc. Los patrones se pueden ver como soluciones de diseño, o sea, la forma en que es diseñado el sistema, por lo tanto es considerado como conocimiento arquitectural, el cual está documentado y almacenado en catálogos y algunos repositorios.

En esta sección se describen algunas de estas herramientas que auxilian en la toma de decisiones de diseño, con las cuales se pueden hacer uso del conocimiento arquitectural (decisiones de diseño) y en algunos casos de su documentación, empezando con las que describen soluciones completas en el diseño de la arquitectura hasta las que proponen soluciones de forma más particular como son los patrones.

2.5.1. REBUILDER

Es un sistema computacional que tiene como propósito almacenar el conocimiento generado en el proceso de desarrollo de software, para después poder ser manejado y reutilizado, entre este conocimiento almacenado se encuentra el conocimiento arquitectural [27]. Esta herramienta utiliza como núcleo de funcionalidad el enfoque Case-Based Reasoning (CBR) que se basa en el re-uso de experiencias, las cuales están almacenadas en una base de datos léxica en forma de casos (cases). La base de datos es WordNet [28], la cual ayuda al análisis automático de los textos almacenados permitiendo tener interacción con la herramienta con un lenguaje natural.

REBUILDER está compuesto de cuatro módulos: el editor UML, el manejador de la base de conocimiento, la base de conocimiento (base de datos), y el CBR. El editor UML es la interface de REBUILDER y el ambiente donde se encuentra el diseñador de software, también es el módulo que integra nuevos diseños para re-uso. El manejador de la base de conocimiento es usado para administrar y mantener consistente la base de conocimiento. La base de conocimiento está compuesta de una librería de casos (case library) que es un repositorio centralizado, un indice para poder hacer la recuperación de los casos, una taxonomía de los tipos de datos y de WordNet.

En REBUILDER cada caso representa situaciones en donde fueron aplicados patrones, los cuales están representados por diagramas de clase de UML. La recuperación de los casos se hace a través de una consulta sobre la base de datos, ofreciendo como resultado una lista de casos o experiencias clasificadas por la misma consulta, es decir, de acuerdo a las necesidades que se hayan expresado.

2.5.2. Stylebase

Stylebase [29] es una herramienta que sirve para crear, mantener e investigar sobre el conocimiento arquitectural del software y es usado exclusivamente para aplicaciones elaboradas sobre eclipse. Stylebase además de permitir compartir conocimiento, apoya en la selección de soluciones que ayudan a satisfacer los objetivos de atributos de calidad. En esta selección se hace la búsqueda de distintos tipos de soluciones con los que puede trabajar con la herramienta, como son los patrones arquitectónicos, de diseño, referencias arquitectónicas y micro y macro arquitecturas o tomando como solución a un conjunto de ellos. Esta información se encuentra almacenada en una base de datos, donde se pueden encontrar imágenes, modelos de cada una de las soluciones almacenadas, así como información escrita (descripciones). La herramienta no menciona como hacer la conexión entre las soluciones encontradas, de tal modo que no se puede saber si dos de las soluciones se puedan perjudicar al usarlas en el mismo sistema [30].

2.5.3. PAKME

PAKME (Process-base Architecture Knowledge Management Environment) es una herramienta que permite el manejo de conocimiento arquitectural de diferentes tipos, tal como escenarios arquitectónicos, arquitecturas específicas, patrones y tácticas [31]. Esta información puede ser consultada ya que se encuentra almacenada en una base de datos, por tal motivo también se puede agregar información a la herramienta.

PAKME está montada sobre una herramienta colaborativa llamada Hipergate, la cual permite el manejo del conocimiento. La búsqueda de éste se puede hacer de varias forma, una es por medio de una palabra clave (keyword) que sea significativa, también permite hacer la búsqueda por medio de operadores lógicos como son And, Or y Not. La herramienta muestra un reporte del conocimiento generado, así como de las relaciones y efectos entre los diferentes artefactos arquitectónicos, entendiendo por artefactos a los diferentes tipos de conocimiento arquitectónico utilizados. PAKME consta de cuatro componentes: la interface de usuario; el manejador de conocimiento el cual brinda servicio de almacenamiento, recuperación y modificación de conocimiento arquitectónico; búsqueda; reporte.

2.5.4. ADDSS

ADDSS (Architecture Design Decision Support System Tool) es una herramienta para el manejo de decisiones de diseño arquitecturales, la cual considera de gran importancia la razones por las cuales se toman estas decisiones de diseño [32]. La información que almacena esta herramienta es conocimiento generado anteriormente sobre otras arquitecturas, así como patrones arquitectónicos, de diseño, estilos arquitectónicos y texto escrito. Esta información es almacenada en una base de datos, la cual es accedida por medio de un navegador web.

ADDSS puede ser utilizada por diferentes usuarios por medio de una cuenta (en el navegador) y brinda a cada uno de ellos diferente información de acuerdo al usuario. Esta información es conocimiento arquitectural existente, es decir, decisiones tomadas por el mismo usuario para otros sistema como son patrones y estilos

arquitectónicos utilizados anteriormente. Debido a que sólo toma en cuenta las decisiones tomadas por el mismo usuario, no permite compartir este conocimiento a menos que el usuario decida hacerlo.

ADDSS permite registrar en cada iteración de algún proceso de diseño de arquitectura las decisiones tomadas y de esta forma mostrara la evolución de la misma, ésto a través de vistas. Las decisiones y la arquitectura son documentadas y plasmadas en un reporte en formato PDF generado automáticamente. En el caso de la selección únicamente de patrones, no menciona la compatibilidad o incompatibilidad entre ellos.

2.5.5. Web Of Patterns (WOP)

Es una herramienta que facilita compartir y consultar conocimiento acerca de diseño de software [33], específicamente patrones de diseño. WOP consta de dos partes, un formato de lenguaje neutral para describir patrones de diseño y otra parte que consta de un conjunto de plugins de Eclipse los cuales utilizan este formato, de tal manera que permite buscar instancias de patrones en proyectos java y en repositorios en línea. Permite la búsqueda de patrones en linea dentro de listados de patrones (repositorio), blogs y servicios de marcadores (bookmarking), de tal forma que pueden ser incluidos en el repositorio que se esté utilizando. La búsqueda se hace usando reglas, estas reglas están publicadas usando un estándar llamado SWRL (Semantic Web Rule Language) [34]. Permite la evaluación de patrones, la cual es hecha por más usuarios. En este caso los patrones que son buscados y utilizados son de diseño [20].

2.5.6. ArchE

ArchE es un sistema experto creado por el SEI [22] para auxiliar en la toma de decisiones de diseño, enfocado en los atributos de calidad, patrones y tácticas de diseño. ArchE se basa en cuatro conceptos que son, escenarios de atributos de calidad, los cuales deben de estar especificados como el SEI lo sugiere [10] (Ver Apéndice A); marco de razonamiento, es el conocimiento generado al tomar decisiones de diseño y determinar si se ha alcanzado la satisfacción de los atributos, éstos da como resultado diseños iniciales (una propuesta de diseño); responsabilidades, éstas son actividades con las cuales el sistema se empieza a diseñar (requerimientos funcionales); tácticas, las cuales son decisiones de diseño que influyen en la respuesta de los atributos de calidad [10].

Para cada atributo de calidad existe un marco de razonamiento que convierte al escenario en un modelo de especificación del atributo de calidad. Cada modelo representa un diseño que satisface el correspondiente atributo de calidad. Si no se satisface el atributo de calidad con el modelo propuesto, se aplican tácticas de diseño. Las responsabilidades se usan para hacer conexiones entre los modelos creados por los marcos de razonamiento. A su vez los marcos de razonamiento resuelven conflictos entre los modelos de atributos de calidad y de esta forma se obtiene como resultado un modelo global que satisface a todos los atributos de calidad. Posteriormente éste modelo global se convertirá en la representación del diseño de la arquitectura.

Los marcos de razonamiento pueden ser creados para incrementar la experiencia del sistema experto por medio de una infraestructura colaborativa que permite contribuir a los marcos de razonamiento. Ésto lo hace por medio de una interface que proporciona un API (application programming interface) para la comunicación y por medio de una estructura orientada a objetos que soporta el desarrollo de los marcos de razonamiento. Estos son vistos como plugins compatibles para ArchE. Fue construido como un sistema Rule-based (sistemas de predicción basados en reglas) con los modelos de atributos de calidad vistos como Frames (marcos o estructuras). Tanto Rule-based como Frames son tecnologías de Sistemas Expertos.

Existe una gran interacción entre ArchE y el usuario, ya que éste es el que le proporciona la información necesaria para poder hacer el diseño, como son los atributos de calidad, las características que el sistema debe soportar y el último diseño que se hizo, por ejemplo en alguna iteración anterior. El uso de ArchE es iterativo, ya que se puede ir ocupando en cada iteración del proceso de diseño de la arquitectura de software [21].

2.5.7. Open Pattern Repositories

Es un repositorio en linea (web), enfocado en patrones y en algunas tecnologías de software que ayudan a hacer el diseño de la arquitectura. Tanto a las tecnologías como a los patrones los considera de la misma manera al hacer alguna búsqueda de soluciones, ésto es, como patrones, donde cada uno contiene una descripción del problema, una solución y el contexto en el que se pueda aplicar. Estos patrones (tanto tecnologías como patrones) están relacionados entre si.

Esta herramienta muestra una descripción de los patrones y su representación, muestra los patrones con los que está relacionado y el impacto sobre los atributos (esta sección no se encuentra en el demo de la herramienta). Clasifica a los elementos existentes en el repositorio, de tal forma que se tiene una sección de patrones arquitectónicos, otra de diseño y otra de tecnologías [35][36].

2.5.8. Modelo Cognitivo de Hinojosa

Un modelo cognitivo es un modelo que trata de resolver problemáticas por medio de la simulación, en el caso del Modelo Cognitivo de Hinojosa la problemática es enfocada a la elección de patrones para ser aplicados en el diseño de la arquitectura. Este modelo se enfoca en los patrones de diseños de Gamma [18], específicamente en los patrones de comportamiento (sólo son once patrones de diseño clasificados en este libro como patrones de comportamiento) y en situaciones reales donde se hace selección de éstos con los cuales se tuvo éxito [37]. Utiliza un lenguaje de programación lógica llamado Prolog, con el cual se mapean las características de los patrones. Debido a que cada uno de los patrones se tiene que transformar manualmente a un grupo de elementos comprensible para el lenguaje, es difícil de escalar el dominio de solución (número de patrones que se utiliza).

2.6 Comparación de las Herramientas

Para hacer una comparación de la herramientas mencionadas anteriormente, se consideran algunas de las características que se proponen en el artículo titulado "A Survey of Existing Approaches for Pattern Search and Selection" de Birukou [38]. Estas características son usadas para hacer análisis de herramientas de búsqueda y selección de soluciones de diseño:

- Rastreo (Crawling) – Busca nuevas ligas a soluciones (como patrones) automáticamente en Internet.
- Indexar (Indexing) – Después del rastreo se clasifican los resultados para poder hacer la búsqueda de patrones.
- Búsqueda (Searching) – Búsqueda a través de etiquetas, palabras claves, descripción de los patrones o alguna otra forma de búsqueda.
- Soporte la selección – Teniendo una gran cantidad de alternativas, seleccionar una o varias y mostrarlas en alguna lista (ordenada por clasificación, por relevancia o por algún método o algoritmo – sin tomar en cuenta necesidades).
- Recomendaciones de uso – Hacer sugerencias de cual solución usar de acuerdo a las necesidades.
- Despliegue y Detalle (Browsing) – Desplegar la lista de patrones disponibles y mostrar el detalle de cada unos de ellos donde se muestre la descripción completa de éste.
- Navegación (Navigation) – Poder ir de una solución o patrón a otro a través de ligas (links) o de relaciones entre ellos.
- Conexión (Linking) – Poder hacer las conexiones entre las soluciones de un mismo repositorio u otros.
- Colaboración – Permitir que los usuarios colaboren entre ellos, poder hacer comentarios y compartir experiencias con el uso de las soluciones.

De la misma forma que se toman en cuenta las características anteriores, se toman en cuenta las siguientes propiedades para completar la comparación:

- Dominio de manejo – Qué tipo de soluciones son manejadas en la herramienta.
- Requerir de pre-procesamiento de soluciones – Si es necesario de un procesamiento previo de las soluciones antes de aplicar la herramienta.
- Método – Si el enfoque provee un método para hacer la búsqueda o selección de las soluciones.
- Guía de soporte – Si se ofrece alguna guía que oriente en la aplicación de la herramienta.
- Fácil integración a métodos – Qué tan fácil es que la herramienta se pueda integrar a métodos o procesos utilizados para crear el diseño de la arquitectura (ADD, Arquitectura Conceptual de ACDM, etc.).

La facilidad de integración se refiere a que al hacer uso de la herramienta en el proceso de diseño, no sea necesario modificar el proceso. De ser necesario modificar estos procesos que esta modificación sea insignificante, es decir que no altere el proceso. Idealmente para ser de fácil integración sería necesario que los procesos de diseño de arquitectura no presenten modificación. Ésto último no se cumple con la integración de algunas herramientas. Finalmente con las características y propiedades mencionadas anteriormente se obtuvo la tabla 2 comparativa de herramientas.

De esta tabla 2 se puede concluir que casi todas las herramientas sólo usan un repositorio y es difícil que puedan utilizar otros repositorios diferentes, debido al formato en el cual están almacenadas las soluciones (Web Of Patterns por su naturaleza lo permite). Algunas de ellas muestran listas de soluciones pero que no necesariamente resuelvan el problema o satisfagan las necesidades. Un aspecto descubierto en este análisis es que no en todas las herramientas existe la posibilidad de saber como ir de una solución a otra (con excepción de ArchE). En cuestión del dominio que tienen las herramientas, sí existen gran variedad de soluciones en algunas herramientas, aunque predominan los patrones de diseño. La mayoría de las herramientas necesitan un pre-procesamiento de las soluciones, ya sea porque estas están en una base de datos o porque el método o lenguaje que se utiliza para hacer las búsquedas requiere tener determinado formato para poder almacenar las soluciones. Otro aspecto que se considera muy importante es la fácil integración de la herramienta a los procesos que se utilizan, ya que ésto afecta a empresas con procesos de desarrollo de software bien establecidos, en este caso la mayoría de las herramientas sí son de fácil integración. Aun a pesar de las herramientas, en muchos de los casos aun es necesario tener conocimientos sobre las soluciones propuestas, ya que sólo presentan una lista de soluciones sin recomendaciones.

Escenarios	REBUILDER	Stylebase	PAKME	ADDSS	Web of Patterns	ArchE	Open Pattern Repositories	Modelo Cognitivo de Hinojosa
Rastreo	No es posible debido a que las soluciones son situaciones que se representan por casos	No lo puede hacer ya que las soluciones se encuentran en una base de datos	No lo permite debido que la información está almacenada en una base de datos	Debido a que se utiliza una base de datos, no es posible tener esta característica	Permite buscar instancias de patrones en proyectos en Java o en repositorios en linea	No permite el rastreo en linea	No permite hacer rastreo	No lo permite por ser sólo un modelo
Indexar	No se aplica debido a que se almacena en una base de datos	No se aplica debido a que se almacena en una base de datos	No se aplica	No permite indexar	No permite indexar	No tiene una clasificación	Sin hacer rastreo, las soluciones están clasificadas por tipos de patrones y tecnologías	No están clasificados
Búsqueda	Por medio de consultas a la base de datos	Por el nombre, por alguna palabra o por atributos de calidad	Por medio de palabras clave o por operadores lógicos	Por medio de consultas a la base de datos	Utiliza marcadores o servicios de búsqueda	Se hace por medio de los escenarios de atributos de calidad	La búsqueda se hace por palabras clave	Se hace a través de situaciones que se quieren resolver
Soporte de Selección	Hace la selección de alternativas por medio de CBR, mostrando una lista de soluciones	Con base en la búsqueda, muestra los resultados	Selecciona varias alternativas y las muestra	Muestra una lista de soluciones	Solo muestra los patrones	Permite hacer la selección de las soluciones	Muestra las alternativas sugeridas por la selección	No se define en la fuente de información
Recomenda-ciones	La lista está clasificada de acuerdo con la consulta	No hace más recomendaciones que la búsqueda	No hace más sugerencias que las de la búsqueda	No realiza recomendaciones	No hace alguna recomendación	Hace recomendaciones de acuerdo a los escenarios	No hace una recomendación en particular	Sí hace sugerencia
Despliegue y Detalle	Despliega la lista con diagramas UML	Despliega las soluciones con opción de poder ver su detalle	El resultado de la búsqueda lo despliega con opción de ver el detalle de los resultados	Despliega una lista de soluciones con su correspondiente detalle y una mini-gráfica de cada solución	Despliega una lista pero no muestra detalle de los patrones	Muestra detalle de la recomendación hecha	Despliega una lista de posibilidades y con ella la opción de ver el detalle de cada una de las opciones	No hace ningún tipo de despliegue y muestra detalles
Navegación	No se puede ir de una solución a otra, ya que los casos son experiencias concretas	No tiene relaciones entre las soluciones	No se muestra la relación entre las soluciones	Sólo se pueden ligar las decisiones tomadas con la arquitectura resultante	No se puede ir de una solución a otra	Se va de una solución a otra por medio de la información que se va ingresando	Permite la navegación con los patrones relacionados (no en el demo)	No permite la navegación entre los patrones
Conexión	No se puede hacer ésto ya que se utiliza una base de datos	No puede hacer conexiones	No puede hacer conexiones	No se puede hacer conexiones hacia otros repositorios	Sólo con algunos repositorios (Swoogle, wopper server))	No permite hacer conexiones con otros repositorios	Sólo con el repositorio de la herramienta	No lo hace ya que sólo usa once patrones
Colaboración	No se reporta que acepte colaboración	Sí permite la colaboración de tal forma que se puede añadir o eliminar soluciones	Sí permite la colaboración de los usuarios, ya que éstos pueden añadir y modificar las soluciones	No permite que se compartan los diseños a menos que el usuario creador decida colaborar con otros proyecto	Permite que los usuarios puedan tener colaboración	Permite la colaboración debido a que es un sistema experto	Si permite la colaboración añadiendo y editando las soluciones	No se reporta que acepte colaboración
Dominio de Manejo	Patrones de Diseño	Patrones arquitectónicos, de diseño, referencias arquitectónicas y micro y macro arquitecturas	Escenarios arquitectónicos, arquitecturas específicas, patrones y tácticas	Patrones y estilos arquitectónicos	Patrones de diseño	Patrones y tácticas de diseño	Patrones y tecnologías útiles	Once patrones de diseño
Requerir de Pre-procesamien-to de Soluciones	Cada solución se debe transformar en "caso" para ser almacenada	Las soluciones se deben almacenar en la base de datos	Se deben de almacenar las soluciones con su descripción	Se requiere almacenar los conocimientos en la base de datos	Es necesario que se almacenen los patrones con cierto formato	Sí se requiere de un pre-procesamiento de las soluciones	Se requiere almacenar los datos en el repositorio de la herramienta y los diagramas de patrones	Es necesario que los patrones estén expresados en clausulas de Horn, para poder ser usados por Prolog
Método	CBR – Case_Based Reasoning	No provee algún método	No provee un método, utiliza la herramienta llamada Hipergate	No provee un método	Jane – Framework de Web Semántica)	Rule-based (sistemas de predicción basados en reglas)	La fuente no menciona algún método	El método de búsqueda es el uso de reglas del tipo "Si en verdad el antecedente, entonces es verdad el consecuente"
Guía de Soporte	Artículo	Guía de usuario	Artículo	Artículo	Páginas de Internet	Guías	Guías	Artículo
Fácil Integración a Métodos	Es necesario que el proceso de diseño se adapte para que se pueda usar esta herramienta	Es necesario hacer el proyecto en eclipse.	No es necesario que el proceso de diseño se adapte para usar esta herramienta	Es fácil de integrar ya que sólo intervine en la búsqueda y selección de soluciones	Es necesario tener el proyecto en Eclipse para poder hacer uso de ella	No es fácil de integrar a otros métodos que no usen escenarios de atributos de calidad	Es de fácil integración ya que no es necesario tener un formato especial para su uso	No es necesario hacer cambios en los procesos para poder ser utilizado

Tabla 2: Comparación de Herramientas

Capítulo 3

Problemática y Propuesta

Como se puede observar, la creación del diseño de la arquitectura no es una tarea trivial. Al contrario, es necesario seguir procesos bien establecidos y aún más, tener experiencia en la creación de arquitecturas de software, como se sugiere en cada proceso analizado en al capítulo anterior. En este capítulo se presenta la problemática que se sustenta en los antecedentes expuestos previamente y describe la solución propuesta.

3.1 Problemática

En el proceso de diseño ADD se menciona que la toma de decisiones de diseño, especialmente las relacionadas con la selección de patrones que satisfagan los drivers arquitectónicos, particularmente los atributos de calidad, se hace en el Paso 4.

Figura 11: Paso 4 de ADD

Específicamente lo que sugiere este paso es hacer un listado de los patrones que satisfagan los atributos de calidad y, con base en la experiencia del arquitecto, hacer una selección de los que más convengan, teniendo como referencia para la selección escenarios correspondientes a los atributos de calidad de esa iteración.

La selección de estos patrones no es, sin embargo, una tarea simple dado que hay una gran cantidad de ellos. Además, existen varios patrones que satisfacen a los mismos atributos de calidad, las diferencias entre ellos son

que unos pueden satisfacer a un solo atributo de calidad o a más, así como el contexto en el cual se pueden aplicar. Otra diferencia es que aquel patrón que satisface a un atributo de calidad, puede perjudicar también a uno o más atributos de calidad.

3.1.1. Catálogo de patrones

Existen catálogos donde se encuentran plasmados los patrones, éstos puede tener algunas similitudes en la forma de documentar la información de los patrones, como pueden ser los siguientes puntos:

- Problemática – Descripción del problema de una forma general, de tal manera que se pueda aplicar en varios sistemas.
- Contexto – Entorno en el cual se desarrolla la situación o problema a atacar.
- Uso – Opciones de algunos usos que se le pueden dar al patrón descrito.
- Implicación – Las consecuencias de aplicar el patrón.
- Relación entre patrones – Cuales son los patrones con los que se relaciona el patrón descrito.

En la tabla 3 se mencionan algunos catálogos de patrones así como algunas de sus particularidades:

Título	Cantidad de Patrones	Enfoque	Particularidades
Patterns of Enterprise Application Architecture [24]	51	El enfoque que se le da al catálogo esta relacionado con el tipo de sistema empresarial	Los patrones son descritos por tres secciones, una para describir el patrón en forma conceptual, otra para explicar cuando se usa y la última para explicar el uso del patrón por medio de un ejemplo
Core J2EE Patterns Best Practices and Design Strategies [25]	21	El catálogo está enfocado a los patrones usados en proyectos realizados con J2EE	Los patrones se describen por medio del planteamiento de un problema, una serie de acciones que se quieren controlar a las cuales se les llama "fuerzas" y la descripción de la solución. Se muestra también la estructura que tiene el patrón por medio de un diagrama de clases, así como la relación que se tiene con el "cliente" mediante un diagrama de secuencia
Enterprise Integration Patterns [26]	65	El catálogo se enfoca en realizar una interacción entre los patrones, a través de los mensajes que se pueden mandar entre ellos	Los patrones están catalogados de acuerdo a los mensajes que se pueden enviar entre ellos y a la finalidad de estos patrones. Cada patrón está representado por un diagrama, en el cual se describe la forma en que trabaja al ser usado
Pattern-Oriented Software Architecture	114	Este catálogo se enfoca en el diseño e implementación de sistemas de software distribuido	Los patrones están catalogados de acuerdo a el uso que se les puede dar en un sistema. Cada uno de ellos está descrito por la necesidad que tiene el usuario, la problemática que se presenta al querer cubrir la necesidad y en tercer lugar se ofrece la solución al problema por medio de la aplicación del patrón y un diagrama de la solución propuesta

Tabla 3: Relación de algunos de los catálogos de patrones

Estos catálogos incluyen dentro de la descripción de cada patrón, información de los patrones con los cuales

está relacionado.

3.1.2. El problema de la toma de decisiones de diseño

La problemática sobre la cual se enfoca este trabajo ocurre al momento que se eligen patrones, como lo menciona el paso 4 de ADD, dentro del proceso de diseño de la arquitectura. De manera especifica, al querer elegir uno o un conjunto de patrones, el arquitecto se encuentra con que existe una gran cantidad de opciones posibles.

Un ejemplo de lo anterior se puede ilustrar tomando como base el catálogo de patrones POSA 4 [17]. A pesar de que en la mayoría de los catálogos de patrones se describen las relaciones entre ellos, este catálogo tiene la particularidad de que presenta a cada uno de los patrones en mapas, donde se plasma la relación que existe entre el patrón que se está tratando con algunos otros. La figura 12 presenta un mapa del catálogo de patrones de POSA 4, específicamente muestra la relación que existe entre el patrón Domain Model con aquellos enfocados a la estructuración inicial del sistema.

Los patrones son representados como nodos en el mapa y las aristas que unen a los nodos tienen asociada una necesidad. Al elegir un primer patrón, es posible elegir otro con el cual esté relacionado el primero. Esta selección se hace con base en las necesidades, tales como "Partición interna" o "Procesamiento de flujo de datos" (figura 12).

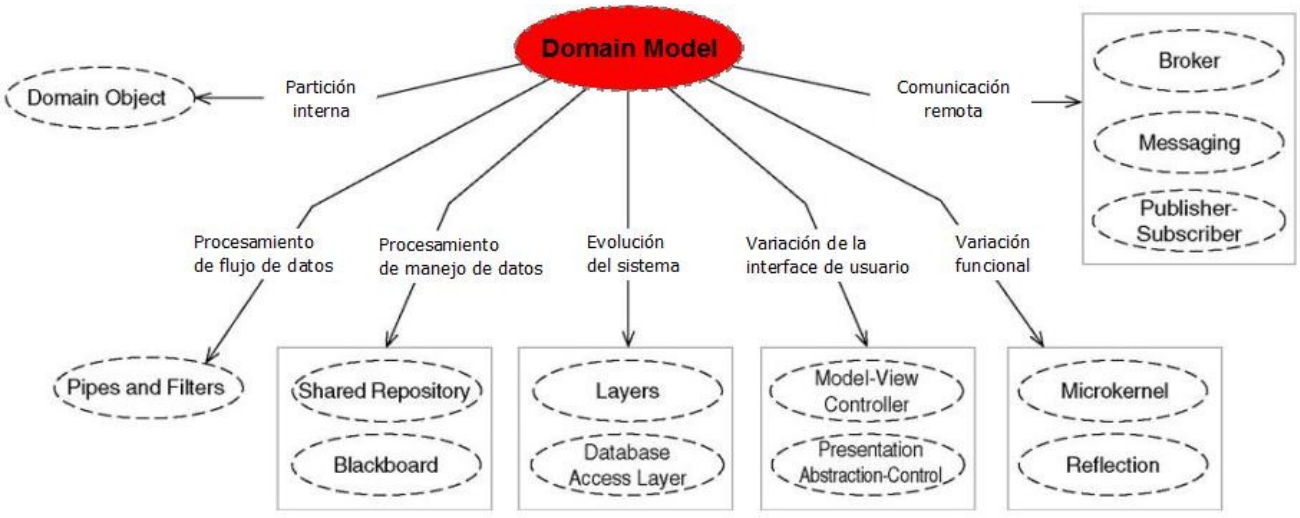


Figura 12: Mapa que representa los patrones relacionados con el patrón Domain Model de acuerdo con POSA 4 [17]

En la figura 12, se puede observar que al partir de un patrón específico, como puede ser Domain Model, no es trivial la elección del patrón que se va a aplicar después. Ésto se debe a que se tienen que tomar en cuenta los atributos de calidad que se quieren satisfacer, así como el nivel de descomposición del sistema y la gran cantidad de patrones que se pueden elegir. Por ejemplo si el arquitecto desea satisfacer el atributo de Modificabilidad y partir de Domain Model, se abren ante él un gran variedad de posibilidades ya que tiene como opciones todas las aristas, por lo tanto todos los patrones que son en este caso trece, de acuerdo con el mapa mostrado en la figura 12. La elección del patrón subsecuente depende de la experiencia del arquitecto, si es que la tiene, de lo contrario tendrá que leer la información relacionada con cada uno de los patrones. Este proceso de elegir un patrón para satisfacer atributos de calidad puede ocurrir varias veces dentro de una misma iteración en el diseño de la arquitectura, con diferentes patrones y sus correspondientes mapas.

Al unir cada uno de los mapas de los patrones contenidos en el catálogo POSA 4, dada la cantidad de ellos, se obtiene un grafo de patrones, como se muestra en la figura 13, donde cada nodo es un patrón y las aristas son las necesidades que llevan de un patrón a otro. En la figura 13 se observa la complejidad de la toma de decisión, aún teniendo como base este grafo, dada la gran cantidad de relaciones entre los patrones. Por otro lado, es importante recordar que el proceso de diseño de una arquitectura generalmente es enmarcado por un proyecto de desarrollo de software en donde el tiempo es limitado. Esto hace que un arquitecto (en particular si es poco experimentado) se enfrente con el reto de tener que tomar una gran cantidad de decisiones de diseño apropiadas para el proyecto y de manera rápida.

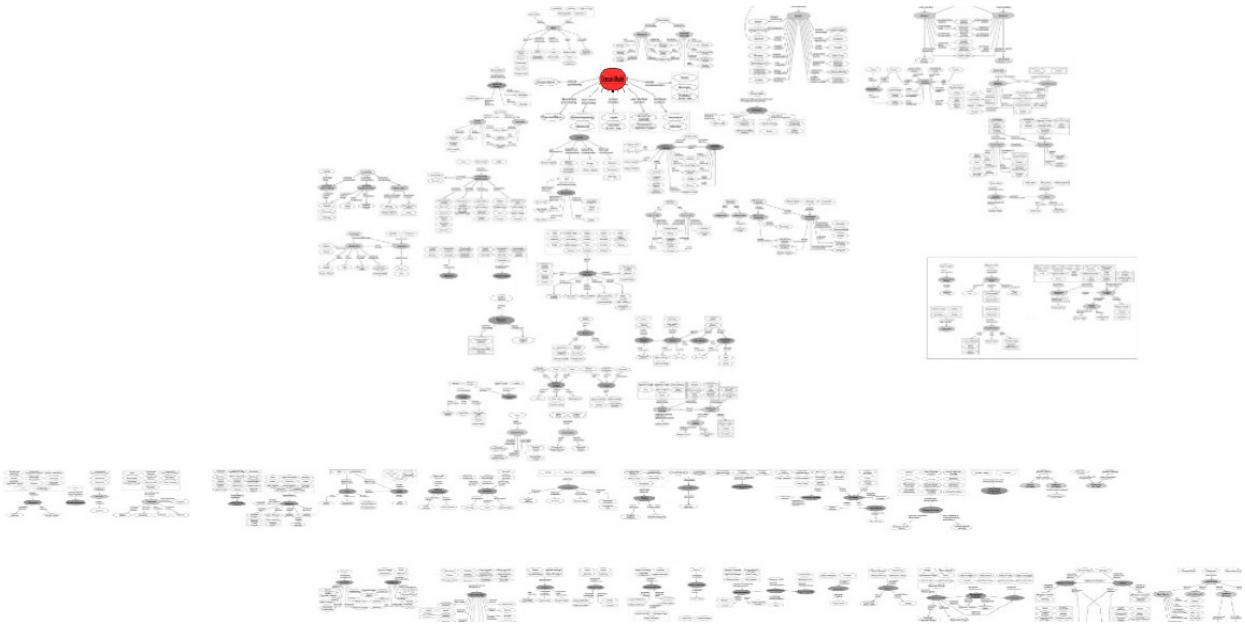


Figura 13: Grafo de patrones - unión de los mapas del catálogo POSA 4 [17]. El patrón Domain Model es el nodo rojo del grafo

La problemática de la toma de decisiones de diseño, durante una iteración de diseño de la arquitectura, puede

ser vista como un problema de recorrido parcial del grafo. Ésto se debe, en primer lugar, a que cada uno de los patrones son vistos como nodos (nodo-patrón), los cuales tienen relaciones con otros patrones a través de sus aristas o necesidades (aristas-necesidades) y a través de éstas se puede pasar de un patrón a otro. En segundo lugar, el recorrido es parcial debido a que éstos se hacen tomando en cuenta sólo una parte del grafo en cada iteración; de esta forma, en cada recorrido parcial se puede obtener una lista de patrones que satisfagan a ciertos atributos de calidad. Para hacer este recorrido parcial del grafo de patrones se deben considerar en todo momento los atributos de calidad de la iteración en curso, ya que el satisfacerlos es la finalidad de ocupar patrones.

En síntesis, la problemática se ubica en la elección de patrones que satisfagan atributos de calidad en una iteración del diseño de la arquitectura de software. Tomando en cuenta el grafo de patrones obtenido de POSA 4 [17], la problemática se puede plantear entorno a hacer búsquedas sobre el grafo para encontrar patrones, de tal forma que haciendo recorridos parciales sobre él se podrán obtener listas de patrones que satisfagan a los atributos de calidad de la iteración con la cual se trabaja.

3.2 Propuesta

Ante la problemática descrita anteriormente, se propuso crear un método asistente para la toma de decisiones de diseño de arquitectura de software (MATDDS). Para crear este método (MATDDS) se tomaron en cuenta varios aspectos.

Considerando que existe un grafo de patrones, se puede pensar en realizar, por cada iteración en el proceso de diseño de la arquitectura, un recorrido parcial del grafo de patrones como se mencionó anteriormente. De esta forma, si se tiene un patrón como punto de partida para el recorrido, se puede llegar a otro patrón que esté a uno o más niveles de profundidad. Como también se mencionó, el grafo contiene nodos, cada nodo es un patrón y las aristas que unen a los nodos son a lo que llamamos necesidades, como se muestra en la figura 14. El grafo de la figura 14 tiene un recorrido parcial, donde el nodo de inicio es el patrón marcado como “Patrón 1”, posteriormente se visita el nodo “Patrón 2” y finalmente el “Patrón 3”, realizando un recorrido parcial sobre el grafo.

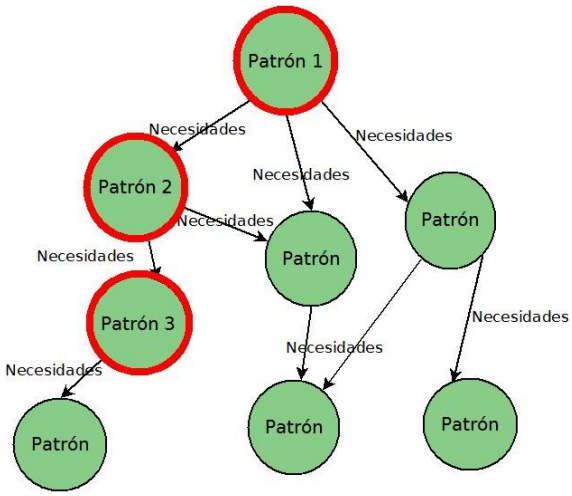


Figura 14: Grafo de patrones, donde los nodos son patrones y las aristas son necesidades

Generalmente cuando se hace la toma de decisiones de diseño en cada iteración, no se seleccionan los mismos patrones. El grafo de patrones permite que se inicie el recorrido parcial en cualquier nodo o patrón, de esta forma se puede ir haciendo la descomposición del sistema usando patrones distintos. Ésto está relacionado con el paso "Elegir un elemento para descomponer", ya que ahora se determina el elemento que se va a descomponer y con cuál patrón se va a iniciar, siendo generalmente con el patrón que produjo dicho elemento o con alguno de los que se trabajó en ese elemento.

Como se mencionó anteriormente, la finalidad de aplicar patrones es la de satisfacer atributos de calidad, por lo tanto el recorrido parcial se puede considerar como una forma de encontrar una lista de patrones que satisfagan los atributos (Patrón 1, Patrón 2, Patrón 3). Para poder encontrar las rutas que cubran las necesidades que se tienen en la creación de la arquitectura (satisfacción de los atributos de calidad) se propuso usar algún algoritmo de búsqueda.

3.2.1. Algoritmos de Búsqueda

Los algoritmos de búsqueda pretenden encontrar soluciones adecuadas para resolver algunos problemas, como los de optimización [22]. En ocasiones, esa pretensión no se cumple, ya que no siempre encuentran la solución optima para resolver el problema que se está atacando. Sin embargo son muy utilizados en distintos problemas, como aquellos que se presentan en las redes de comunicación.

Una aproximación posible a la resolución del problema anterior es el uso de un algoritmo llamado Algoritmo de Búsqueda A* (A estrella).

*3.2.1.1 Algoritmo de Búsqueda A**

El algoritmo de búsqueda A* es un algoritmo de búsqueda informada, ésto quiere decir que con base en cierta información se va a decidir seguir un camino sobre el grafo. Este algoritmo busca encontrar el camino más corto de un nodo de partida o inicial a un nodo objetivo o meta. Un nodo objetivo o meta es aquel al cual se desea llegar, éste puede o no ser conocido de antemano. Es muy importante mencionar que para este proyecto, el nodo meta no se conoce de antemano, ya que éste va a estar determinado cuando todos los atributos de calidad de la iteración con la cual se está trabajando estén satisfechos.

La búsqueda del nodo meta se basa en una función llamada *función de evaluación*, esta función debe estar contenida en cada nodo perteneciente al grafo que se va a recorrer, como se representa en la figura 15. La función es determinada a través de la fórmula f(h) = g(n) + h(n), donde n es el nodo actual, g(n) es el costo del recorrido del nodo inicial o raíz al nodo actual y h(n) es un valor heurístico que representa el costo de ir del nodo actual al nodo meta. El algoritmo A* es una combinación de dos algoritmos de búsqueda, la búsqueda de primero en anchura y la de primero en profundidad, donde g(n) representa el valor de primero en anchura y h(n) representa el valor de primero en profundidad.

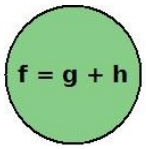


Figura 15: Función de Evaluación

La idea fundamental del algoritmo de búsqueda de primero en anchura es que partiendo del nodo inicial (o es su defecto, partiendo de cualquier nodo) se selecciona el nodo hijo que tenga menor g(h), o sea, el que tenga menor costo de llegar del nodo inicial al nodo hijo.

Para el algoritmo de búsqueda de primero en profundidad la idea principal es que a partir de un nodo (nodo actual), se tiene que seleccionar el hijo que tenga menor costo para llegar al nodo meta. Para obtener este costo, se utiliza un valor heurístico, el cual se determina por medio de una función, a ésto se le llama búsqueda heurística. Esta búsqueda es una técnica común para encontrar una solución en un árbol de decisión o en un grafo, donde existe una o más respuestas [22]. Se utiliza generalmente en problemas donde la solución exacta es desconocida o cuando se conoce un método, pero podría ser difícil o imposible su aplicación para resolver el problema [23].

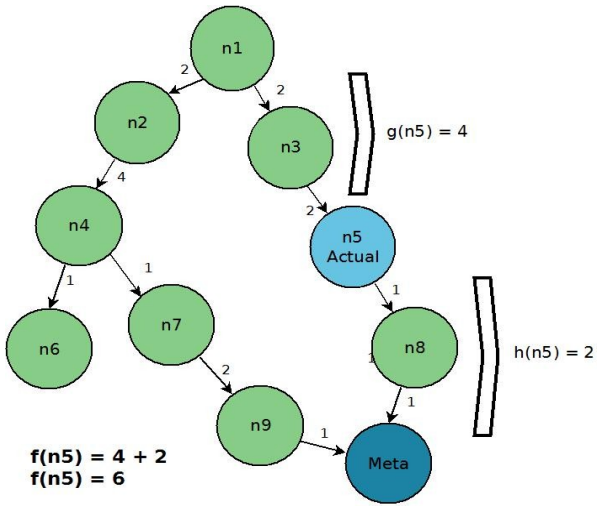


Figura 16: g(n) y h(n) vistos en un grafo

En la figura 16 se muestra cómo se ven los valores g(n) y h(n) en un grafo y la forma en que se obtienen sus valores, así como la forma en que se obtiene el valor de la función de evaluación f(n). En este caso g(n5) es igual a cuatro (g(n5) = 4) debido a que es la suma de los valores de las aristas que van del nodo inicial "n1" al nodo actual "n5". h(n5) es igual a dos (h(n5) = 2) debido a que es la suma de los valores de las aristas del nodo actual "n5" al nodo "Meta". El algoritmo A* encuentra el camino más corto, ya que va descartando nodos que ya se habían elegido y de esta forma acortando el camino.

Como se acaba de mencionar, el algoritmo descarta nodos para encontrar el camino más corto al nodo meta, ya que la finalidad de éste es la de encontrar el camino más corto. Por esta razón se concluye que no podría proveer una solución buena a la problemática. Ésto se debe a que al aplicar el algoritmo al grafo de patrones para querer satisfacer los atributos de calidad en una iteración e ir obteniendo un camino que lleve al nodo meta, en este caso al nodo con el cual se tengan la satisfacción de los atributos, va descartando patrones que ya se tenían elegidos dando como resultado que al final no se obtengan como resultado todos los nodos que satisfacen a los atributos de calidad de la iteración.

3.2.1.2 Algoritmo de Búsqueda en Profundidad

Ya que los resultados no son favorables a la problemática al aplicar el algoritmo A*, una posibilidad alternativa es la aplicación de un algoritmo de búsqueda en profundidad. Como se mencionó anteriormente, el algoritmo en profundidad busca saber cuál es el costo de llegar del nodo actual al nodo meta y tomando en cuenta ésto encoger el mejor camino. Una forma "rudimentaria" de aplicar este algoritmo es usando *backtracking* para encontrar el nodo meta. Primero se determina un nodo inicial, el cual expande sus nodos sucesores. De estos nodos sucesores se selecciona el primero de ellos (generalmente el de la izquierda), el cual se vuelve a expandir, así consecutivamente hasta llegar a los nodos hojas o nodos terminales. En los nodos terminales o cuando se

puede determinar si el nodo actual no va a llegar a alguna solución, se le da marcha atrás a la búsqueda. De esta forma la expansión de nodos procede con el nodo anterior recientemente expandido [22], hasta llegar al nodo meta. Este método no garantiza que se encuentre la solución optima, como se muestra en la figura 17, donde se observa que existe un mejor camino para llegar del nodo inicial al nodo meta, ésto se sabe ya que es más corto llegar por el camino con flechas azules (menor costo) que por el encontrado por el algoritmo. Por esta razón, el algoritmo en profundidad usando *backtracking* no siempre da un buen resultado.

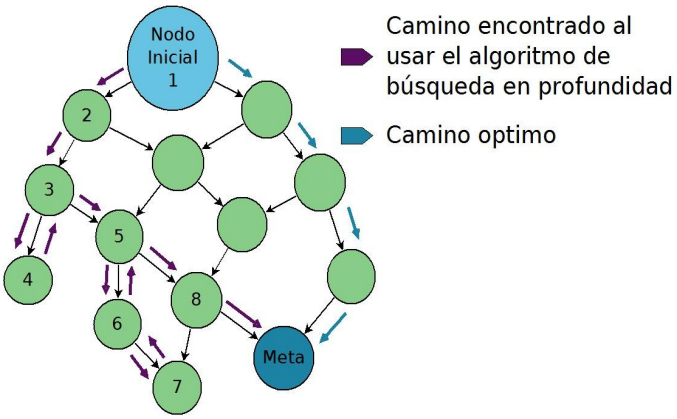


Figura 17: Recorrido en profundidad sobre un grafo

Otra forma de utilizar el algoritmo en profundidad es por medio del uso de una función heurística y valores heurísticos, a esta búsqueda se le llama búsqueda heurística. Al aplicar una función heurística y por lo tanto un valor heurístico, se determina el orden de expansión de los nodos, sin tener la necesidad de hacer el recorrido en profundidad para conocer el costo de llegar del nodo actual al nodo meta, ya que el costo está estimado por el valor heurístico. Este valor heurístico debe estar contenido en cada uno de los nodos del grafo que se desea recorrer.

Tomando como ejemplo la figura 17 del grafo y determinando el valor heurístico de cada uno de los nodos, de tal forma que se pueda llegar del *nodo actual* al *nodo meta*, se obtuvo la figura 18. Para este ejemplo el nodo meta es conocido de antemano. En la figura se observa que se encuentra el camino óptimo con base en los valores heurísticos. Ésto se logra seleccionando aquellos nodos con mejor heurística al ir recorriendo el grafo, donde la mejor heurística es la que tiene un valor menor, debido a que se desea encontrar el camino con menor costo y sabiendo que cada arista tiene un valor de uno se obtienen los valores heurísticos para cada nodo del grafo. Partiendo del *Nodo Inicial* se selecciona el nodo sucesor con mejor heurística. Los hijos del nodo seleccionado son los que se revisan para seleccionar nuevamente aquel que cumpla con la condición de mejor heurística.

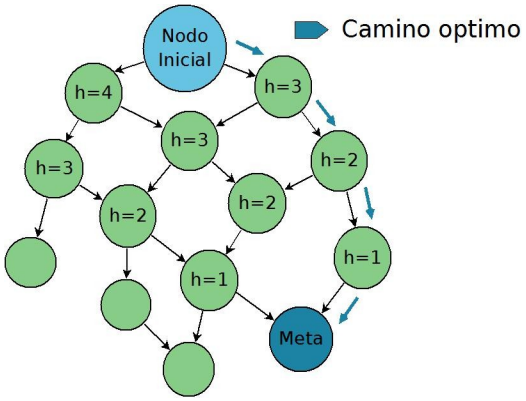


Figura 18: Recorrido en profundidad sobre un grafo usando valores heurísticos y donde se conocen de antemano el nodo inicial y el nodo meta

Para realizar la implementación del recorrido de esta forma, el algoritmo utiliza dos colas. Una cola de prioridad a la que se le llama cola de ABIERTOS donde se guardan nodos dependiendo de su valor heurístico. La segunda cola se usa para guardar los nodos que ya se han visitado, a la cual se le llama cola de CERRADOS. El algoritmo para realizar el recorrido en profundidad sobre un grafo usando valores heurísticos es el siguiente (presentado de forma general):

Teniendo determinado el *nodo de inicio* y la condición del *nodo meta* se inicia el algoritmo. La condición del *nodo meta* es aquella con la cual se decide terminar la búsqueda, como se mencionó anteriormente, ya sea que se sepa cuál es el nodo meta o se tenga que determinar, el valor heurístico se define como h (la determinación del nodo meta depende de cada caso donde se utilice el algoritmo en profundidad).

1. El nodo inicio se pone en la cola de ABIERTOS, con su correspondiente valor de h. Se inicializa la cola de CERRADOS como una cola vacía.
2. Repetir el siguiente procedimiento hasta que se encuentre el nodo meta:

   Se revisa si existen nodos en ABIERTOS, de no ser así el algoritmo termina sin solución. De lo contrario, tomar aquel nodo con mejor valor h y se nombrará a este nodo como MEJORNODO. Se verifica que MEJORNODO sea el NODOMETA. Si MEJORNODO es un NODOMETA se termina la búsqueda, por lo tanto el algoritmo llega a su fin. De no ser MEJORNODO el NODOMETA entonces se saca de ABIERTOS se mete a CERRADOS y se expande con sus sucesores o nodos hijos. Cada uno de estos nodos se verifican de la siguiente forma:

   1. Verificar si SUCESOR está en CERRADOS. En caso afirmativo, llamar VIEJO al nodo de CERRADOS y no meter a ABIERTOS. VIEJO es aquel nodo que ya se ha visitado anteriormente y no se usará más.
   2. Si SUCESOR no estaba en CERRADOS, ponerlo en ABIERTOS y calcular su valor h.

   Finalmente cuando los nodos hijos de MEJORNODO se han metido a ABIERTOS, como es una cola de prioridad, se acomodan los nodos de acuerdo a la mejor heurística. Se regresa al principio del paso 2.

3. El resultado de aplicar este algoritmo puede ser sólo el NODOMETA o el camino que se ha recorrido para llegar al NODOMETA.

El éxito de este algoritmo está fuertemente influenciado por el valor heurístico, ya que una mala estimación de este valor, daría como consecuencia no encontrar el camino más corto o no encontrar el nodo meta (al sobrestimar o subestimar el valor). Como se mencionó anteriormente, el valor heurístico se puede estimar por medio de una función, llamada función heurística. Para que el valor heurístico tenga una buena estimación, es necesario hacer una buena definición de la función heurística. Esta función se determina dependiendo del problema que se vaya a tratar.

3.2.1.3 Definición de la función heurística y uso del valor heurístico en el grafo de patrones

Para la definición de la función heurística de este proyecto se utilizan dos factores (figura 19) con los cuales se trabaja en la etapa de diseño de la arquitectura:

- Se considera la priorización con la que llegan los atributos de calidad en cada iteración (Paso 3 de ADD).
- También se considera la influencia que tiene cada patrón sobre distintos atributo de calidad (Ponderación).

La priorización de los atributos de calidad en cada iteración es determinada, como ya se mencionó, en el paso 3 del método ADD. Esta priorización se realiza con base en la importancia que tiene cada atributo de calidad por iteración, en primer lugar para el usuario y en segundo lugar se considera el impacto sobre la arquitectura. Ésto da como resultado una priorización expresada por parejas (H,M), importancia para el usuario e impacto sobre la arquitectura respectivamente, donde la H representa alta prioridad, la M es mediana prioridad y la L representa baja prioridad, teniendo un total de nueve parejas o nueve combinaciones de prioridades (ver pagina 22).

Figura 19: Factores considerados en cada nodo-patrón del grafo para la definición de la función heurística

Priorización	Equivalencia Numérica
(H, H)	9
(H, M)	8
(H, L)	7
(M, H)	6
(M, M)	5
(M, L)	4
(L, H)	3
(L, M)	2
(L, L)	1

Tabla 4: Muestra la escala de equivalencia numérica de cada priorización

A cada prioridad se le asigna una equivalencia numérica, donde la pareja con más alto valor numérico es aquella que tiene mayor valor para el usuario y mayor impacto en la arquitectura, de esta forma se van asignando los valores de forma decreciente. En la tabla 4 se muestra esta correspondencia.

El otro factor a considerar, como ya se mencionó, es la influencia que tiene cada patrón sobre los atributos de calidad. Esta influencia puede ser positiva, negativa o nula, ésto quiere decir que al aplicar un patrón en el diseño de la arquitectura, éste puede satisfacer o perjudicar a los atributos, o simplemente no tener influencia sobre ellos. A esta influencia se le asigna una equivalencia numérica, a la cual se denomina ponderación, donde influir de manera positiva sobre un atributo de calidad tendrá una ponderación de 1, el perjudicar tendrá una ponderación de -1 y el no influir sobre el atributo tendrá una ponderación de 0. Cabe hacer notar que se decide eligir asignar una ponderación de 1 cuando la influencia es positiva porque sería muy complicado elegir otra escala numérica que determine cuál patrón satisface más o menos a un atributos de calidad dado (figura 20). Como se mencionó antes, sólo se toman en cuenta los atributos de calidad considerados por el SEI en el libro "Software Architecture in Practice" de Bass [10].

Figura 20: Ponderación correspondiente al patrón Observer

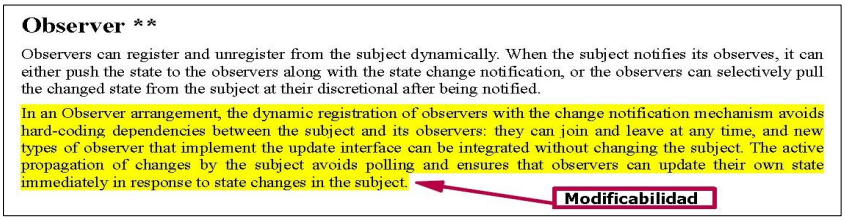


Figura 21: Fragmento de la descripción del patrón Observer del catálogo POSA 4 [17]

Para obtener las ponderaciones de la influencia de cada patrón sobre los atributos de calidad, se hizo un análisis de la descripción de los patrones contenidos en el catálogo POSA 4 [17], ya que ni en este catálogo ni en otros se expresa esta idea directamente. De esta forma se analizaron al rededor de 100 patrones de este catálogo. En la figura 21 se presenta un pequeño fragmento del contenido del catálogo POSA 4, correspondiente a la descripción del patrón Observer, donde se señala un párrafo marcado con amarillo de donde se determina que satisface positivamente al atributo de calidad de Modificabilidad.

El resultado del análisis del catálogo es almacenado en tablas, donde se encuentra la información de aproximadamente 100 patrones, como se muestra en la figura 22.

Tomando en cuenta estos dos factores, la función para determinar el valor heurístico de cada patrón es la suma de los producto de la ponderación de cada atributo de calidad por su correspondiente priorización (ésta última es con la que se inicia cada iteración del proceso de diseño). Dicha función se muestra en la fórmula 1.

Whole-Part	**Ponderación**
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Composite	**Ponderación**
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Half-Object Plus Protocol	**Ponderación**
Modificabilidad	0
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Master-Slave	**Ponderación**
Modificabilidad	0
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Replicated Component Group	**Ponderación**
Modificabilidad	-1
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Page Controller	**Ponderación**
Modificabilidad	1
Disponibilidad	1
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Figura 22: Porción de las tablas de ponderaciones, resultado del análisis del catálogo de patrones POSA 4 [17]

$$h_p = \sum_{i=0}^{5} (priorizaciónA_i * ponderaciónA_i)$$

Fórmula 1: Función heurística para determinar el valor h

Donde los sub-índices tienen los siguientes valores:

p : representa al nombre del patrón para el cual es el valor heurístico

i = 0 : representa al atributo de calidad Modificabilidad

i = 1 : representa al atributo de calidad Disponibilidad

i = 2 : representa al atributo de calidad Desempeño

i = 3 : representa al atributo de calidad Usabilidad

i = 4 : representa al atributo de calidad Seguridad

i = 5 : representa al atributo de calidad Facilidad de Pruebas

El valor de la priorización siempre va a ser variable en cada sistema y posiblemente en cada iteración, dependiendo de las necesidades de los involucrados y de la importancia que tenga cada atributo de calidad para ellos. El valor de la ponderación asignada a cada patrón siempre va a ser fija, ya que se hace la suposición de que el patrón siempre va a satisfacer, perjudicar o no influir de la misma forma en los atributos de calidad, independientemente de las necesidades.

Un ejemplo de la determinación del valor heurístico se muestra en la tabla 5, donde se realiza la operación correspondiente a el patrón Observer que da como resultado el valor heurístico mostrado en la figura 23. Se puede observar que en la tabla 5 existen prioridades con valor igual a cero, siendo que la tabla 4, mostrada anteriormente no incluía este valor, la explicación es que estos atributos de calidad no se están considerando para la iteración con la cual se está trabajando.

Priorización (variable)	**Ponderación (fija)**
Modificabilidad = 5	Modificabilidad = 1
Disponibilidad = 9	Disponibilidad = 0
Desempeño = 5	Desempeño = 0
Usabilidad = 0	Usabilidad = 0
Seguridad = 0	Seguridad = 0
Facilidad de Pruebas = 0	Facilidad de Pruebas = 1

Tabla 5: Determinación del valor heurístico del patrón Observer

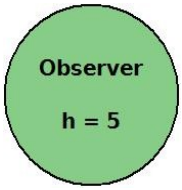


Figura 23: Resultado de aplicar la función heurística con los valor de la tabla 5

Finalmente para determinar que se ha llegado a un nodo meta, se considera que los atributos de calidad de esa iteración estén satisfechos. Ésto ocurre debido a que al encontrar un patrón que tenga un nivel positivo de satisfacción sobre un atributo, este atributos se considera satisfechos, por lo tanto para llegar al nodo meta se deben considerar satisfechos todos los atributos de la iteración. Estas consideraciones son hechas por el arquitecto.

3.2.1.4 Nodo Meta

Como se mencionó anteriormente, el nodo meta será aquel patrón que tenga un influencia positiva o satisfaga (ya sea parcial o totalmente) a el último atributo de calidad de una iteración. Ésto se refiere a que al ir recorriendo el grafo de patrones, se van satisfaciendo los atributos de la actual iteración, pero no todos al mismo tiempo, de ahí que se tenga que seguir recorriendo el grafo hasta haber satisfecho uno a uno los atributos. En el momento en que se consideran satisfechos todos los atributos, se termina la búsqueda del nodo meta, por lo tanto la condición es haber satisfecho todos los atributos de calidad con el uso de varios patrones, a través de un recorrido parcial del grafo.

Analizando un poco la forma en que se determina el nodo meta, se llega a la conclusión de que debido a que los valores heurísticos varían por iteración para cada categoría de atributos de calidad, se pueden tener distintos nodos metas dependiendo de los valores heurísticos que se obtenga por patrón, aún empezando por el mismo nodo de inicio (patrón).

3.2.1.5 Refinamiento adicional a la propuesta

El grafo de patrones cuenta con aristas, las cuales son necesidades que llevan en el recorrido de un patrón a otro, éstas se pueden considerar como un factor más de ayuda para elegir el camino sobre el grafo. Cada una de estas necesidades indican la satisfacción de uno o varios atributos de calidad, por ejemplo *Procesamiento de Datos* satisface Desempeño. Ésto se determinó en primer lugar por la necesidad misma y a qué atributo podría satisfacer, en segundo lugar se determinó por los atributos de calidad que satisfacen en común el grupo de patrones a los que lleva la necesidad. De esta forma se lograr un filtro de patrones, limitando la búsqueda dependiendo nuevamente de los atributos de calidad que se quieren satisfacer en cada iteración. Cabe mencionar

que al igual que para determinar la influencia de cada patrón sobre los atributos de calidad, para determinar la satisfacción o influencia de una necesidad sobre los mismos atributos se hizo nuevamente un análisis del catálogo POSA 4 [17], tomando en cuenta que éste muestra las necesidades para ir de un patrón a otro.

Por ejemplo si lo que se necesita satisfacer son los atributos de calidad de Modificabilidad y Usabilidad, el arquitecto selecciona las aristas o necesidades que cubran estos atributos de calidad (tabla de la izquierda de la figura 24).

Necesidad	Atributos de Calidad que Satisface
Procesamiento de datos	Desempeño
Evolución del sistema	Modificabilidad
Variación de interface del usuario	Usabilidad
Variación funcional	Modificabilidad, Usabilidad
Concurrencia remota	Modificabilidad, Desempeño

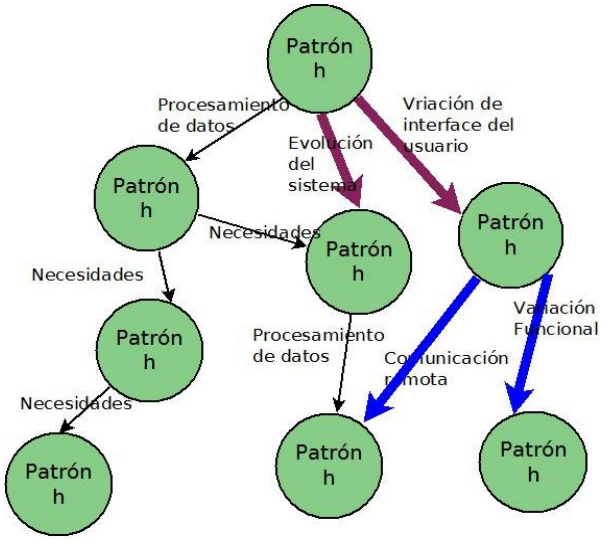


Figura 24: Refinamiento usando las necesidades que van de un patrón a otro

En la figura 24 se muestran las necesidades que van de un patrón a otro, tomando en cuenta que para el ejemplo se tienen que cubrir los atributos de calidad de Modificabilidad y Usabilidad, entonces se consideran solamente las necesidades que los cubran. En este caso para el nodo de inicio o raíz se tienen tres opciones, de las cuales sólo se consideran dos (flechas color rojo – "Evolución del sistema" y "Variación de interface del usuario") ya que la tercera no cubre ni Modificabilidad ni Usabilidad (Procesamiento de datos - Desempeño). Para el siguiente nivel, se vuelve a aplicar la misma dinámica, considerando las dos únicas opciones que se desprenden de este nodo (flechas de color azul).

Se considera también que para el caso en que no se tengan necesidades o aristas que satisfagan a los atributos de calidad con los cuales se está trabajando, la opción es tomar todas las aristas que se tengan. Ésto no significa que los patrones a los que llevan las necesidades no sean adecuados, ya que una necesidad en general, no sólo lleva a un patrón, sino que puede agrupar a varios de ellos. Es por esta razón que en alguno de ellos se podría encontrar, de forma individual, algún patrón que satisfaga a alguno de los atributos de calidad de la iteración. Como resultado de este refinamiento, se obtuvo un filtro que facilita y guía la búsqueda del nodo meta o satisfacción de los atributos de calidad por iteración.

3.2.2. Algoritmo de Búsqueda en Profundidad Adaptado

El algoritmo de búsqueda en profundidad adaptado que usa el Método MATDDS como motor de búsqueda necesita como entradas las prioridades hechas a los atributos de calidad para una iteración particular y el patrón donde se desea empezar la búsqueda sobre el grafo, este patrón en la primera iteración para un sistema que se inicia de cero debe ser el patrón Domain Model o nodo-patrón raíz. MATDDS aplica el algoritmo con las siguientes adaptaciones:

Se localiza en el grafo el nodo o patrón inicial. Una vez encontrado el inicio, se comienza a aplicar el algoritmo.

1. El nodo inicio se pone en la cola de ABIERTOS, con su correspondiente valor de h, este valor se obtiene con las prioridades correspondientes a los atributos de calidad y las ponderaciones correspondientes al patrón o nodo de inicio. Se inicializa la cola de CERRADOS como una cola vacía.
2. El nodo inicial se nombra MEJORNODO, sin sacarlo de la cola de ABIERTOS.
3. Se verifica que MEJORNODO sea el NODOMETA (todos los atributos estén satisfechos). Si MEJORNODO es un NODOMETA, se termina el la búsqueda, por lo tanto el algoritmo llega a su fin. De no ser MEJORNODO el NODOMETA entonces se saca de ABIERTOS se mete a CERRADOS. Cuando está en CERRADOS se verifica que no sea nodo hoja, de serlo se termina el algoritmo. De lo contrario se expande con sus sucesores o nodos hijo. Cada uno de estos nodos se verifican de la siguiente forma:
    1. Verificar que la arista que va de MEJORNODO a SUCESOR satisfaga a alguno de los atributos de calidad con los que se está trabajando en la actual iteración. De no ser así, se descarta el nodo SUCESOR y se continua con el siguiente SUCESOR. En caso de que sí satisfaga a alguno de los atributos de calidad, se continua con el siguiente paso.
    2. Verificar si SUCESOR está en CERRADOS. En caso afirmativo, llamar VIEJO al nodo de CERRADOS y no meter a ABIERTOS (no volver a usar el mismo nodo dos veces en una iteración).
    3. Si SUCESOR no estaba en CERRADOS, ponerlo en ABIERTOS y calcular su valor h.
4. Cuando los nodos hijos de MEJORNODO se han metido a ABIERTOS, como es una cola de prioridad, se acomodan los nodos de acuerdo a la mejor heurística. Para este proyecto la mejor heurística es aquella con valor más alto. Una vez acomodados se elige el o los nodos con la mejor heurística.
    1. Si solo existe un nodo con mejor heurística, a éste se le nombra MEJORNODO.
    2. De lo contrario, se toman los nodos o patrones con la mejor heurística y se selecciona uno de ellos. El nodo seleccionado será el que encabece la cola y se le nombrará MEJORNODO.

    Se regresa al paso 3.

El resultado requerido al aplicar este algoritmo es el camino que se recorrió para llegar al NODOMETA, contenido

en la cola de CERRADOS y el NODOMETA, obteniendo una lista de patrones.

3.2.3. Forma en que trabaja el Método MATDDS

Como se mencionó antes, el método MATDDS utiliza el algoritmo de búsqueda en profundidad adaptado. Este método es usado en todas las iteraciones durante la creación del diseño de la arquitectura, ya que está pensado para ser usado en el paso 4 de ADD. Para poder usar MATDDS se necesitan las tablas de ponderaciones (patrón – atributos de calidad, figura 22), las tablas de necesidades-aristas (necesidades – atributos de calidad) y la tabla 4 de prioridades. El arquitecto hace la selección de los escenarios de atributos de calidad con los que se va a trabajar en la iteración con la cual está trabajando (paso 3 de ADD) y provee una lista de las categorías de atributos de calidad con las cuales están asociados estos escenarios. El arquitecto también selecciona el patrón con el cuál se va a iniciar la búsqueda (recorrido del grafo). El patrón utilizado en la primera iteración al iniciar un diseño de arquitectura debe ser el llamado Domain Model, ya que este patrón es el más indicado para obtener la primer descomposición del sistema e iniciar desde aquí la búsqueda de más patrones. En las siguientes iteraciones se podrá continuar con alguno de los patrones que hayan resultado de la aplicación del MATDDS y que esté asociado con el elemento que se vaya a descomponer.

Los pasos que sigue el arquitecto para aplicar MATDDS se muestran a continuación. Considerando tener los elementos de inicio como son la lista de categorías de atributos de calidad con su correspondiente prioridad y el patrón de inicio, se comienza la aplicación del método:

1. Inicialmente la lista de atributos de calidad se consideran como no satisfechos.
2. De las necesidades-aristas salientes del patrón de inicio se seleccionan aquellas que influyen de forma positiva a por lo menos uno de los atributos de la lista que se desea satisfacer. Si ninguna de las necesidades-aristas influyera de manera positiva en alguno de los atributos, se seleccionan toda las necesidades-aristas.
3. Se toman los patrones a los que llevan las necesidades-aristas seleccionadas en el punto anterior, para determinar el valor heurístico h de cada uno de ellos utilizando la función heurística (Función 1). De estos patrones se seleccionara un patrón, teniendo las siguientes consideraciones.
    a) Si existe un patrón con el mayor valor de h, se considera como el mejor candidato para continuar al paso 4.
    b) Si existen varios patrones con el mismo valor máximo de h, el arquitecto es el encargado de seleccionar uno de los patrones propuesto para continuar al paso 4.
    c) Si el valor de h es cero ($h = 0$), se considera que no hay patrones que se puedan usar, por lo tanto la iteración termina.
4. Tomando en cuenta que el patrón con mayor valor heurístico h satisface por lo menos a uno de los

atributos, se analizan los resultados.

a) Si todos los atributos de calidad contenidos en la lista han sido satisfechos por el patrón (influencia positiva del patrón sobre los atributos), se decide terminar la iteración.

b) Los atributos que han sido satisfechos por el patrón se descartan de la lista, para continuar con otra iteración de MATDDS, ahora tomando en cuenta como patrón inicial el último utilizado (el de mayor valor h), regresando al paso 2.

Finalmente se obtiene, del uso de MATDDS en el diseño de la arquitectura, una lista de patrones que satisfacen los atributos de calidad de la actual iteración de ADD. Este resultado es obtenido de un recorrido parcial del grafo de patrones.

Considerando que este método realiza una serie de pasos y hace uso de un algoritmo, se pensó en realizar un programa que realice la función del método y de esta forma facilitar el uso del mismo.

3.3 Programa Asistente – Apoyo al Método

La finalidad de hacer la implementación de MATDDS es obtener un programa que sirva de apoyo al método y de esta forma facilitar su aplicación. Para lograrlo se consideró cada uno de los pasos que usa el método, así como el algoritmo en profundidad utilizado como motor de búsqueda, de la misma forma en que lo hace el método. También se consideran como entradas para el programa los mismos factores que usa el método, éstos son, atributos de calidad con su correspondiente prioridad por iteración y el patrón donde se va a iniciar la búsqueda (patrón inicial). Una diferencia del programa con respecto al método explicado anteriormente es que permite ver la información de los patrones al momento de tener que elegir uno de éstos, lo cual representa una ayuda para el arquitecto, completando el método original.

En la figura 25 se muestra la ventana que se presenta para ingresar las equivalencias numéricas de las priorizaciones, al igual que el patrón inicia. En esta figura se observa que los atributos de calidad de la iteración son Modificabilidad con priorización de nueve (9), Disponibilidad de ocho (8) y Facilidad de Pruebas de cuatro (4), utilizando como patrón inicial a Encapsulated Implementation.

El resultado de las entradas mostradas en la figura 25 se muestra en la figura 26 donde se puede observa la lista de patrones opcionales ofrecida por el Asistente-MATDDS y la selección de uno de ellos (Wrapper Facade), el cual al ser seleccionado presenta la información relacionada con él.

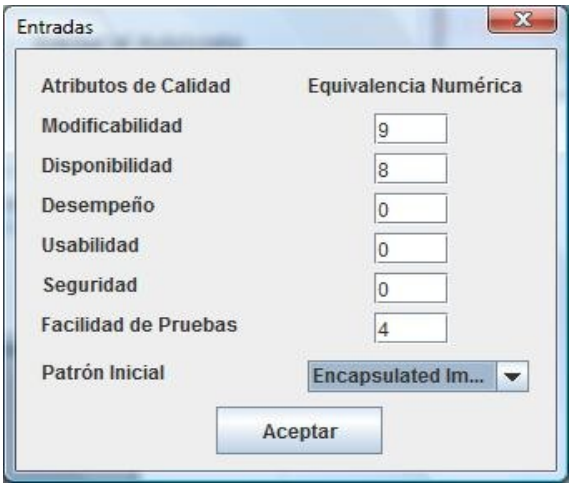


Figura 25: Ventana para ingresar las entradas utilizadas por el Asistente-MATDDS

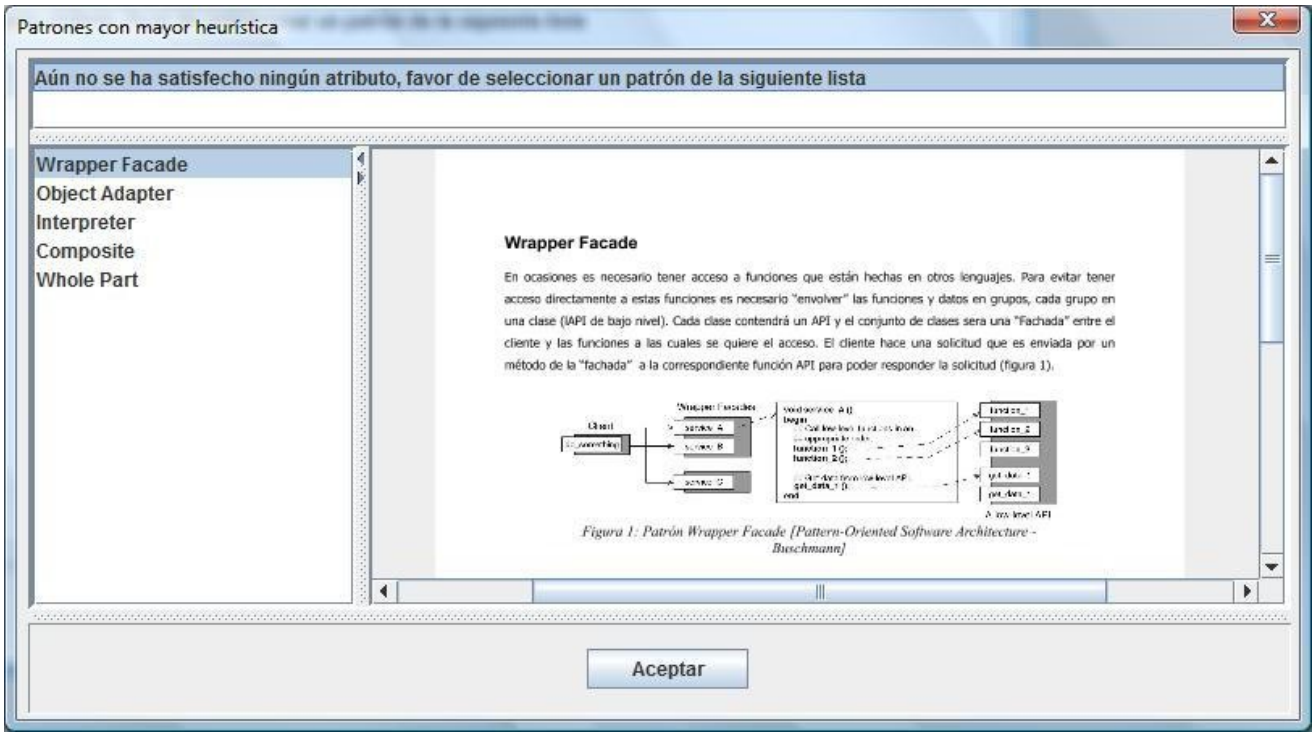


Figura 26: Ventana donde se muestran los patrones ofrecidos por el Asistente-MATDDS y la selección de uno de ellos con su correspondiente información

Finalmente en la figura 26 se muestra el resultado ofrecido por el Asistente-MATDDS, el cual presenta la lista de los atributos de calidad satisfechos y la lista de patrones con los cuales se pueden satisfacer dicho atributos de caldiad. Este programa es utilizado en la sección siguiente como asistente en la toma de decisiones de diseño sobre dos ejemplos.

Figura 27: Ventana donde se muestra el resultado ofrecido por el Asistente-MATDDS, donde se muestra la lista de los atributos de calidad satisfechos y la lista de patrones a utilizar

3.4 Resumen

Del análisis del catálogo POSA 4 [17] se obtuvo un grafo de patrones al cual se le puede aplicar un algoritmo de búsqueda (búsqueda en profundidad adaptado) que permita poder seleccionar patrones que influyan de manera positiva en un conjunto de categorías de atributos de calidad. Ésto dio como resultado un método, el cual se puede aplicar como parte auxiliar del método de diseño ADD, específicamente en el paso 4. Para poder aplicar este método es necesario contar con tablas de ponderación (nivel de satisfacción) de los patrones con respecto a los atributos de calidad y tablas de necesidades-aristas, resultado del análisis del catálogo POSA, así como de una tabla propuesta de prioridades de escenarios de atributos de calidad por iteración de ADD, la cual se determina al inicio del desarrollo del diseño de la arquitectura.

Finalmente se implemento el método de tal forma que se obtuvo un Asistente que facilita la aplicación del método, ya que cuenta con los mismos parámetros de entrada y búsqueda que el Método MATDDS y ayuda visual al momento de hacer una selección.

Capítulo 4

Evaluación de la Propuesta y Resultados Obtenidos

En este capítulo se presenta en primer lugar una comparación entre el ejemplo plasmado en el catálogo de patrones POSA 4 [17] y los resultados obtenidos al aplicar el Asistente-MATDDS al mismo ejemplo. A partir de esta comparación se hace un breve análisis.

En segundo lugar se hace una evaluación del mismo Asistente-MATDDS como auxiliar en la toma de decisiones de diseño dentro del método ADD, aplicado en el desarrollo del diseño de la arquitectura de un sistema de gestión de dispositivos en red. Finalmente se realiza un análisis general de la herramienta, en comparación con las herramientas plasmadas en el capítulo II de Estado del Arte.

Cabe mencionar que los siguientes ejemplos fueron realizados por una persona sin experiencia (estudiante) en el diseño de la arquitectura y más aún en el conocimiento de patrones y su aplicación en el diseño.

4.1 El Asistente-MATDDS aplicado a ejemplo POSA

Con el objetivo de tener un punto de comparación entre los resultados que ofrece el Asistente-MATDDS y las decisiones que pudiera tomar una persona con experiencia, se decide utilizar el ejemplo plasmado en el libro POSA 4 [17]. La idea fundamental de aplicar el Asistente en este ejemplo es la de comprobar que tanto difieren los resultados ofrecidos por el Asistente de las decisiones de diseño que se tomaron en el ejemplo del catálogo ya que éstas fueron hechas por una persona con experiencia. Con base en los datos obtenidos al realizar la comparación se podrá realizar un análisis de los resultados.

A continuación se presenta una descripción del sistema plasmado como ejemplo en el catálogo POSA 4, de tal

forma que sirva para realizar el diseño de la arquitectura de dicho sistema.

4.1.1. Descripción de un Sistema de Gestión del Proceso de Almacenamiento

Un sistema de gestión del proceso de almacenamiento es el encargado de administrar el uso del espacio en un almacén, dirige y optimiza las existencias de artículos basado en información en tiempo real. Generalmente este tipo de sistemas son usados por compañías donde se manejan grandes cantidades de artículos [17].

Los sistemas de gestión del proceso de almacenamiento pertenecen a la categoría de sistemas industriales automatizados y forman parte de una estructura de tres niveles, formando una pirámide llamada "pirámide de automatización", como se muestra en la figura 28. El nivel superior de la pirámide o nivel de operación es un sistema encargado de organizar, planificar y supervisar los procesos de negocio de un almacén (por ejemplo Enterprise Resource Planning – ERP, Manufacturing Execution – ME o Supply Chain Process Management - SCPM). El nivel intermedio es llamado nivel de control de procesos, sistema tomado como ejemplo en el libro, encargado de supervisar que las actividades organizadas y planificadas por el nivel superior sean ejecutas correctamente; de la misma forma se encarga de la administración de las tareas realizadas en el almacén tales como administración de productos, manejo de ordenes de recepción y envío, transportación de mercancía dentro del almacén. El nivel más bajo de la pirámide, es llamado nivel de entidad ya que es el encargado de ejecutar operaciones con entidades o elementos del mundo físico, ésto es, un sistema que representa y controla hardware utilizado como cintas transportadoras, grúas apiladoras o dispositivos de interacción humano-máquina.

Dentro del contexto del sistema de gestión del proceso de almacenamiento es importante resaltar que el sistema se encuentra en el nivel medio de la pirámide. Recibe órdenes del nivel de operaciones (nivel superior de la pirámide) y a él tiene que reportar sus actividades. También tiene comunicación con el nivel de entidad (nivel inferior de la pirámide) que le permite la interacción con elementos del mundo físico.

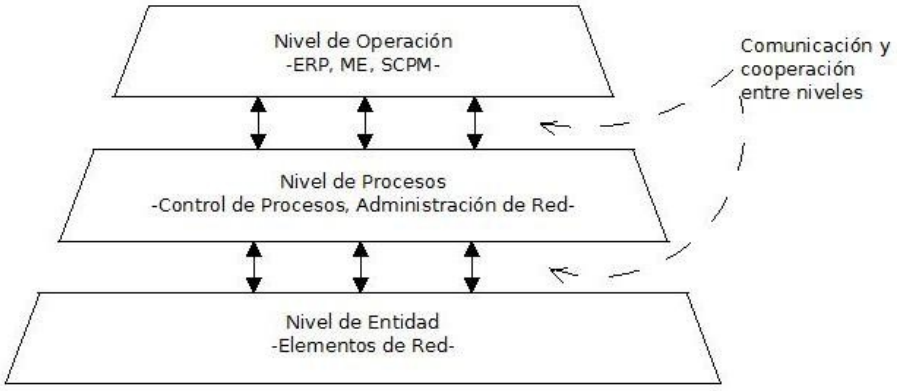


Figura 28: Pirámide de Automatización

Para entender de mejor manera las actividades que realiza el sistema del cual se desea hacer el diseño de la

arquitectura, a continuación se mencionan sus características principales:

- Manejo de existencia. Se hace manejo de existencia y de datos importantes, tanto para tipos de artículos como para cada uno de ellos de forma individual, cómo son nombre, descripción, disponibilidad de existencias y lugar de almacenamiento.
- Manejo de orden. El sistema realiza un manejo de los diferentes tipos de órdenes que recibe del nivel de operación, tales como órdenes de envío, de manufactura o producción en línea y anuncios de futuras recepciones y envíos.
- Envío. Se maneja todos los datos relacionados con el envío de artículos, tanto a nivel de tipo de artículo como de forma individual y es registrada toda la información.
- Recepción. Se maneja todos los datos de recepción de los artículos que llegan y se preparan antes de ser almacenados. Ésto se hace por medio de registros en el sistema, ya sea de datos del tipo de artículo, como de forma individual.
- Control de Flujo de Material. El sistema lleva un control del flujo de material. Envía las instrucciones concretas al respectivo hardware de automatización y recibe acuses de envío y mensajes de estatus en respuesta. Cualquier proceso en la ejecución de órdenes de transportación es reportado a la funcionalidad de *manejo de orden*.
- Administración de Topología. El sistema es el encargado de administrar la topología del almacén, así como proveer una representación de ésta para el *manejo de órdenes* y *control de flujo* de material.

Teniendo conocimiento de las características, es preciso mencionar los requerimientos no-funcionales del sistema de gestión de almacenamiento de donde se obtuvieron algunos de los atributos de calidad usados en el ejemplo. Cabe mencionar que entre paréntesis se encuentran las categorías de atributos de calidad asignadas a cada requerimiento no-funcional, las cuales son consideradas como datos de entrada en el Asistente.

- Distribución: El Sistema de Gestión del Proceso de Almacenamiento es esencialmente distribuido, ya que los diferentes clientes de éste se encuentran de esta forma, como son las estaciones de recolección o las máquinas elevadoras móviles, entre otros (Desempeño).
- Desempeño: El sistema debe cumplir con los plazos definidos y de manera eficiente, sin ninguna interrupción visible (Desempeño).
- Escalabilidad: Debe ser capaz de ajustarse en tamaño, como un pequeño o gran almacén (Modificabilidad).
- Disponibilidad: Se requiere una disponibilidad del 99.999%, ésto es, tener un máximo de 5 minutos de reposo por año (Disponibilidad).
- Persistencia: Es necesario mantener datos persistentes, consistentes y actualizados, como son la

topología del almacén, la disponibilidad de las existencias y todos los procesos de ordenes (Disponibilidad).

- Portabilidad: El sistema debe correr en distintos sistemas operativos, bases de datos y hardware (Modificabilidad y Usabilidad).
- Configuración Dinámica: El sistema debe permitir hace re-configuración y re-instalación en tiempo de ejecución (Modificabilidad).
- Interacción Humano-Computadora: Los usuario se deben poder comunicarse con el sistema a través de una gran variedad de interfaces. Por esta razón el sistema debe ser fácil de usar (Usabilidad).
- Integración de Componentes: El sistema debe soportar la integración de componentes como bases de datos o software (Modificabilidad).

4.1.2. Desarrollo del Sistema de Gestión del Proceso de Almacenamiento

Las características y los requerimientos no-funcionales presentados anteriormente son datos necesarios para entender de mejor manera el sistema plasmado en el libro POSA 4 [17], y sobre todo datos para realizar el diseño de la arquitectura de software. Estos datos son analizados para poder hacer uso del Asistente-MATDDS, tomando en cuenta los siguientes puntos:

1. Definir cuáles son los atributos de calidad que se desean satisfacer.
2. Determinar de qué patrón se va a iniciar la iteración, debido a que el resultado cambia dependiendo de éste.

Posteriormente del análisis se realiza una comparación entre los de los resultados obtenidos con el Asistente-MATDDS y los resultados obtenidos en el libro.

Cabe mencionar que el ejemplo no maneja iteraciones sino necesidades, por lo cual cada necesidad (plasmada en el libro) se toma como una iteración. Otro punto a resaltar es que debido a que no se usa una priorización de antemano en las necesidades, a cada atributo de calidad utilizado en éstas se le asigna una priorización de (H, H) alta prioridad para el usuario y alto impacto sobre la arquitectura, con equivalencia numérica de 9.

Primera necesidad:

La necesidad es poder organizar las funcionalidades del sistema en grupos coherentes, donde cada grupo pueda desarrollarse y modificarse independientemente. Esta necesidad se podría considerar como parte fundamental en el comienzo de todos los desarrollos de diseño de arquitecturas, ya que permite iniciar el sistema con base en las funcionalidades del sistema y hacer una estructuración inicial.

POSA – En el ejemplo se toma como primer patrón al llamado Domain Model, ya que con éste se representan las funcionalidades que se han identificado en el contexto del problema, básicamente representan los

requerimientos funcionales primarios. Posteriormente se aplica el patrón llamado Layers con la finalidad de agrupar las funcionalidades en diversas capas y poder tener bajo acoplamiento ayudando a la Modificabilidad del sistema, dividiendo éste en cinco capas, una capa de Presentación, otra de Procesos de Negocio, una de Objetos de Negocio, una de Infraestructura y finalmente una de Acceso a Datos. Cada una de estas capas agrupó funcionalidades con las cuales pueda desempeñar sus funciones, es decir ayudan a su comportamiento.

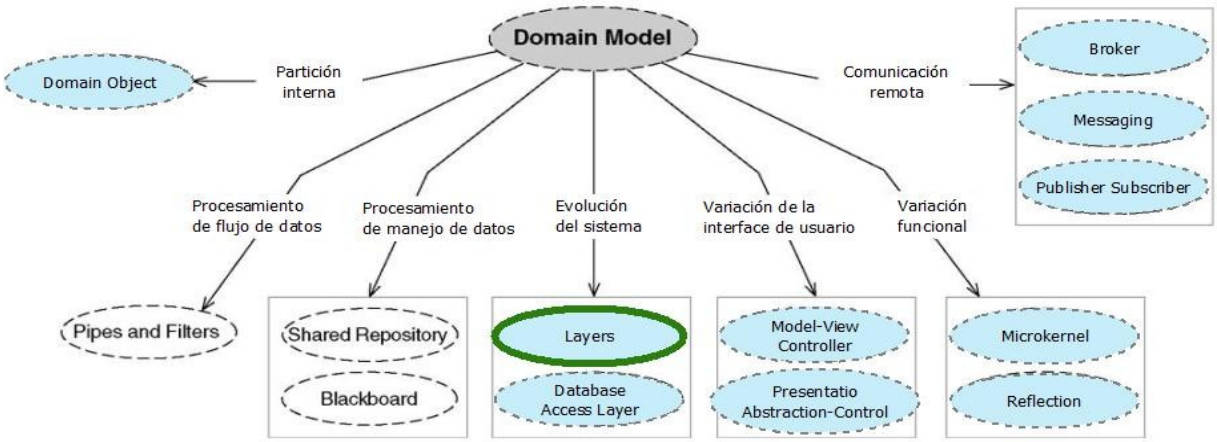


Figura 29: Mapa del patrón Domain Model y los patrones con los cuales está relacionado [17]

Figura 30: Ventana de entrada que muestra el Asistente

Asistente-MATDDS – Se determinan primero los datos de entrada, tomando como atributo de calidad a Modificabilidad y como patrón inicial al patrón Domain Model como se muestra en la figura 30; cabe mencionar

que a los atributos seleccionados (sea uno o varios atributos) en cada necesidad para este ejemplo se les asigna una ponderación de 9 y a los atributos no utilizados en cada necesidad se les asigna una de 0. De esta forma los atributos podrán ser las entradas para el Asistente. Un punto a resaltar es que Domain Model es el primer patrón considerado en el libro POSA 4, es decir, la raíz del grafo de patrones y debido a que lo que se quiere es iniciar el diseño de la arquitectura, éste patrón es el recomendado. Al aplicar el Asistente da una lista de diez patrones como se muestra en la figura 31, de un total de trece patrones relacionados con Domain Model (como se muestra en la figura 29). Ésto se debe a que la mayoría tiene una ponderación positiva con respecto a Modificabilidad, ayudando en cierto grado a este atributo. Dentro de esta lista se encuentra el patrón Layers, opción que es tomada en el libro. En la figura 29 se muestran todos los patrones relacionados con el patrón de inicio, Domain Model; en colo azul se muestran todas las opciones ofrecidas por el Asistente y con contorno verde se muestra la opción elegida en el libro.

Cabe mencionar que el orden en que se muestran las opciones no tiene que ver con el valor heurístico ni con ningún otro tipo de valor, ya que estos patrones se muestran debido a que tienen el mismo valor heurístico.

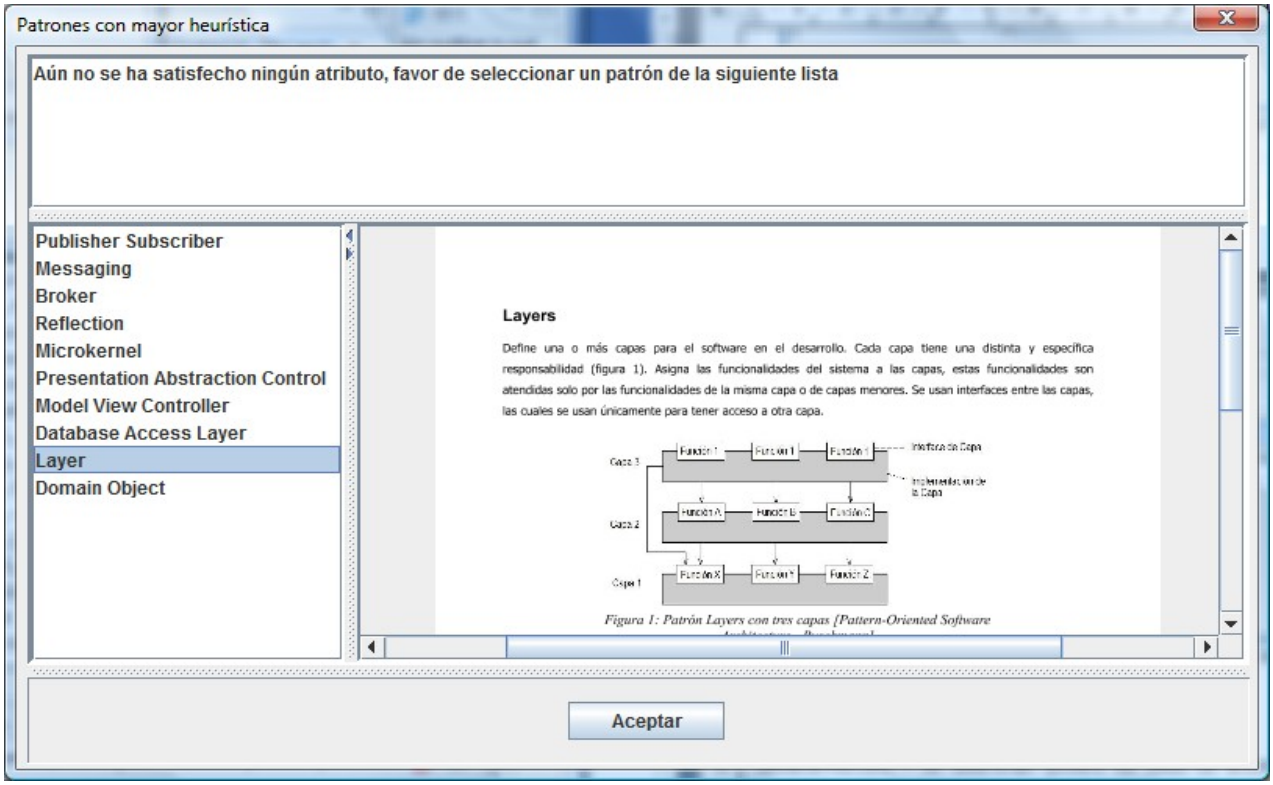


Figura 31: Opciones mostradas por el Asistente

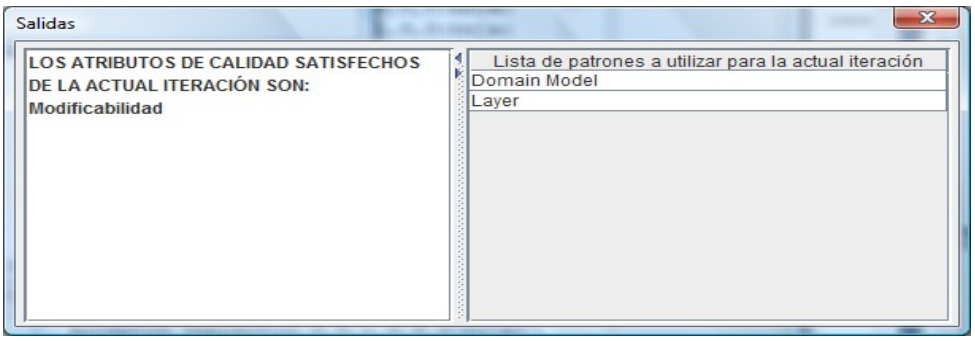


Figura 32: Al elegir el patrón Layers, el Asistente muestra el resultado obtenido

Segunda necesidad:

Poder refinar más las capas obtenidas, por medio de la creación de partes más pequeñas y poder separar y encapsular cada una de ellas, definiendo claramente sus responsabilidades. De esta forma proveer un correcto nivel granular para desarrollarse y evolucionar de forma independiente (Modificabilidad) cada una de las funcionalidades obtenidas. Esta necesidad está relacionada también con el atributo de Facilidad de Pruebas, ya que para desarrollarse es necesario hacer pruebas a cada una de las partes obtenidas.

POSA – Se decide aplicar el patrón Domain Object (creando varios objetos con responsabilidades bien definidas y separados).

Asistente-MATDDS – Al aplicar el Asistente se consideran como atributos de entrada a Modificabilidad y Facilidad de Pruebas y como patrón inicial Layers ya que es el patrón donde se quiere hacer el refinamiento de responsabilidades, obteniendo una lista de ocho patrones como resultado. Este resultado fue arrojado debido a que estos ocho patrones satisfacen positivamente a los dos atributos de calidad de entrada. En este caso se obtuvieron el total de patrones relacionados con Layers, dentro de los cuales se encontraba el patrón elegido en el libro, Domain Object. En la figura 33 se muestran los ocho patrones ofrecidos por el Asistente y con contorno verde se muestra el elegido en el libro.

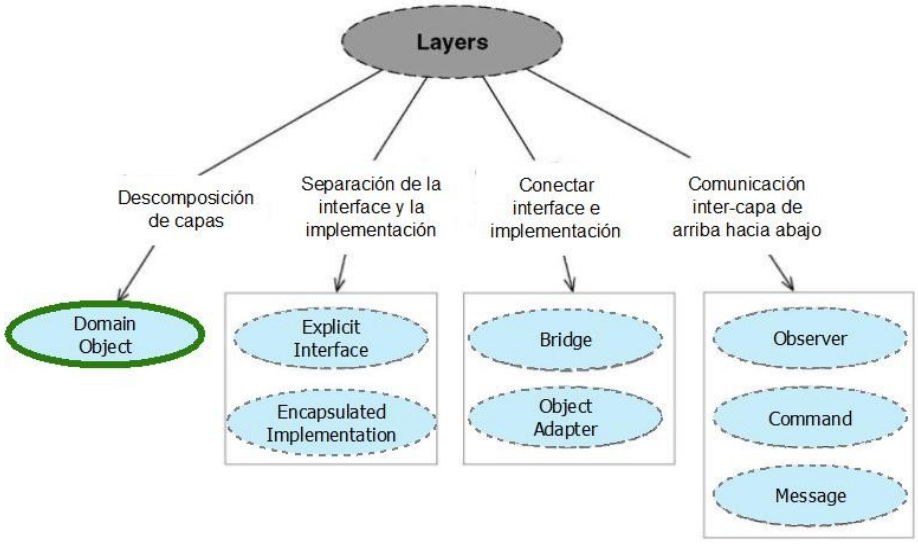


Figura 33: Mapa del patrón Layers y los patrones con los cuales está relacionado en el grafo de patrones POSA 4 [17]

Tercera necesidad:

Hasta este momento los objetos de dominio obtenidos al aplicar Domain Object están fuertemente acoplados, es decir, al hacer cambios en ciertos objetos pudieran afectar a otros objetos con los que estén relacionado. Es necesario asegurar que no haya dependencia en la implementación de los objetos (Modificabilidad y Facilidad de Pruebas), por lo tanto es necesario reducir el acoplamiento entre los objetos de dominio.

POSA – Se decide utilizar los patrones Explicit Interface y Encapsulated Implementation sobre los objetos obtenidos de Domain Object. Encapsulated Implementation encapsula la implementación de los objetos, mientras que Explicit Interface es la interface por medio de la cual se accede a los éstos, lo cual favorece el bajo acoplamiento, así como la Modificabilidad y la Facilidad de Pruebas sobre cada objeto.

Asistente-MATDDS – Al aplicar el Asistente con atributos de calidad de entrada de Modificabilidad y Facilidad de Pruebas y como patrón inicial Domain Object se obtiene una lista de cinco patrones de un total de diez que son con los que se relacionado a este patrón, como se muestra en la figura 34, donde los patrones con color azul representan los patrones ofrecidos por el Asistente y con contorno color verde los elegidos por el libro. Ésto reduce en un 50% las opciones para escoger, en este caso específico.

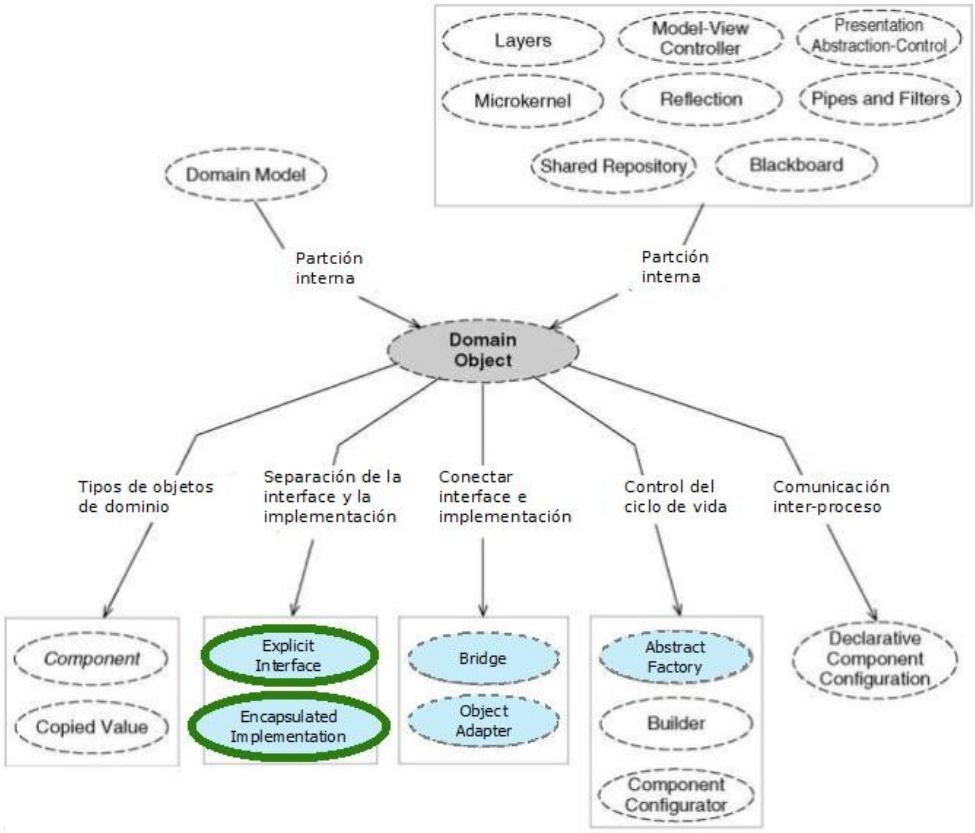


Figura 34: Mapa del patrón Domain Object y los patrones relacionados POSA 4 [17]

Cuarta necesidad:

Debido a que la naturaleza del sistema es distribuida, existen objectos (Domain Object) locales y remotos. El acceso a éstos es diferente en cada caso, ya que para los remotos es necesario hacer uso de la red. Esta diferencia no debe ser notada por los clientes al invocar métodos de los objetos, ni afectar con ésto el rendimiento (Desempeño), la Modificabilidad y la Disponibilidad del sistema.

POSA – Para atacar estas necesidades se propuso utilizar el patrón Broker que permite encontrar y acceder a los objectos distribuidos.

Asistente-MATDDS – Se decide utilizar como atributos de calidad a Modificabilidad, Desempeño y Disponibilidad y como patrón de inicio a Domain Model debido a que esta necesidad es para todo el sistema y sus funcionalidades, entre las capas y los objetos que se encuentran en ellas. Como resultado se obtuvo una lista de seis patrones (seis de trece, reducción al 55%) los cuales tienen nivel positivo de satisfacción sólo en Modificabilidad y Desempeño, faltando Disponibilidad. Por tal motivo es necesario elegir uno de los patrones ofrecidos de tal forma que el Asistente pueda seguir con el recorrido del grafo, decidiendo tomar el mismo que se toma en el libro, este es Broker ya que la finalidad es saber que tanto difieren los resultados del libro y los ofrecidos por el Asistente. Al

hacer la elección de éste patrón, se presenta otra lista con un total de cinco opciones para elegir (cinco de catorce, reducción del 75%), de donde se elige el patrón Component Configuration que permite hacer cambios en los componentes sin necesidad de apagar el sistema, permitiendo tener disponibilidad en el sistema. Esta diferencia, de usar dos patrones en lugar de uno para satisfacer los atributos, entre los resultados del ejemplo y los resultados obtenidos con el Asistente es debido a que las ponderaciones de los patrones del grafo se produjeron a partir de la interpretación de la definición de los patrones, la cual podría tener diferencia con la que se pensó al hacer uso de Broker en el libro POSA 4. En la figura 35 se muestra el camino recorrido por el Asistente para obtener dos patrones con un nivel positivo de satisfacción con respecto a los atributos de Modificabilidad, Desempeño y Disponibilidad, marcados con color azul los patrones opcionales y los elegidos mostrados con contorno de color verde.

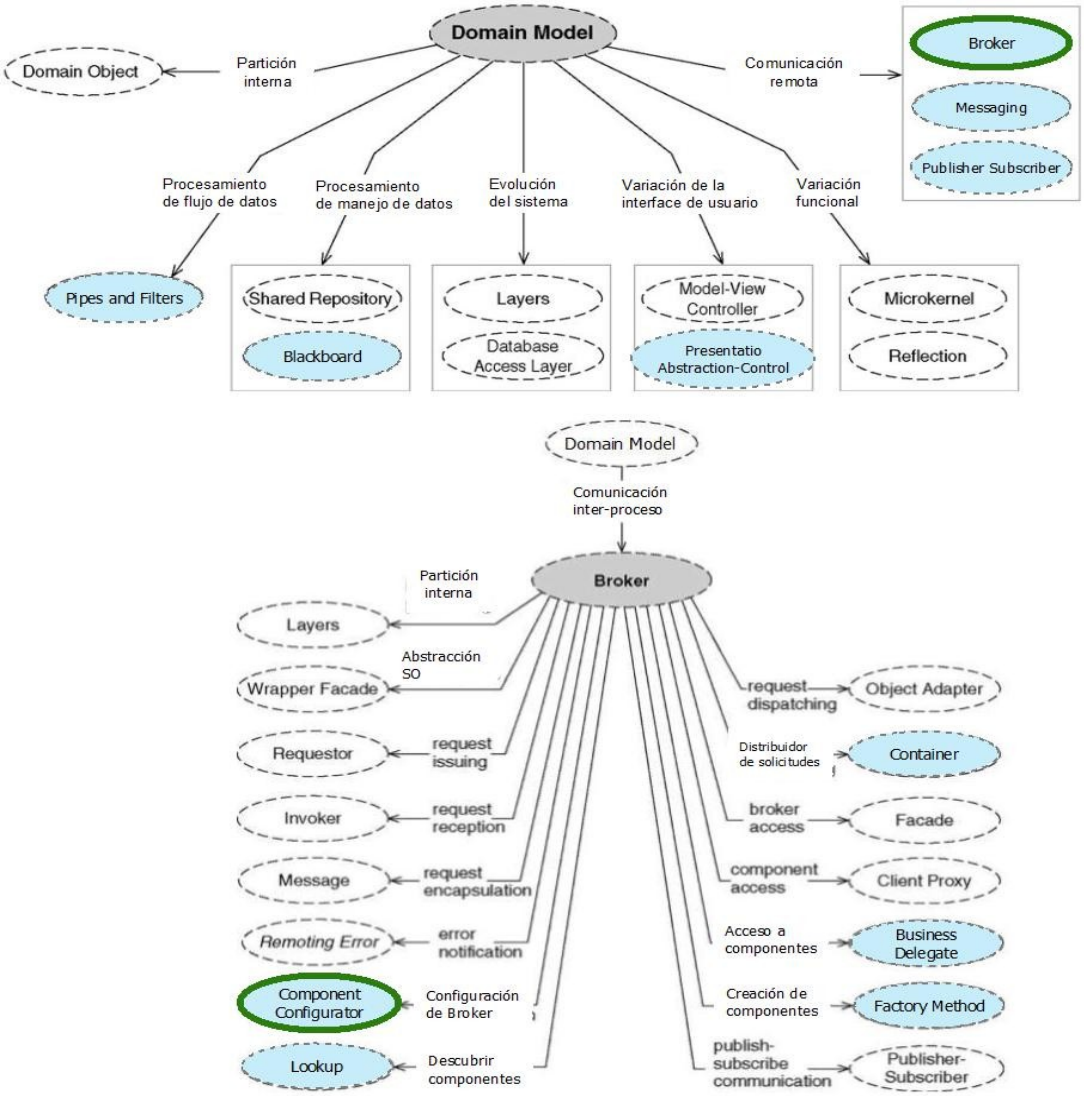


Figura 35: Camino recorrido por el Asistente para satisfacer los atributos de calidad

Quinta necesidad:

Se requiere dar acceso a las funcionalidades del sistema a usuarios (Usabilidad) u otros sistema, a los cuales se les llamará clientes. La información (almacenada en capas inferiores) que se presenta a los clientes debe estar actualizada de cambios realizados sin afectar el acoplamiento y la coherencia de las capas inferiores (Modificabilidad), por tal razón es necesario hacer sondeo continuo con lo cual se hace consumo de recursos (Desempeño) aún cuando no es necesario realizar alguna actualización, afectando al desempeño.

POSA – Para asegurar que los objetos de dominio (Domain Object) de la capa de Presentación estén actualizados sin hacer uso indebido de recursos, en el libro se propone utilizar el patrón Model View Controller (MVC) ya que minimiza el acoplamiento entre los objetos de la capa de Presentación y los de la capa de Proceso de Negocio asegurando con ello la cooperación y consistencia entre las capas. Cabe señalar que en el libro se indica que otra opción puede ser el patrón llamado Presentation Abstraction Control (PAC).

Asistente-MATDDS – Al hacer uso del Asistente se decide utilizar como patrón inicial a Domain Model, ya que la necesidad indica que se quiere dar acceso a las funcionalidades (Domain Model), por lo tanto se ve involucrado todo el sistema. Los atributos de calidad utilizados son Modificabilidad, Desempeño y Usabilidad, obteniendo como resultado directo (sin necesidad de elegir) al patrón Presentation Abstraction Control (PAC) (reducción del 97%). Ésto se debe a que en las ponderaciones del patrón MVC el atributo de Desempeño tiene valor cero (0), al contrario de PAC que tiene ponderación de 1, por lo tanto el Asistente se inclina por PAC en lugar de MVC.

En la figura 36 se muestra la relación del patrón Domain Model y los patrones involucrados para resolver la actual necesidad. Se observa que los dos patrones, con contorno de color verde MVC y de color amarillo PAC, están relacionados por la misma necesidad "Variación de la interface de usuario". Cabe señalar que el patrón PAC tiene relleno azul debido a que es el elegido por el Asistente.

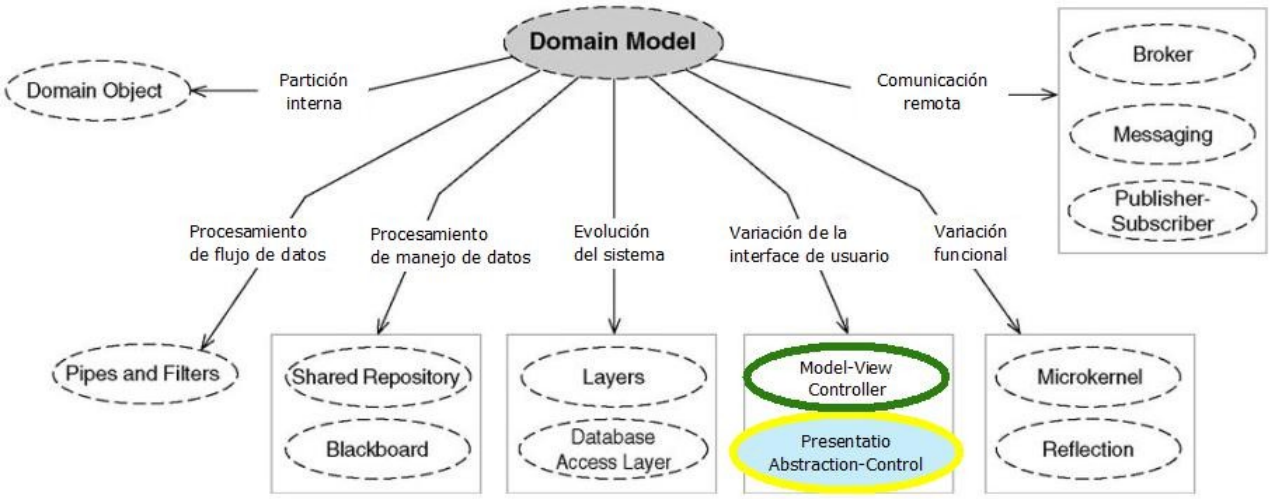


Figura 36: Mapa de Domain Model y su relación con el patrón Presentation Abstraction-Control

Sexta necesidad:

El acceso a los objetos es a través de una interface, obtenida ya sea por el patrón Explicit Interface o por la aplicación del patrón Broker (proxy remoto), pero ésto no es suficiente en un ambiente donde las funcionalidades se encuentran distribuidas físicamente en nodos distinto. A ésto se suma la necesidad de mantener el Desempeño y la escalabilidad (Modificabilidad) requeridos por el sistema, y permitir que sea tolerante a fallos (Disponibilidad).

POSA – Para satisfacer estos atributos en el libro se propone utilizar el patrón llamado Half-Object Plus Protocol. Este patrón divide la funcionalidad en grupos de Half-Object, donde cada objeto colabora en los espacios de memoria de cada cliente y cada uno de ellos implementa la funcionalidad requerida. La forma de comunicarse entre los objectos es por medio del protocolo del patrón Half-Object Plus Protocol.

Asistente-MATDDS – Se decide partir del patrón Encapsulated Implementation, ya que éste permite encapsular la implementación de los objetos antes de aplicar cualquier interface. Tomando como atributos de calidad por satisfacer a Desempeño, Modificabilidad y Disponibilidad se obtuvo un resultado directo, ésto es que no se tuvo que elegir de una lista de opciones, el patrón Half-Object Plus Protocol (el mismo que fue elegido en POSA 4 – figura 37), reduciendo las posibilidades a un 95%.

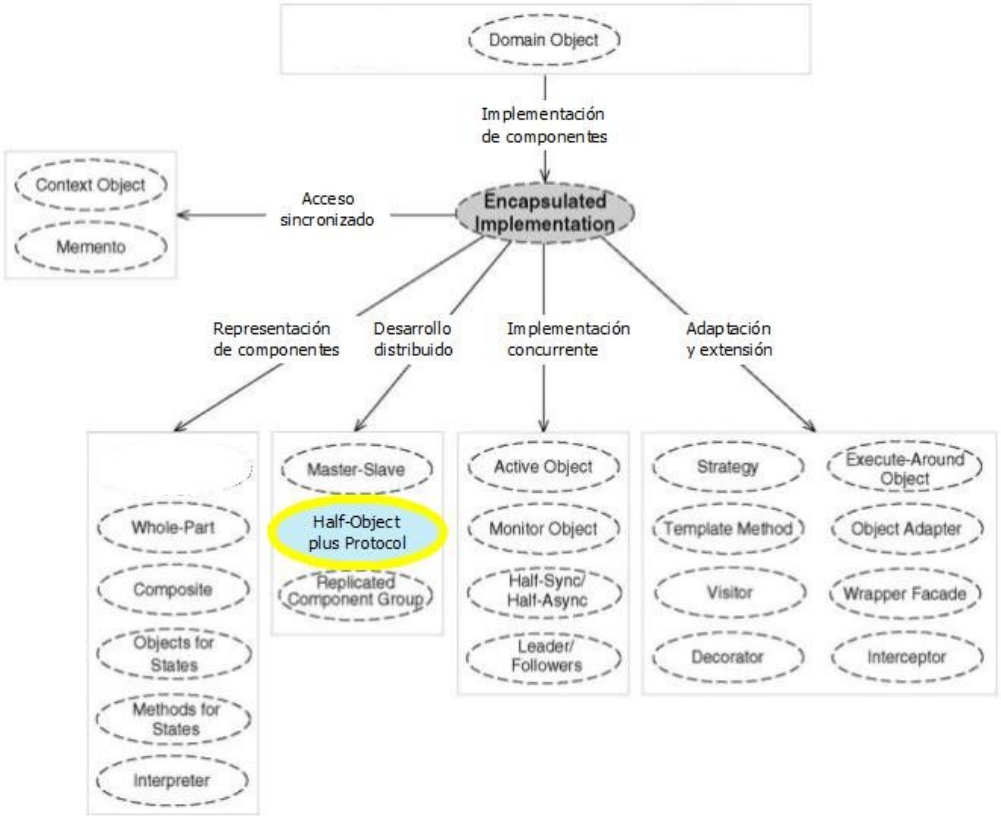


Figura 37: Del total de patrones relacionados con Encapsulated Implementation sólo se dio una opción, Halfa Object plus Protocol (contorno amarillo)

Séptima necesidad:

Utilizar el patrón Half-Object Plus Protocol en los objetos (Domain Object), permite tener Desempeño, escalabilidad (Modificabilidad) y ser tolerante a fallas (Disponibilidad) para peticiones hechas por un cliente. Pero en el caso de tener que responder a peticiones de forma simultanea de varios clientes (concurrencia) puede implicar cuellos de botella, afectando el Desempeño del sistema.

POSA - Para solucionar este problema se propone aplicar el patrón Active Object, que ayuda a la concurrencia de peticiones de clientes, por medio de una interface del componente para cada cliente a través de hilos y su planificación de la ejecución de las peticiones (figura 38).

Asistente-MATDDS – Al hacer uso del Asistente se utilizó como patrón inicial Encapsulated Implementation ya que se intenta partir de la implementación encapsulada de cada objeto, y Desempeño como atributo a satisfacer teniendo como resultado una lista de doce patrones de veintidós, donde se encontraba el patrón Active Object. Con este resultado se reducen las opciones en un 45% para este caso.

Octava necesidad:

El sistema realiza registro de datos a una bitácora por medio de objetos (Domain Object) especializados para esta labor (Logging Domain Object), pero ésto ocasiona muchas veces que el Desempeño del sistema baje si existe una gran cantidad de datos para registrar.

POSA – El libro propone que se utilice el patrón Leader/Follower, el cual utiliza un pool de hilos pre-establecidos (donde un hilo es el líder "leader" y los demás son los seguidores "followers") para evitar la sobre carga dinámica de hilos. De tal forma que un registro que llega a un Logging Domain Object es esperado por el "líder" (Leader) para ser leído, procesado y almacenado, cambiando su estatus a hilo de proceso y su lugar será ocupado por un hilo de los "seguidores" (follower) para esperar al nuevo registro (figura 38).

Asistente-MATDDS – Se tomó como patrón inicial a Encapsulated Implementation por las mismas razones que en las necesidades anteriores, debido a que es el patrón que permitió encapsular la implementación de los objetos, y al atributo de Desempeño como atributo a satisfacer, obteniendo la misma lista que en la necesidad anterior, donde también se encontró el patrón Leader/Follower; es aquí donde la ayuda proporcionada por el Asistente es de gran utilidad ya que se puede hacer la elección del patrón de acuerdo a esa información.

En la figura 38 se muestran las diferentes decisiones que se tomaron para satisfacer las tres necesidades anteriores, partiendo del patrón Encapsulated Implementation, donde se marcan con contorno verde los patrones elegidos en el catálogo POSA 4 en las necesidades 7 y 8.

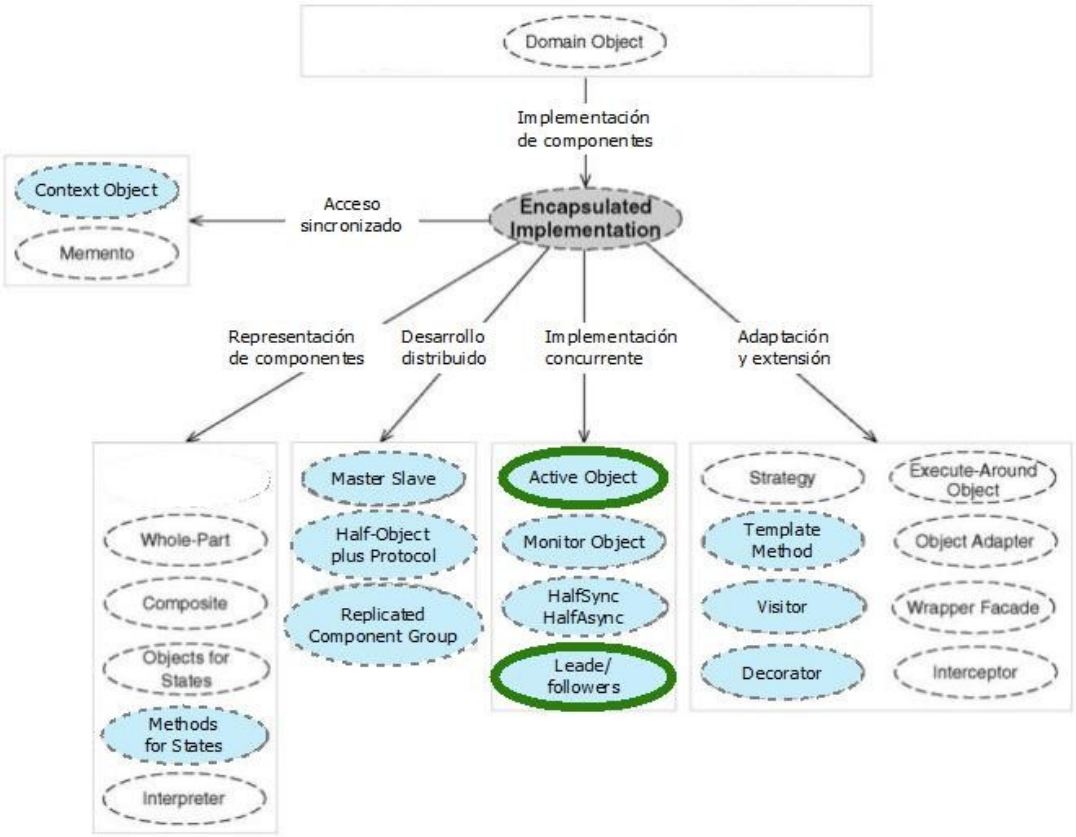


Figura 38: Mapa del patrón Encapsulated Implentation y los patrones relacionados con él [17]

Novena necesidad:

Existen datos manejados por el sistema que deben ser almacenados de forma persistente y cambiados de forma transaccional, tal como los *Envíos*, *Recepción*, *Administración de la Topología* (requerimientos funcionales primarios u objectos de negocio), etc., elementos obtenidos de Domain Model (Modelo de Dominio). Estos datos deben ser manejados de forma correcta, primero para ser obtenidos y luego almacenados de forma persistente, y de forma inversa existe la misma problemática; al igual que si se necesita hacer cambio de la base de datos, el problema podría ser peor.

POSA – Para solucionar este problema se aplica el patrón llamado Database Access Layer, que permite que los objetos de negocio (Domain Model) sean transformados a una forma persistente, adecuada para ser usados en una base de datos.

Asistente-MATDDS – Al aplicar el Asistente se tomó en cuenta como patrón inicial a Domain Model y como atributo de calidad a Modificabilidad, con lo cual se obtuvo una lista diez de patrones opcionales de un total de trece patrones, teniendo una reducción de 33%. En este caso la opción más conveniente es la misma que toma el libro ya que el patrón Database Access Layer es el que hace manejo de objetos para poder ser manejados en

una base de datos.

Décima necesidad:

Como se mencionó anteriormente, la Disponibilidad requerida del sistema debe ser del 99.999%, ésto es aplicable a todos los objectos (Domain Object) del sistema, no importando su funcionalidad. Se requiere también la configuración y re-implementación de los objetos en tiempo de ejecución (Modificabilidad).

POSA – Para resolver esta problemática se propuso utilizar el patrón llamado Component Configuration el cual soporta la carga dinámica, re-configuración y re-implementación en tiempo de ejecución de los objectos sin afectar la Disponibilidad de partes del mismo sistema que no están involucradas en esta actividad.

Asistente-MATDDS – Se toma como patrón inicial a Domain Object y atributos a satisfacer a Disponibilidad y Modificabilidad obteniendo una lista de dos patrones opcionales, donde se encontró el patrón utilizado por el libro, Component Configuration. En este caso se redujo en un 80% las posibilidades ofrecidas por el Asistente.

4.1.3. Análisis de comparación de resultados

Al concluir el desarrollo del sistema de gestión del proceso de almacenamiento, se prosiguió al análisis de la comparación de los resultados del libro POSA 4 [17] y los resultados obtenidos con el Asistente-MATDDS. Para realizar esta comparación se tomó en cuenta principalmente los patrones ofrecidos por el Asistente y la reducción que se tuvo en cuestión de opciones, tomando un porcentaje de reducción.

En la siguiente tabla se muestra la comparación entre las dos aplicaciones (libro vs MATDDS), la necesidad que se tiene, razón por la cual se define el o los atributos de calidad a satisfacer, patrón de inicio utilizado y razón por la cual se tomó como tal. Finalmente se muestra en la tabla el patrón o patrones resultante en cada caso (libro y aplicación del Asistente) y el porcentaje de reducción en caso de que haya habido alguna.

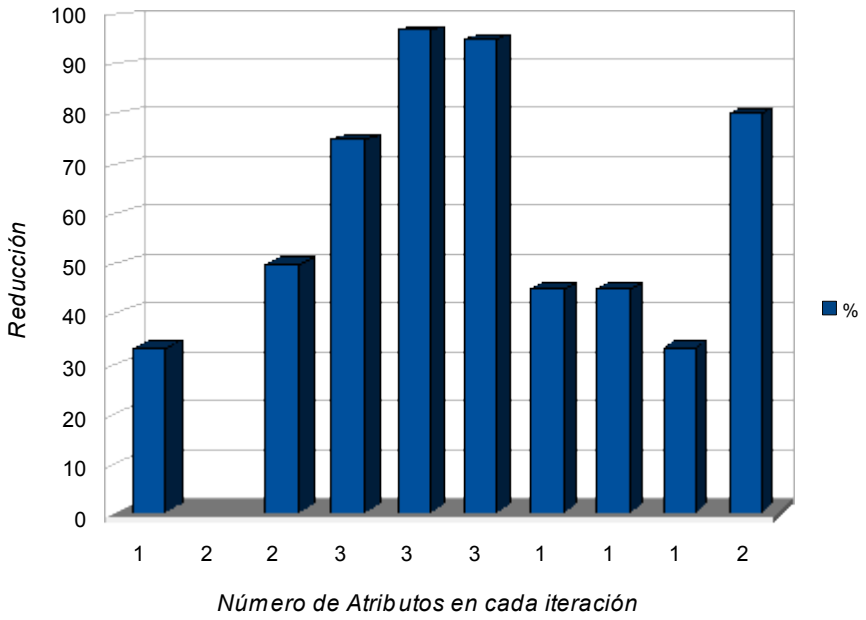
Necesidades del sistema	Atributos de Calidad	Patrón Inicial	Razón de utilizar el Patrón Inicial	Resultado POSA 4	Resultados MATDDS	Total de Patrones hijos	Porcentaje de Reducción	Comentarios
1.- Organizar las funcionalidades del sistema en grupos coherentes	Modificabilidad	Domain Object	Es considerado como el primer patrón que se debe tomar para iniciar un sistema debido a que representa las responsabilidades funcionales, a un nivel granular alto	Layers	Lista de diez patrones	Trece patrones hijos de Domain Model	33 %	La razón por la cual el Asistente ofrece el total de patrones relacionados con Domain Model es porque éstos tienen ponderación positiva con respecto al atributo de Modificabilidad. Dentro de esta lista se encontró al patrón Layers
2.- Encapsular y modularizar las funcionalidades por capa	Modificabilidad y Facilidad de Pruebas	Layers	La idea es trabajar dentro de cada capa creada (refinamiento de responsabilidades)	Domain Object	Lista de ocho patrones	Ocho patrones hijos de Layers	0%	Dentro de la lista ofrecida por el Asistente se encuentra Domain Object
3.- Asegurar que no hay dependencia en la implementación de los objetos, al igual que en la comunicación	Modificabilidad y Facilidad de Pruebas	Domain Object	La necesidad está directamente relacionada con los objetos del sistema, obtenidos de aplicar Domain Object	Explicit Interface y Encapsulated Implementation	Lista de cinco patrones	Diez patrones hijos de Domain Object	50%	En la lista de patrones ofrecida por el Asistente se encontraron los patrones Explicit Interface y Encapsulated Implementation
4.- Es necesario hacer uso de objetos locales y remotos sin que la diferencia sea notada por los clientes	Desempeño, Modificabilidad y Disponibilidad	Domain Model	La necesidad es de tipo estructural y afecta a todas las funcionalidades del sistema (funcionalidades obtenidas por el patrón Domain Model)	Broker	Primero una lista con seis patrones	Trece patrones hijos de Domain Model	55%	Dentro de la lista se encontró el patrón Broker utilizado en el libro. De acuerdo con el resultado del Asistente, los patrones ofrecidos no satisfacen a todos los atributos de calidad de entrada, por lo tanto se tuvo que elegir un patrón para obtener otra lista de patrones
					Después una lista con cinco patrones	Catorce patrones hijos de Broker	75%	La segunda lista fue a partir de Broker ya que es el patrón que se decidió utilizar. La razón por la cual se tuvo que elegir en dos listas es porque la ponderación de Broker en el Asistente no tiene un nivel positivo con respecto a Disponibilidad, por lo tanto es necesario buscar otro patrón que satisfaga este atributo
5.- Tener información actualizada para los clientes, sin hacer uso indebido de recursos	Modificabilidad Desempeño y Usabilidad	Domain Model	Se requiere trabajar con las funcionalidades del sistema, obtenidas utilizando éste patrón	Model-View-Cotroller (MVC)	Presentation Abstraction Control (PAC)	Trece patrones hijos de Domain Model	97%	En este caso el Asistente da como única opción a patrón PAC, reduciendo al 97% las posibilidades. En el libro se sugiere como opción alternativa a MVC éste mismo patrón PAC
6.- Las interfaces establecidas hasta el momento para la comunicación no son suficientes en un ambiente distribuido	Desempeño, Modificabilidad y Disponibilidad	Encapsulated Implementation	Se decide empezar por este patrón ya que es el que encapsula la implementación antes de aplicar cualquier interface	Half-Object Plus Protocol	Half-Object Plus Protocol	Veintidos patrones hijos	95%	El Asistente ofrece directamente a el patrón Half-Object Plus Protocol como única opción
7.- Responder a peticiones de forma simultanea a varios clientes	Desempeño	Encapsulated Implementation	Se decide empezar por este patrón ya que es el que encapsula la implementación antes de aplicar cualquier interface	Active Object	Lista de doce patrones	Veintidos patrones hijos	45%	En la lista obtenida se encontró el patrón Active Object
8.- Es necesario hacer registro de datos en una bitácora por medio de objetos especializados, in que el rendimiento se vea afectado	Desempeño	Encapsulated Implementation	Se decide empezar por este patrón ya que es el que encapsula la implementación antes de aplicar cualquier interface	Leader and Followers	Lista de doce patrones	Veintidos patrones hijos	45%	El Asistente ofrece la misma lista de doce patrones de la necesidad anterior. Es en estos casos donde la ayuda proporcionada por el Asistente es de gran utilidad, ya que por medio de ella es posible determinar cuál patrón es el más conveniente
9.- Persistir datos manejados por el sistema, para poder ser utilizados posteriormente	Modificabilidad	Domain Model	Se desea trabajar con varias funcionalidades definidas por el patrón Domain Model	Database Access Layer	Lista de diez patrones	Trece patrones hijos de Domain Model	33%	La lista obtenida es la misma que en la primera iteración, pero sigue siendo la necesidad la que define el patrón a elegir
10.- Se requiere una disponibilidad del sistema del 99.99%	Disponibilidad y Modificabilidad	Domain Object	La necesidad requerida afecta a todos los objetos sin importar su funcionalidad	Component Configuration	Lista de dos patrones	Diez patrones hijos	80%	En la lista ofrecida por el Asistente se encontró el patrón Component Configuration

Tabla 6: Comparación y análisis de resultados del libro POSA 4 [17] y el Asistente-MATDDS

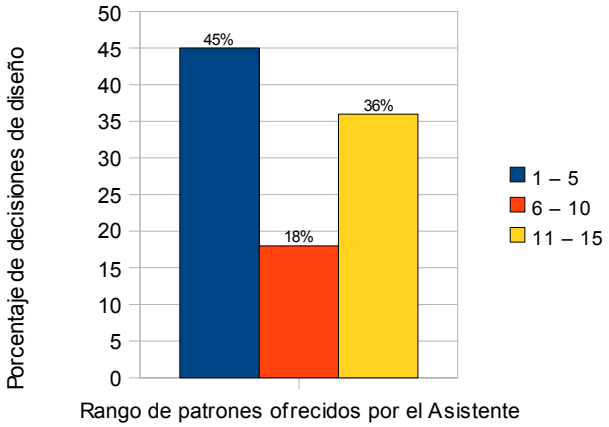
Como resultado del análisis de la tabla 6 de comparación se obtuvieron los siguientes puntos:

1. Se observa que generalmente cuando se tienen varios atributos de calidad como entrada para ser satisfechos, el Asistente proporciona una lista con un sub-conjunto menor del total de patrones relacionados con el patrón elegido como inicial. Esto se debe a que al tener varios atributos de calidad por satisfacer, habrá menos patrones que puedan satisfacer de forma simultanea a todos o a la mayoría de las categorías de atributos de calidad. No obstante, ésto no asegura que el tener mayor cantidad de patrones para satisfacer, ayude a tener la mejor opción, pero sí ayuda a limitar las opciones de patrones.
2. Los porcentajes de reducción en los casos donde se utilizaron uno o dos atributos de calidad fueron de un 0% al 50%. En los casos donde se utilizaron más de dos atributos el porcentaje fue mayor al 55%, con excepción del último caso donde se obtuvo una reducción del 80% utilizando dos atributos de calidad.
3. Se observó que no siempre se tiene una reducción de opciones al aplicar el Asistente, ya que de las diez necesidades utilizadas del ejemplo de POSA, solo en siete hubo reducción de opciones. De estas siete necesidades donde se presentó una reducción, el Asistente proporcionó un resultado único sólo en dos.

De los puntos anteriores se obtuvieron las siguientes gráficas. En la gráfica 1 se ve la relación entre el número de atributos de calidad con el porcentaje de reducción de opciones, proporcionado por el Asistente. Se puede observar lo mencionado anteriormente, cuando el número de atributos de calidad utilizados fue mayor el porcentaje de reducción de opciones fue también mayor.



Gráfica 1: Relación entre el número de atributos y el porcentaje de reducción



Gráfica 2: Rangos de patrones ofrecidos por el Asistente (bajo, mediano, alto)

Del total de once tomas de decisión de diseño (en diez necesidades del ejemplo), cuatro (36%) tuvieron un rango alto de opciones proporcionadas por el Asistente, es decir de once a quince patrones ofrecidos (color amarillo de la gráfica 2); dos (18%) tuvieron un rango medio, es decir de seis a diez (color naranja de la gráfica 2); y cinco (45%) un rango bajo, es decir de uno a cinco patrones opcionales (color azul de la gráfica 2). Con base en lo anterior, se considera que un rango bajo, es decir de 1 a 5 patrones opcionales ofrecidos por el Asistente, es un resultado factible para ser usado por arquitectos con poca experiencia, ya que no es necesario revisar demasiados patrones para poder tomar una decisión. Cabe mencionar nuevamente que sólo hubo dos casos en donde el Asistente ofreció un solo patrón.

Concluyendo, para este ejemplo en particular, la reducción de opciones no es mucha. No obstante se puede asegurar que las opciones ofrecidas por el Asistente tienen grado positivo de satisfacción con respecto a el o los atributos de calidad que se deseen cubrir o satisfacer. También se observó que es importante que el arquitecto tome buenas decisiones con respecto al patrón de inicio, ya que dependerá de ésto las buenas o malas opciones que ofrezca el Asistente-MATDDS. Para este ejemplo se logró por medio del análisis de las necesidades utilizadas.

Para tener otro punto de comparación y de análisis del Asistente-MATDDS se decide realizar otro ejemplo, esta vez dentro del contexto de ADD y utilizando los requerimientos de un sistema de gestión de redes de dispositivos.

4.2 Uso del Asistente-MATDDS dentro del contexto de ADD

Como se mencionó en la sección de Métodos de Diseño del capítulo II, ADD es un proceso de diseño de

arquitectura de software, conformado por ocho pasos. La aplicación de MATDDS se puede hacer dentro de este proceso, ya sea como método o como Asistente. Para ésto se toma en cuenta que en el paso 4 y en cada iteración, se presenta la necesidad de obtener una lista de patrones. Por lo tanto la aplicación de MATDDS se hace de forma natural, ya que éste proporciona una lista de patrones que influyen positivamente a las categorías de atributos de calidad que se tienen como entrada.

Para explicar con mayor detalle el uso de MATDDS en ADD es necesario mencionar que dentro del paso 4 existen seis sub-pasos que cubren desde la identificación de los problemas asociados a los drivers arquitectónicos, hasta la evaluación y resolución de inconsistencias en el diseño conceptual. En los sub-pasos 2 y 3 se crea una lista de patrones, esta elección es hecha con base en el conocimiento, habilidades y experiencia que tiene el arquitecto con respecto a los patrones y es aquí donde se hace uso de MATDDS, como apoyo a estos dos sub-pasos.

El curso de los siguiente sub-pasos, del 4 al 6, se realiza de forma normal, es decir, no se vuelve a hacer uso de MATDDS hasta la siguiente iteración del diseño de la arquitectura, como se ilustra en la figura 39. En esta figura se muestra, por medio de un diagrama de actividad, cada uno de los pasos de ADD, así como los sub-pasos contenidos en el paso 4 de ADD.

La interacción entre MATDDS como Asistente y ADD se ilustra a continuación por medio de un ejemplo.

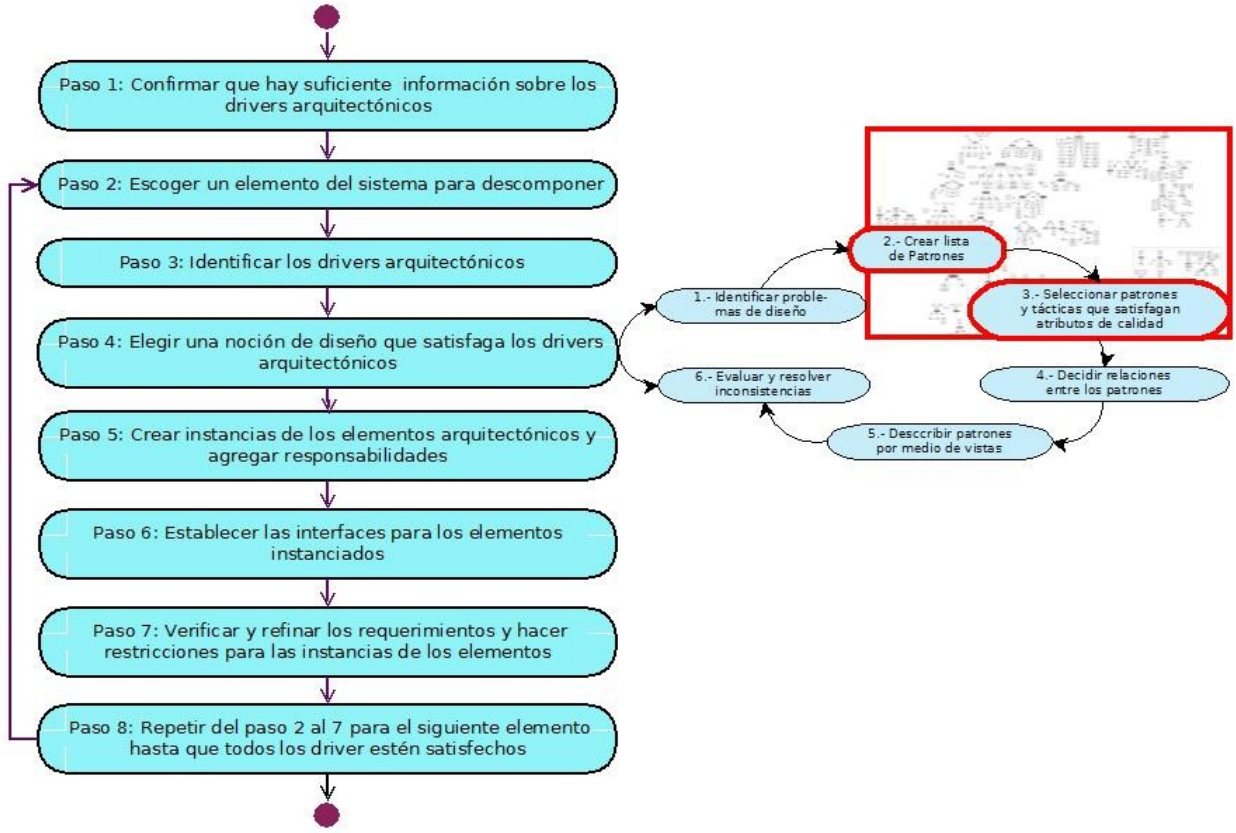


Figura 39: Uso de MATDDS como auxiliar dentro de los sub-pasos 2 y 3 del paso 4 de ADD

4.3 Aplicación del Asistente-MATDDS en un ejemplo

El problema que se tomó como ejemplo para la aplicación del Asistente-MATDDS fue presentado en la clase de Arquitectura y Calidad en el Desarrollo de Software, de la Maestría en Ciencias y Tecnologías de la Información de la Universidad Autónoma Metropolitana.

4.3.1. Descripción del problema

La organización X de telecomunicaciones, desea innovar en el mercado de los sistemas de administración de red, con una solución altamente configurable. Se desea realizar un sistema de gestión de redes de dispositivos heterogéneos que sea centralizado, ya que actualmente se cuenta con una infraestructura de dispositivos monitoreados a través de PCs por técnicos de la empresa. Hoy en día la organización se enfrenta a una serie de problemas ya que la red de dispositivos ha crecido y la infraestructura utilizada no garantiza, por ejemplo que las PCs estén monitoreando a todos los dispositivos, por lo tanto es posible que las alarmas emitidas por algún dispositivo no sean detectadas para corregir el problema. Otra problemática es que al agregar dispositivos se requiere de un esfuerzo considerable de configuración. Por estas razones es necesario contar con un sistema que permita resolver las problemáticas expuestas.

Los objetivos de negocio que dicha organización quiere alcanzar con el sistema de gestión de redes son los siguientes:

- Centralizar la administración de los dispositivos – La administración de la infraestructura de red de dispositivos de forma centralizada permitirá eliminar varios de los problemas a los cuales se enfrenta la organización actualmente.
- Reducción de tiempo de atención a fallas de dispositivos – Es indispensable reducir el tiempo actual de atención a fallas en dispositivos, con el fin de proporcionar una calidad de servicio aceptable, por tal motivo es necesario que un técnico pueda enterarse de la falla en un periodo no mayor a 5 minutos de haber ocurrido.
- Proporcionar un servicio confiable – El monitoreo no debe interrumpirse por periodos mayores a 5 minutos durante horas pico de operación, ésto es de 8:00 a 20:00 horas.
- Permitir crecer la infraestructura de la organización – Se considera que la infraestructura organizacional constará de alrededor de 100 dispositivos en menos de un año, pero que en los próximos 5 años, la infraestructura podría crecer hasta 1000 dispositivos.

En la figura 40 se representan una vista preliminar de la solución a través de un diagrama de implantación. Específicamente muestra el sistema como centralizado, la necesidad de tener notificación de fallas y de una estructura crecida de la organización (varios dispositivos).

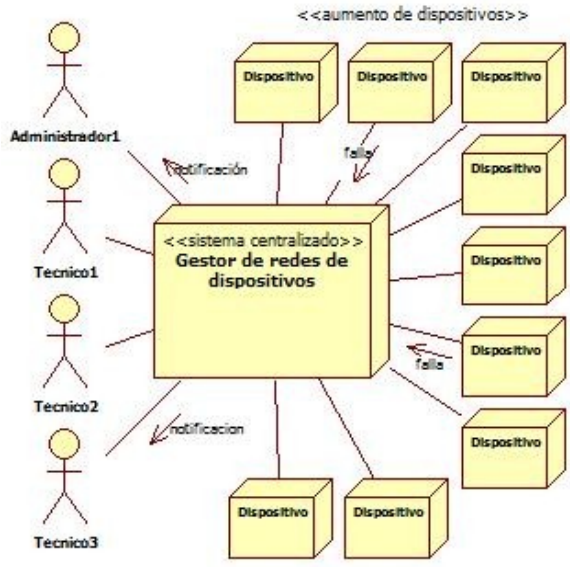


Figura 40: Vista preliminar de la solución

Para alcanzar los objetivos de negocio y resolver las problemáticas expresadas previamente, se identificaron las

siguientes necesidades del cliente y de la organización de desarrollo, mostradas en las tablas 7 y 8 respectivamente.

Necesidad	Características
Disponer de un sistema confiable y extensible que permita administrar la red de dispositivos de forma centralizada	Administrar la red de dispositivos soportando la introducción, retiro y configuración de los mismos
	Soportar la estructuración lógica de la red de dispositivos de acuerdo a un modelo jerárquico
Disponer de un mecanismo que permita identificar problemas en dispositivos, en un periodo óptimo de tiempo	Permitir realizar la representación visual de la red completa
	Recibir alarmas de dispositivos
	Detectar fallas de dispositivos
	Notificar sobre alarmas o fallas de dispositivos en la representación de red
	Notificar sobre alarmas de dispositivos mediante otros mecanismos como correo o SMS
Permitir un acceso controlado al sistema y a la red de dispositivos	Disponer de al menos dos niveles de usuario (técnico y administrador) soportando la administración de estos usuarios
	Guardar una bitácora de las acciones de los usuarios

Tabla 7: Necesidades del cliente

Nombre	Descripción
Garantizar prueba de la mayoría de los componentes	Con el fin de asegurar aspectos de calidad del sistema, se deberá asegurar que al menos un 95% de los componentes del sistema puedan ser probados de manera unitaria.
Generar componentes reutilizables	Con el fin de acortar el tiempo de desarrollos futuros, se buscará que algunos componentes sean genéricos

Tabla 8: Objetivos de negocio de la organización de desarrollo

Con base en lo descrito anteriormente, se determinaron los escenarios de atributos de calidad mostrados en la tabla 9, los cuales se utilizaran para obtener el diseño de la arquitectura. En esta tabla se muestra el identificador que se le asigna al escenario, una descripción de él, la prioridad que tiene el escenario para el cliente, el impacto en la arquitectura asignada por el grupo de desarrollo. Finalmente muestra la prioridad obtenida con base en la prioridad para el cliente y el impacto en la arquitectura expresada en tuplas y con su correspondiente equivalencia numérica (tabla 4).

ID	Descripción	Prioridad	Impacto sobre	Prioridad

		del Cliente	la Arquitectura	
DIS-01	Disponibilidad: No se debe interrumpir el monitoreo por más de 5 minutos, en horarios de operación	Alta	Alta	(H, H) = 9
MOD-01	Modificabilidad: Para incrementar la cantidad de dispositivos se necesita re-configurar, es decir cambiar parámetros del sistema, pero no código	Mediana	Mediana	(M, M) = 5
DES-01	Desempeño: Incorporar más dispositivos para monitoreo, sin afectar el tiempo de respuesta del sistema	Mediana	Alta	(M, H) = 6
USA-01	Usabilidad: Cuando haya un error en un dispositivo, se debe presentar al usuario (técnico o administrador) de tal manera que llame su atención	Alta	Baja	(H, B) = 7
DES-02	Desempeño: El tiempo que transcurre entre la detección de una falta en un dispositivo y su presentación en la pantalla del usuario no debe ser mayor a 5 minutos	Alta	Alta	(H, H) = 9
SEG-01	Seguridad: Para poder tener acceso al sistema, disponer de cuentas con las cuales poder identificar el tipo de usuario que se desea ingresar	Alta	Baja	(H, B) = 7
PRU-01	Facilidad de Pruebas: Al menos 95% de los componentes deben poder ser probados de forma unitaria	Alta	Media	(H, M) = 8

Tabla 9: Escenarios correspondientes a atributos de calidad

4.3.2. Desarrollo del diseño de la arquitectura con ADD y la intervención de MATDDS

1era iteración:

Se confirma que hay suficiente información sobre los atributos de calidad que se van a utilizar, dicha información se muestra en la tabla 9 (paso 1). Por ser la primera iteración se inicia trabajando con el sistema, visto como un solo elemento (paso 2), el cual se quiere descomponer en partes más pequeñas. Para lograr ésto, se desea utilizar en primer lugar patrones estructurales, los cuales pueden ser obtenidos con base en las necesidades primarias del sistema, o mejor dicho requerimientos funcionales primarios. La idea fundamental de este proyecto es el uso únicamente de atributos de calidad sin tomar en cuenta los demás drivers arquitectónicos (requerimientos funcionales primarios y restricciones). Por tal motivo se realizó un análisis de los requerimientos funcionales y de esta forma obtener atributos que influyan y ayuden a satisfacer a los requerimientos para empezar el diseño de la arquitectura. Haciendo el análisis se decidió utilizar únicamente un requerimiento, ya que éste es el que influye más sobre la estructura que podría tener el sistema (paso 3):

- Disponer de un sistema confiable y extensible que permita administrar la red de dispositivos de manera centralizada.

Considerando que el atributo de calidad con alguna influencia sobre este requerimiento es Modificabilidad, se decide hacer uso del Asistente con este atributo.

La aplicación del Asistente se inicia con el atributo anteriormente mencionado y tomando como patrón de inicio el nodo raíz del grafo de patrones, Domain Model, ya que es la primera iteración. Como resultado se obtuvo una lista con diez patrones estructurales tal como Model View Controller y Presentation Abstraction Control que ayudan a la variación interface de usuario, Microkernel utilizado cuando es necesario tener variación funcional en el sistema, Layers que ayuda a la evolución del sistema, decidiendo utilizar éste último con el cual se permitirá dividir el sistema en varias capas (paso 4). De esta forma se obtiene como resultado dos patrones, aplicando éstos de la siguiente forma (pasos 5 y 6).

Domain Model:

Este patrón es una estructura inicial para empezar el desarrollo de software, es un modelo del contexto funcional que debe tener el sistema y sirve para definir y delimitar las funcionalidades, así como determinar las abstracciones principales, como se muestra a continuación (primero funcionalidades y después las abstracciones obtenidas):

- Administrar la red de dispositivos:
    - Agregar dispositivos – El Administrador introduce un nuevo dispositivo a la red.
    - Retiro de dispositivos – El Administrador retira un dispositivo de la red.
    - Configuración de dispositivos – El Administrador configura un dispositivo registrado en la red de dispositivos del sistema.
- Detectar fallas de dispositivos:
    - Mostrar topología – El sistema muestra al Técnico la topología de la red.
    - Recibir alarmar de dispositivos – El sistema detecta cuando un dispositivo se alarma.
    - Detectar fallas en los dispositivos – El sistema detecta la falla de un dispositivo.
    - Notificación de alarma y falla – El sistema informa al Técnico de que existe una alarma o una falla en un dispositivo, mostrando tal información en pantalla.
- Monitoreo de dispositivos – El sistema monitorea continuamente los dispositivos.
- Acceso al sistema – El usuario (Administrador y Técnico) ingresa los datos solicitados (nombre, contraseña) para su ingreso al sistema (dependiendo del usuario, se dará acceso al sistema).
- Persistencia de datos – El sistema almacena los datos tales como estado de la topología, usuarios del sistema, etc. en una base de datos.

De estas funcionalidades se obtienen diez abstracciones con las cuales se empieza a representar la estructura del sistema, las cuales se relacionaron como se muestra en la figura 41 (Dispositivo, Configuración, Topología de

red, Alarma, Falla, Notificación, Estado de topología, Administrador, Técnico).

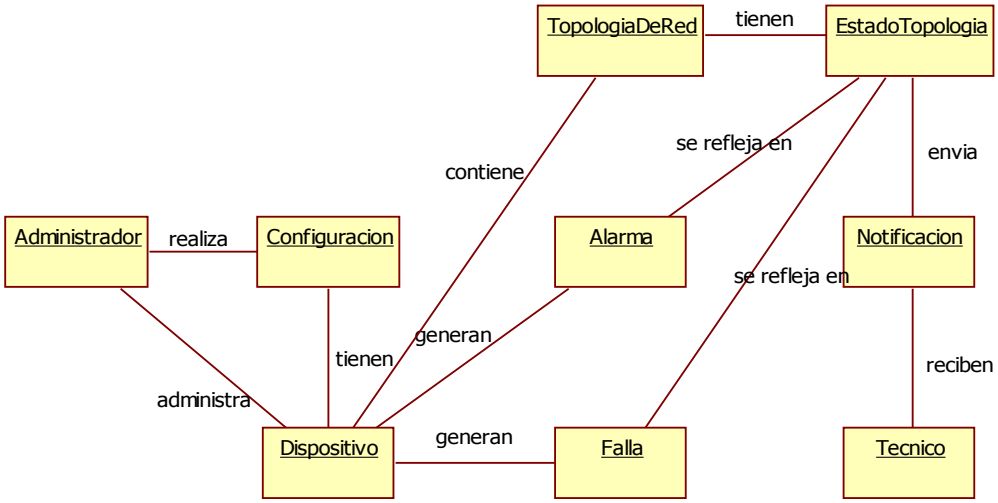


Figura 41: Abstracciones principales del sistema y las relaciones entre ellas

Layers:

Se decide utilizar cuatro capas que ayuden a distribuir las funcionalidades del sistema y con ésto se permita que el desarrollo y la evolución de forma independiente. En la figura 42 se muestra un diagrama con la distribución de las capas.

- Capa de Presentación – Encargada la interacción con los usuarios (técnicos, administrador o sistema externos).
- Capa de Lógica Negocio – Encargada de la lógica de negocio (únicamente), como detectar fallas de dispositivos.
- Capa de Manejo de Dispositivos – Encargada de la interacción con los dispositivos directamente.
- Capa de Datos – Encargada de realizar la persistencia de los elementos manejados en las demás capas. Tiene relación directa con la base de datos (estado de la topología, etc.).

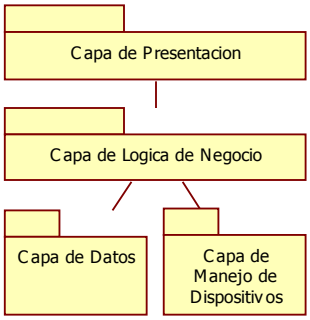


Figura 42: Distribución de cada capa obtenida de la aplicación del patrón Layers

Patrón	Escenario y atributo de calidad afectado	Forma de aplicación	Nivel de satisfacción sobre escenarios
Domain Model	No afecta a ningún escenario de los propuestos, pero es el patrón recomendado para iniciar el diseño de la arquitectura	Se analizan nueve funcionalidades de las cuales se obtienen once elementos que se relacionan entre si	No se satisface ningún escenario. Ayuda a la estructuración del sistema, dividiéndolo en funcionalidades, permitiendo tener bajo acoplamiento con respecto al comportamiento que va a tener
Layers	Ninguno, ya que se tomó en cuenta la necesidad de que el sistema sea centralizado y extensible	Se aplican cuatro capas - Presentación - Lógica de Negocio - Manejo de Dispositivos - Datos	El patrón tiene nivel positivo de satisfacción con respecto a Modificabilidad, permite que el sistema pueda evolucionar y ser extensible

Tabla 10: Patrones utilizados y forma de aplicarlos (instancia)

Se analiza que se haya cubierto en su totalidad la necesidad de estructurar el sistema y de que sea centralizado y extensible, decidiendo que ésto no ha sido así (paso 7), por lo tanto se realizar otra iteración.

2da Iteración:

Para esta iteración se decide utilizar la misma necesidad y requerimiento (centralizado y extensible) ya que no se considera que se han satisfecho estas necesidades y se quiere seguir estructurando el sistema. Ahora el patrón de inicio será Layer (paso 2) y el atributo de calidad seguirá siendo Modificabilidad (paso 3).

El resultado obtenido al aplicar el Asistente fue una lista de ocho patrones (totalidad de patrones relacionados con Layers) que tienen nivel de satisfacción positivo sobre Modificabilidad, de donde se decide utilizar el patrón Domain Object (paso 4). La razón para elegir este patrón es porque encapsula las funcionalidades, es decir, cada elemento u objeto se vuelve más específico, permitiendo que cada uno pueda evolucionar de forma independiente. Los objetos obtenidos se distribuyen en cada capa, para ir asignando las responsabilidades que

le corresponde a cada una de ellas (pasos 5 y 6).

Una vez instanciando el patrón se vuelve a analizar si se han cubierto las necesidades (paso 7). A pesar de tener un mayor nivel granular de las funcionalidades o responsabilidades de cada objeto, no se ha satisfecho del todo la necesidad de que sea escalable.

3era iteración:

Se decide aplicar nuevamente el Asistente tomando como patrón inicial a Domain Object (paso 2) y utilizando el mismo atributo de calidad de Modificabilidad (paso 3) ya que se quiere satisfacer aún la misma necesidad y requerimiento. El resultado de esta iteración fue una lista de siete patrones que ayudan a la Modificabilidad. Éstos se analizan para determinar cuál de ellos es el que más conviene al sistema y sus necesidades, dejando como opciones a usar Encapsulated Implementation que establece una interface para un grupo de componentes u objetos con la misma funcionalidad, Component Configurator que permite hacer re-configuración de componentes de forma dinámica y Explicit Interface que separa la interface de un componente de su implementación. Se decide utilizar el patrón Encapsulated Implementation que permite tener la implementación de los objetos de forma encapsulada ayudando a que el sistema sea extensible ya que no esta fuertemente acoplado (paso 4). La forma de aplicar estos patrones se muestra en la tabla 11.

Hasta este momento se tiene funcionalidades asignadas (Domain Model) a objetos con responsabilidades bien definidas (Domain Object) y con bajo acoplamiento (Encapsulated Implementation), estos objetos están distribuidos en capas (Layers), obteniendo una estructura inicial del sistema, como se muestra en la figura 43.

Patrón	**Escenario y atributo de calidad afectado**	**Forma de aplicación**	**Nivel de satisfacción sobre escenarios**
Domain Object	Ninguno, ya que se tomó en cuenta la necesidad de que el sistema sea centralizado y extensible	Se crean objetos con responsabilidades o funcionalidades más específicas, de esta forma se desacoplan más los objetos	No satisface ningún escenario, los patrones fueron utilizados para ayudar a la estructuración del sistema y a que sea extensible
Encapsulated Implementation		Se encapsula la implementación de las funcionalidades de un objeto	

Tabla 11: Patrones elegidos y forma de aplicarlos (instancias)

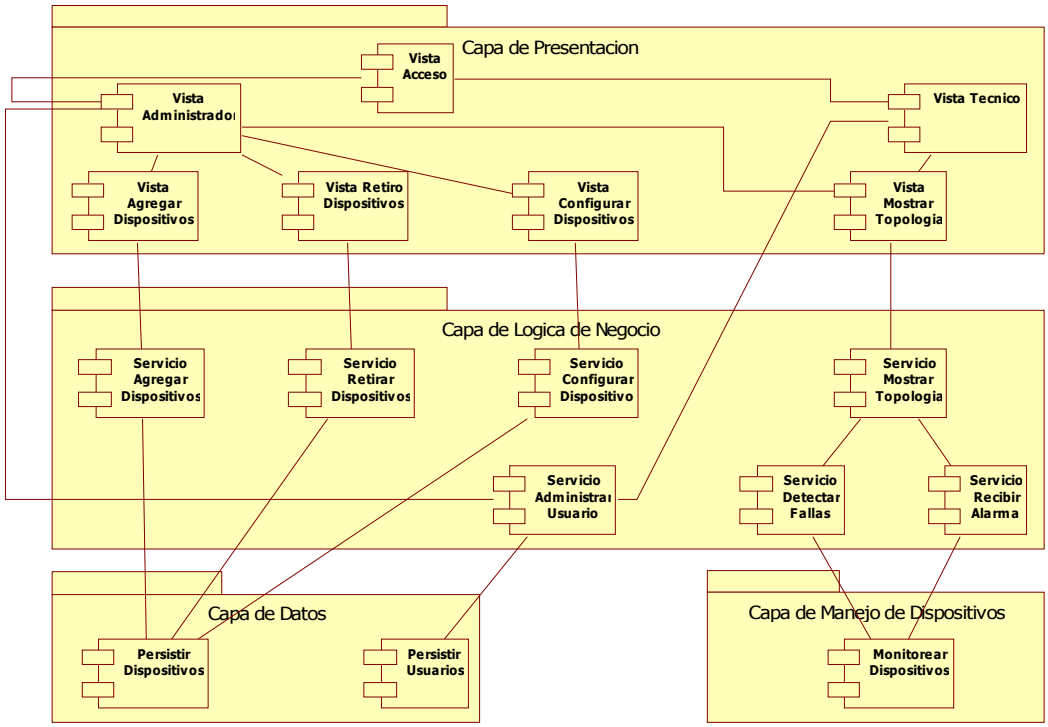


Figura 43: Distribución de los objetos de dominio (Domain Object) en las capas y sus relaciones

Elemento	Patrones Relacionado	Escenarios Asignados
Capa de Presentación	Layers	USA-01, SEG-01
Capa de Lógica de Negocio	Layers	MOD-01
Capa de Datos	Layers	PRU-01
Capa de Manejo de Dispositivos	Layers	MOD-01
Vista Acceso	Domain Object y Encapsulated Implementation	PRU-01, SEG-01
Vista Administrados	Domain Object y Encapsulated Implementation	PRU-01, USA-01
Vista Técnico	Domain Object y Encapsulated Implementation	PRU-01, USA-01
Vista Mostrar Topología	Domain Object y Encapsulated Implementation	PRU-01, USA-01
Servicio Mostrar Topología	Domain Object y Encapsulated Implementation	PRU-01, USA-01
Servicio Detectar Fallas	Domain Object y Encapsulated Implementation	PRU-01, USA-01
Servicio Recibir Alarma	Domain Object y Encapsulated Implementation	PRU-01, USA-01
Objetos en capa de Datos	Domain Object y Encapsulated Implementation	PRU-01
Monitoríar Dispositivos	Domain Object y Encapsulated Implementation	PRU-01, DES-02, DIS-01

Tabla 12: Asignación de escenario (responsabilidades) a elementos obtenidos

Después de ésto, se agregan las responsabilidades a los elementos resultantes como se muestra en la tabla 12 y

se refinan detalles de los elementos (pasos 5 y 6). Es a partir de este momento que se decide iniciar la siguiente iteración con los escenarios de atributos de calidad (escenarios que se tomarán de acuerdo a la ponderación más alta).Cabe mencionar que para determinar la asignación de responsabilidades de la tabla 12, se hizo un análisis de los escenarios y de esta forma poder asignarlos a los elementos.

4ta iteración:

En esta iteración se consideran los escenarios de atributos de calidad con mayor ponderación, DIS-01 y DES-02 ((H, H) = 9). Para determinar el patrón donde va a iniciar la iteración es necesario tomar en cuenta la tabla 12 y el contenido de los escenarios (análisis de ellos). En la tabla 12 se observa que estos escenarios fueron asignados al elemento Monitorear Dispositivos, teniendo como patrones relacionados a Domain Object y Encapsulated Implementation, decidiendo utilizar este último patrón como inicial ya que es el encargado de encapsular la implementación del objeto Monitorear Dispositivos. Por lo tanto se tienen ahora todos los datos de entrada necesarios para iniciar la aplicación del Asistente, patrón de inicio Encapsulated Implementation y atributos de calidad Disponibilidad y Desempeño.

Al aplicar el Asistente da como resultado tres patrones: Replicated Component Group, Half Object Plus Protocol y Master_Slave. Al revisar la información referente a estos patrones se decide utilizar Master- Slave, el cual ayuda a tener mejor Desempeño (reduce tiempo de respuesta) y soporta la tolerancia a fallos (no se interrumpe la actividad realizada por alguna falla de quien proporciona el servicio).

Master- Slave:

Se tendrá un conjunto de esclavos, los cuales estarán encargados del monitoreo de los dispositivos, de tal forma que a cada uno de los esclavos se les asignará un grupo de dispositivos, como se muestra en la figura , con lo cual se satisface en cierto grado Desempeño en el escenario DES-02. Para satisfacer en cierto grado el escenario DIS-01 (Disponibilidad) se utiliza el Maestro como organizador de los esclavos y en caso de que uno de ellos falle, organice a los restantes para realizar el monitoreo de los dispositivos del esclavo que fallo. En la tabla 13 se muestra la forma en que se aplica el patrón y el nivel de satisfacción que se obtuvo con respecto a los escenarios DIS-01 y DES-02.

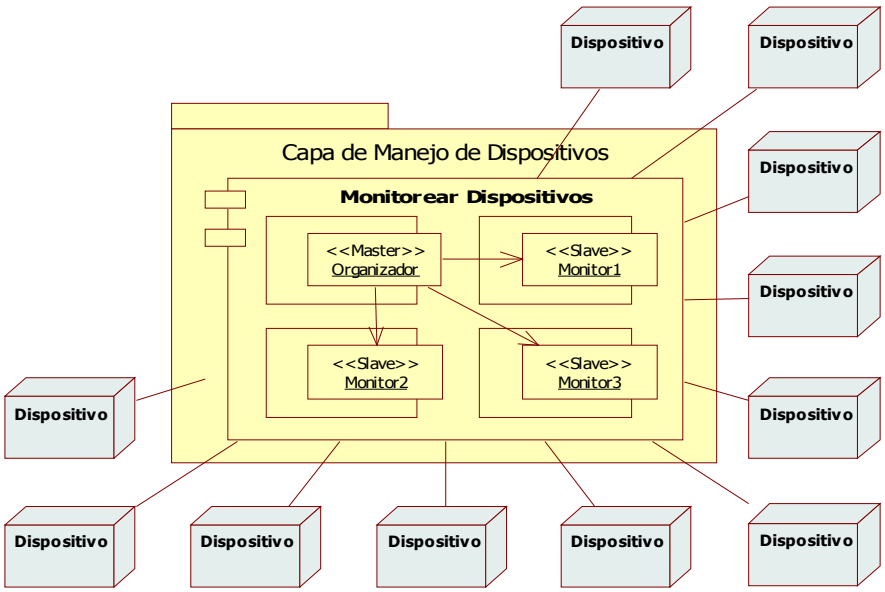


Figura 44: Forma en que se aplicó el patrón Master-Slave en el sistema

Patrón	Escenario y atributo de calidad afectado	Forma de aplicación	Nivel de satisfacción sobre escenarios
Master-Slave	DIS-01 Disponibilidad DES-02 Desempeño	Se aplica en el componente de Monitorear Dispositivos, teniendo al Maestro como el Organizador y a tres Esclavos para monitoreo los dispositivos. En caso de que uno de los Esclavos falle, cualquiera de los otros podrán ocupar su lugar	Se considera que satisface totalmente el escenario DIS-01 porque se mantendrá el monitoreo a pesar de que alguno de los Esclavos falle, no importando la hora. Satisface parcialmente al escenario DES-02 ya que sólo establece que se mantendrá el monitoreo pero no asegura que al detectar la falla se mandará la información al usuario

Tabla 13: Forma de aplicar el patrón Master-Slave y nivel de satisfacción de los escenarios

Driver\It	It1, It2, It3	It4	It5	It6	It7
DIS-01	No satisfecho	Totalmente			
MOD-01	No satisfecho				
DES-01	No satisfecho				
USA-01	No satisfecho				
DES-02	No satisfecho	Parcialmente			
SEG-01	No satisfecho				
PRU-01	No satisfecho				

Tabla 14: Nivel de satisfacción de los escenario en la 4ta iteración

Como se muestra en la tabla 14, DIS-01 se considera totalmente satisfecho, en cambio la satisfacción total del

escenario DES-02 se deja para alguna otra iteración y se continúa con el o los escenarios que tengan mayor prioridad después de DIS-01 y DES-02.

5ta iteración:

En este caso se tomó el escenario PRU-01 ((H, M) = 8) correspondiente al atributo de calidad de Facilidad de Pruebas. Debido a que este atributo afecta a todos los elementos obtenidos hasta el momento, Domain Object, se decide utilizar este patrón como inicial para hacer uso del Asistente. La lista resultantes de cinco patrones, en su mayoría (Abstract Factory, Object Adapter y Bridge), ofrece soluciones para situaciones complejas que no ayudarían a satisfacer el escenario. Otra de las opciones es el patrón Encapsulated Implementation que ya ha sido utilizado, quedando como única opción y más factible el patrón Explict Interface. Este patrón separa la declaración de interface de la implementación de un componente, permitiendo que sea por este medio la forma de comunicarse entre los diferentes componentes existentes en el sistema. Por lo tanto se aplica una interface a cada uno de los objetos obtenidos, después de la aplicación del patrón Encapsulated Implementation como se muestra en la figura 45. En la tabla 15 se muestra la forma en que se aplica el patrón Explicit Interace y el nivel de satisfacción con respecto al escenario PRU-01 correspondiente al atributo de calidad Facilidad de Pruebas, con el cual se está trabajando en la presente iteración.

Patrón	**Escenario y atributo de calidad afectado**	**Forma de aplicación**	**Nivel de satisfacción sobre escenarios**
Explicit Interface	PRU-01 Facilidad de Pruebas	A cada uno de los objetos obtenidos por el patrón Domain Object se les aplica el patrón Explicit Interface	Se considera que satisface totalmente el escenario PRU-01 debido a que no se tiene acceso directo a los objetos, por lo cual se pueden hacer pruebas con cada uno de ellos sin afectar a los demás objetos relacionados

Tabla 15: Forma de aplicar el patrón Explicit Interface y nivel de satisfacción de los escenarios

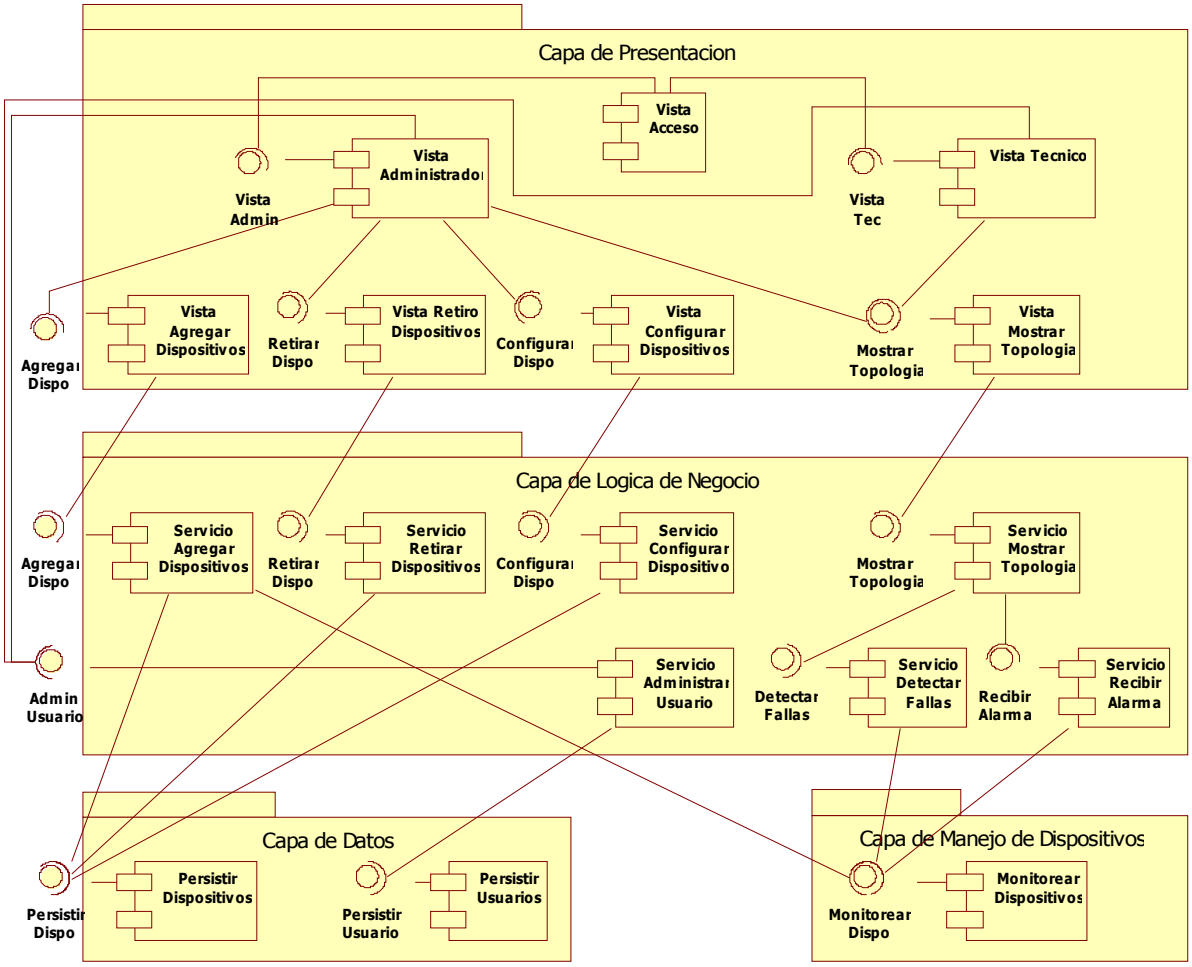


Figura 45: Aplicación del patrón Explicit Interface en todos los objetos del sistema

Driver\It	It1, It2, It3	It4	It5	It6	It7
DIS-01	No satisfecho	Totalmente			
MOD-01	No satisfecho				
DES-01	No satisfecho				
USA-01	No satisfecho				
DES-02	No satisfecho	Parcialmente			
SEG-01	No satisfecho				
PRU-01	No satisfecho		Totalmente		

Tabla 16: Nivel se satisfacción del escenario PRU-01 en la 5ta iteración

Como se muestra en la tabla 16, se considera totalmente satisfecho en la quinta iteración el escenario PRU-01,

teniendo hasta este momento DIS-01 y PRU-01 totalmente satisfecho, a diferencia de DES-02 que está parcialmente satisfecho.

6ta iteración:

En esta iteración se toman en cuenta los escenarios que aún no se han satisfechos y siguen en prioridad, estos son USA-01 (Usabilidad) y SEG-01 (Seguridad) con prioridad de valor 7. Estos escenarios tienen que ver directamente con los usuarios del sistema y están asignados a la capa de Presentación, a la Vista-Mostrar-Topología, a el Acceso-Usuarios, entre otros como lo muestra la tabla 12. Debido a que están involucradas varias partes del sistema para cada escenario, se decide utilizar primero uno de ellos y posteriormente el otro.

En primer lugar se toma el escenario correspondiente a Usabilidad, USA-01. Este escenario involucra en primera instancia al componente Vista-Mostrar-Topología que se encuentra en la capa de Presentación, pero la información que se presenta en la vista proviene de capa inferiores, por lo tanto también están involucradas. Por tal motivo se decide empezar por el patrón Domain Model que abarca la totalidad del sistema y que está relacionado con las funcionalidades del mismo.

El resultado obtenido al aplicar el Asistente fue una lista de cinco patrones: Reflection y Microkernel los cuales ayudan a tener variación funcional (no es necesario para el sistema), Presentation Abstraction Control, Model View Controller estos últimos que están enfocados a la variación de la interface del usuario y Shared Reposity que está enfocado a trabajar con grupos de datos compartidos en un repositorio central. De estos patrones los más convenientes fueron Presentation Abstraction Control y Model View Controller, decidiendo tomar el patrón Model View Controller debido a que permite tener un mecanismo de propagación de cambios, en este caso podría ser cambio de estado en la vista que se le ofrece al usuario.

Model View Controller:

El patrón se aplica en el componente de la capa de Lógica de Negocio y la capa de Presentación, de tal forma que la funcionalidad de Vista se asigna a el componente Vista-Mostrar-Topología, el Modelo se asigna al componente Servicio-Mostrar-Topología y la función de controladores se asigna a los componentes Servicio-Detectar-Fallas y Servicio-Recibir-Alarma como se muestra en la figura 46. Es necesario mencionar que el propósito del patrón Model-View-Controller es que el controlador modifique al modelo, por eso el controlador no está en la capa de Presentación sino en la capa de Lógica de Negocios.

Controllers – En cuanto se detecta una falla o se recibe una alarma se envía en dato a Servicio-Mostrar-Topología.

Model – El modelo cambia dependiendo del tipo de aviso que se haya mandado y también dependiendo de este cambio se envía la información a Vista-Mostrar-Topología. El modelo tiene un estado normal sin aviso y es cuando la topología se mantiene estable.

View – Una vez recibida la información del cambio del modelo, se cambia la interface y se muestra en las correspondientes vistas (Vista Administrador y Vista Técnico).

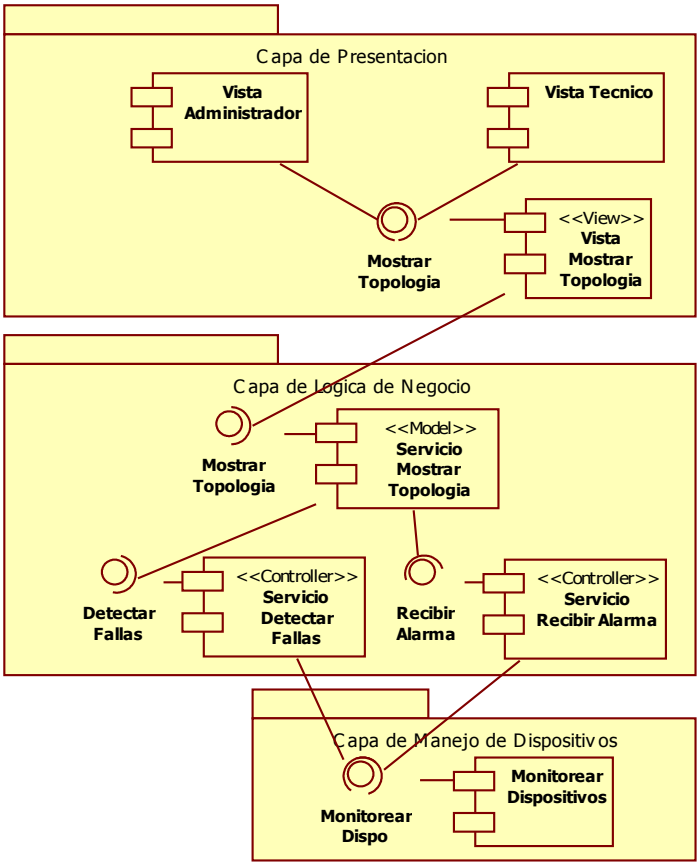


Figura 46: Patrón Model-View-Controller aplicado al sistema para satisfacer el escenario USA-01

Se podría considerar que el escenario es parcialmente satisfecho debido a que sólo se asegura que el usuario (Técnico y Administrador) recibe en su correspondiente vista el error, no se asegura que se presente de tal forma que llame su atención. A pesar de ésto se considera totalmente satisfecho ya que la alarma va a estar implementada en la vista (web) y dependerá del diseño de la misma sin involucrar el diseño de la arquitectura de software.

Ahora se desea satisfacer el escenario SEG-01 correspondiente al atributo de calidad de Seguridad. Este escenario tiene que ver con la seguridad que se necesita para que los usuarios puedan tener acceso al sistema. En este caso sólo están involucrados los objetos (Domain Objects) que tienen que ver con las vistas del Administrador y del Técnico (Usuarios), por lo tanto también las interfaces con las cuales se tiene acceso a estos objetos, Explicit Interface, decidiendo tomar este patrón como inicial en la aplicación del Asistente.

Al aplicar el Asistente se obtiene una lista con cuatro opciones: Data Transfer Object que permite transferir datos a través de la red, Context Object el cual permite transferir información de un componente sin alterar el mismo, Memento que permite conservar el estado original o varias versiones de un componente en un objeto y Authorization que permite revisar si determinado cliente tiene acceso al uso de cierta funcionalidad del sistema, decidiendo tomar éste último, ya que es el patrón que cubre más las necesidades del sistema.

Authorization:

El patrón se utiliza en el componente llamado Vista-Acceso, ya que es éste el que proporciona la vista de entrada a los usuarios del sistema, ya sea al técnico o al administrador. Tal como lo dice la documentación del patrón, mostrada en el libro POSA 4 [17] se utilizan dos métodos. Cada método sólo puede ser usado por un tipo de usuario, ya sea el Administrador o el Técnico, verificando qué tipo de usuario es el que desea ingresar. En caso de que el usuario que desee utilizar un método no esté autorizado, simplemente se le niega el uso. Cada uno de estos métodos lo único que hace es una llamada a la vista correspondiente al tipo de usuario con el cual esté relacionado, como se muestra en la figura 47. Como resulta de la aplicación de este patrón, se considera satisfecho totalmente el escenario SEG-01.

En la tabla 17 se muestra la forma en que se aplican los patrones Model-View Controller y Authorization y el nivel de satisfacción que brindan en los escenarios USA-01 y SEG-01 correspondientes a Usabilidad y Seguridad respectivamente.

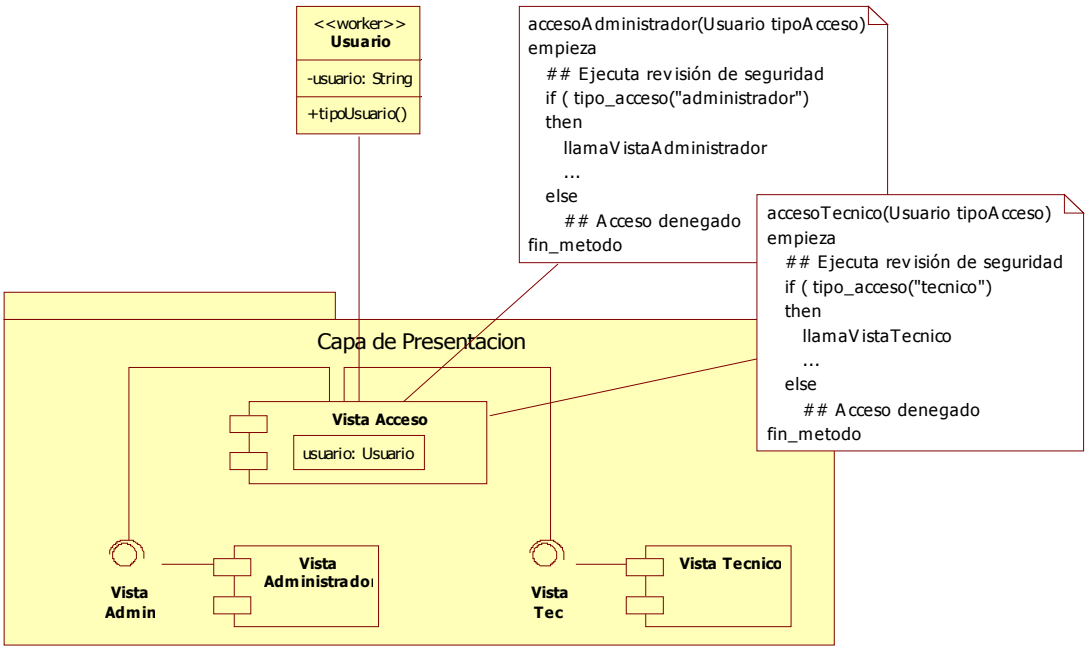


Figura 47: Patrón Authorization aplicado al sistema para satisfacer el escenario SEG-01

Patrón	Escenario y atributo de calidad afectado	Forma de aplicación	Nivel de satisfacción sobre escenarios
Model-View-Controller	USA-01 Usabilidad	Se aplica a varias capas. En la capa de Lógica de Negocio se toman como Controladores a los componentes llamados Servicio-Detectar-Fallas y Servicio-Recibir-Alarma ya que los controladores modifican al modelo que que en este caso es el Servicio-Mostrar-Topología (también se la capa de Lógica de Negocio). Posteriormente este modelo envía el cambio al componente Vista-Mostrar-Topología para ser mostrada las diferentes vistas	El escenario se considera totalmente satisfecho, ya que se asegura que la alarma será enviada a las vista correspondientes del Administrador y del Técnico
Authori-zation	SEG-01 Seguridad	Se aplica en el componente Vista-Acceso, donde se verifica el tipo de usuario que quiere acceder al sistema por medio de dos métodos. Cada uno de estos métodos llaman a la Vista-Administrador o a la Vista-Técnico	Se satisface totalmente debido a que se implementa un mecanismo de verificación de acceso de usuarios

Tabla 17: Forma de aplicar los patrones en el sistema y nivel se satisfacción de los escenarios

Driver\It	It1, It2, It3	It4	It5	It6	It7
DIS-01	No satisfecho	Totalmente			
MOD-01	No satisfecho				
DES-01	No satisfecho				
USA-01	No satisfecho			Totalmente	
DES-02	No satisfecho	Parcialmente			
SEG-01	No satisfecho			Totalmente	
PRU-01	No satisfecho		Totalmente		

Tabla 18: Escenarios satisfechos y nivel de satisfacción hasta la 6ta iteración

Como se muestra en la tabla 18, del total escenarios restan dos con los cuales no se ha trabajado. Tomando como base la tabla 12, se decide continuar con el escenario que tiene mayor prioridad de los dos con los que no se ha trabajado, este es DES-01.

7ma iteración:

El escenario DES-01 está relacionado con el tiempo de respuesta del sistema, al incorporar más dispositivos, ya que el Desempeño de éste no debe ser afectado. Este escenario se aplica en la capa de Manejo de Dispositivos,

específicamente en el componente Monitorear Dispositivos, el cual ya cuenta con la aplicación de un patrón que ayuda justamente a satisfacer este escenario ya que Master-Slaves aparte de apoyar a Disponibilidad, apoya también a Desempeño. El uso de los Monitores (esclavos) permite que al incorporar más dispositivos solo es necesario aumentar el número de ellos.

Debido a que el escenario queda cubierto por la segunda iteración se decide continuar con el escenario restante, este es MOD-01, correspondiente al atributo de calidad de Modificabilidad. Este escenario esta relacionado con la necesidad de tener más dispositivos para monitorear y solamente re-configurar el sistema (cambiar parámetros del sistema), por lo tanto está relacionado con la capa de Manejo de Dispositivos y con el componente Monitorear Dispositivos y su implementación. Este componente fue obtenido con el patrón Domain Object pero su implementación está encapsulada, por tal motivo se decide utilizar como patrón inicial a Encapsulated Implementation.

Al aplicar el Asistente se obtiene una lista de trece patrones, de los cuales sólo se toman en cuenta ocho debido a que son los que tienen más relación con la necesidad que se quiere satisfacer. Los patrones seleccionados como opción son: Strategy, Execute Around Object, Template Method, Object Adapter, Visitor, Wrapper Facade, Decorator, Interceptor, estos patrones están relacionados con la adaptación y la extensión de los sistemas. Finalmente se opta por dejar como posibles opciones a Strategy que se aplica cuando se tiene una serie de pasos (algoritmo) a seguir pero que pueden ser cambiados alguno de ellos para ser aplicados de distinta forma, Template Method que permite que objetos que comparten comportamiento puedan variar en alguna funcionalidad, Object Adapter permite que puedan se utilizadas implementaciones existentes en nuevos contextos a través de una interface, Wrapper Facade permite que no se pueda tener acceso a un API de funcionalidades de forma directa sino a través del objeto y Decorator que permite encapsular el comportamiento de un objeto "original" y hacer uso de otros con diferente comportamiento a través de una interface. De esta lista se decide utilizar el patrón Strategy como se muestra en la figura 48.

Strategy:

Cuando se agrega un dispositivo para ser monitoreado, los parámetros llegan al componente Servicio Agregar Dispositivo y éste manda los datos tanto para ser almacenados y para ser incluidos en el monitoreo. Al ser mandados para monitoreo llegan a un componente que contiene los pasos a seguir para agregar distintos tipos de dispositivos, llamado Incrementar Dispositivos (contexto), dependiendo del tipo de dispositivo que se quiera agregar se recibirán parámetros, los cuales serán utilizados correctamente gracias a un grupo de componentes distintos (Strategys) que se tienen para cada tipo de dispositivo. Estos componentes manejan los parámetros usados para cada dispositivo.

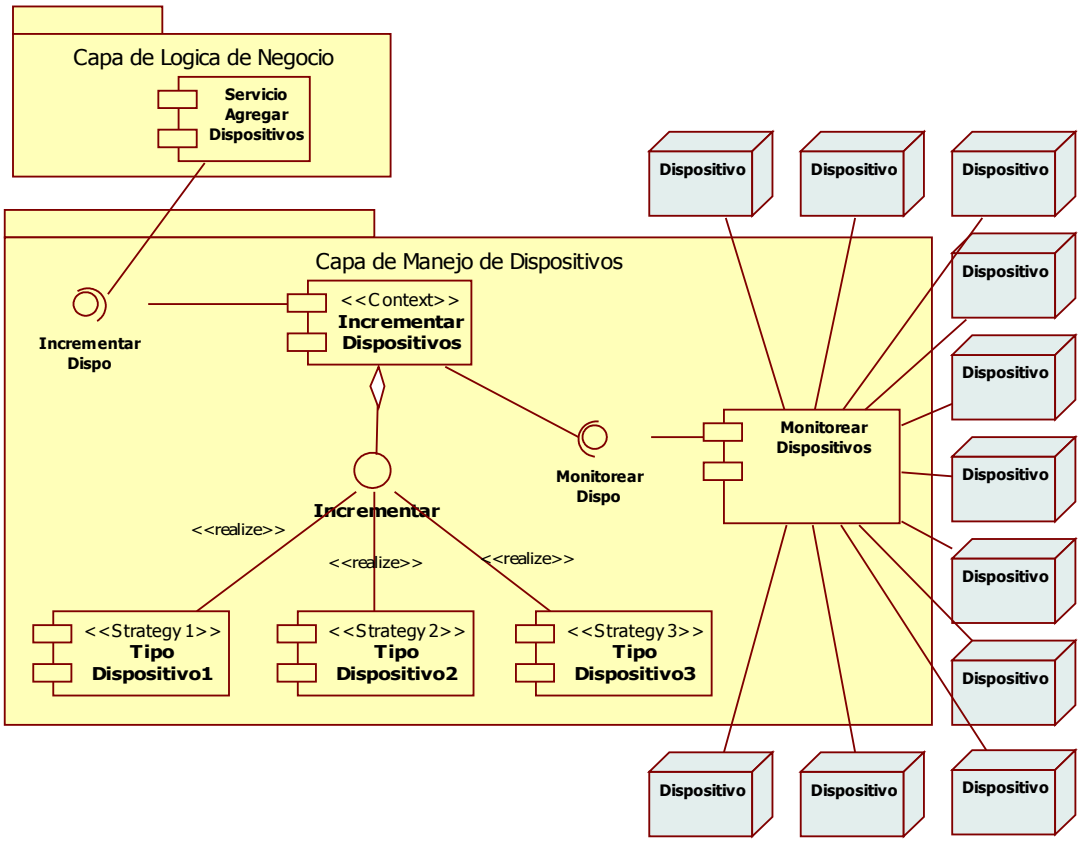


Figura 48: Patrón Strategy aplicado al sistema para satisfacer el escenario MOD-01

Con el uso del patrón Strategy se considera totalmente satisfecho el escenario MOD-01 para el diseño de la arquitectura, ya que permite que varios tipos de dispositivos sean agregados para monitoreo sin necesidad de re-codificar, únicamente re-configurando, es decir sólo con el cambio de parámetros. En la tabla se especifica la forma de aplicación del patrón Strategy sobre le sistema y el nivel de satisfacción que se logra con la aplicación del patrón.

Patrón	**Escenario y atributo de calidad afectado**	**Forma de aplicación**	**Nivel de satisfacción sobre escenarios**
Strategy	MOD-01 Modificabilidad	Se crea un componente que va a tener el algoritmo a seguir para realizar el incremento de dispositivos y para cada tipo de dispositivo se tendrá un componente distinto que se aplicará según el dispositivo a agregar	Se considera totalmente satisfecho ya que cumple con la necesidad de no cambiar código en caso de querer agregar nuevos dispositivos para monitoreo

Tabla 19: Forma de aplicar el patrón Strategy en el sistema y nivel se satisfacción del escenario

Driver\It	It1, It2, It3	It4	It5	It6	It7
DIS-01	No satisfecho	Totalmente			
MOD-01	No satisfecho				Totalmente
DES-01	No satisfecho				Totalmente
USA-01	No satisfecho			Totalmente	
DES-02	No satisfecho	Parcialmente			
SEG-01	No satisfecho			Totalmente	
PRU-01	No satisfecho		Totalmente		

Tabla 20: Escenarios satisfechos y nivel de satisfacción hasta la 7ma iteración

Como se muestra en la tabla 20, todos los escenarios con excepción de uno, se han considerado satisfechos totalmente. El escenario restante se puede satisfacer en alguna otra iteración, siendo que para este proyecto se da por concluido el presente ejemplo, ya que la finalidad de éste es la de mostrar la forma de realizar el diseño de la arquitectura con ayuda del Asistente.

4.3.3. Análisis de resultados

El análisis que se realizó para este ejemplo fue utilizando los mismos puntos tomados para el ejemplo anterior, esto es, patrones ofrecidos por el Asistente y la reducción de opciones que se tuvo con respecto a los contenidos en el libro POSA 4.

El análisis de resultados se presenta en la tabla 21, donde se muestra la necesidad que se tiene, el o los atributos que se quieren satisfacer, patrón de inicio, razón por la cual se tomó como tal el patrón de inicio. Finalmente se muestra la cantidad de patrones resultantes junto con el patrón elegido, el total de hijos que tiene dicho patrón de inicio y el porcentaje de reducción en comparación de los patrones-hijo del patrón de inicio y los ofrecido por el Asistente.

Necesidades del sistema	Atributos de Calidad	Patrón Inicial	Razones para utilizar el Patrón Inicial	Resultado MATDDS	Total de Patrones hijos	Porcentaje de Reducción
Estructurar el sistema	Modificabilidad	Domain Model	Este patrón en el nodo raíz del árbol de patrones, al igual que es un patrón del tipo estructural, es decir, da una estructura o base a la arquitectura que se empieza a diseñar	Lista de diez patrones, eligiendo Layers	Trece patrones hijos de Domain Model	33%
		Layer	Se necesita trabajar en cada una de las capas obtenidas de forma independiente	Lista de ocho patrones, eligiendo Domain Object	Ocho patrones hijos	0%
		Domain Object	Ya que se obtuvieron una serie de objetos del patrón Domain Object se decide trabajar con estos objetos	Lista de siete patrones, decidiendo utilizar Encapsulated Implementation	Lista de diez patrones hijos	30%
Es necesario tener disponibilidad en el monitoreo de dispositivos así como un buen desempeño al detectar los errores	Disponibilidad y Desempeño	Encapsulated Implementa-tion	Este patrón es el que permite encapsular la implementación de los objetos con los cuales se ha trabajado	Lista de tres patrones, eligiendo Master-Slave	Veintidós patrones hijos	86%
Tener más dispositivos para monitorear y solamente re-configurar el sistema	Facilidad de Pruebas	Domain Object	Se decide utilizar este patrón debido a que la necesidad afecta a todos los elementos obtenidos hasta el momento	Lista de cinco patrones, decidiendo utilizar Explicit Interface	Diez patrones	50%
Presentar los errores de los dispositivos en las vistas de los usuarios	Usabilidad	Domain Model	Están involucradas distintas partes del sistema, al igual que sus funcionalidades	Lista de cinco patrones, eligiendo Model-View-Controller	Trece patrones	60%
Seguridad que necesita el sistema para dar acceso a los usuarios	Seguridad	Explicit Interface	La necesidad involucra la forma en que se comunican los objetos, ésto lo hace por medio de las interfaces	Lista de cuatro patrones, eligiendo Authorization	Catorce patrones	71%
Los componentes deben poder ser probados de forma unitaria	Modificabilidad	Encapsulated Implementa-tion	La necesidad tiene que ver con el acceso a la implementación de los objetos relacionados con el monitoreo	Lista de trece patrones, decidiendo utilizar Strategy	Veintidós patrones	41%

Tabla 21: Análisis de resultados ofrecidos por el Asistente-MATDDS

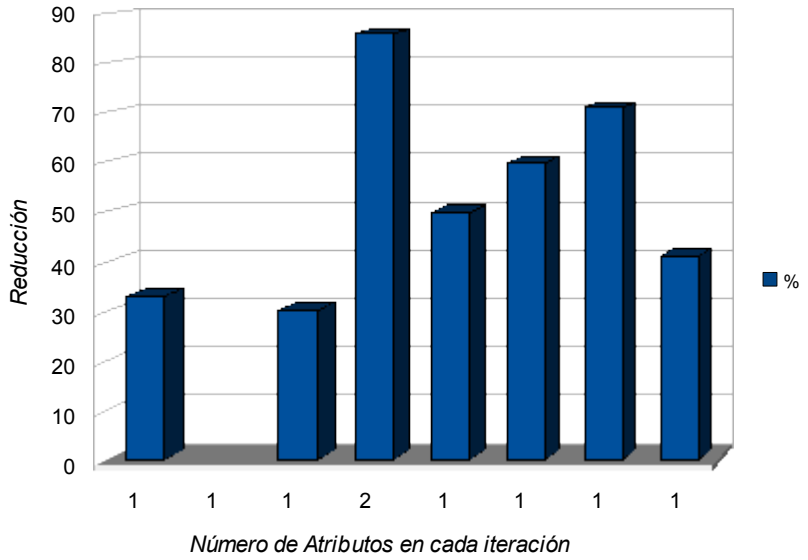
Haciendo ahora un análisis de la tabla 21 se obtuvieron los siguientes puntos:

1. Para este ejemplo se observa que en general (con excepción de un caso), el Asistente proporcionó un sub-conjunto menor de la lista total de patrones relacionados con el patrón elegido como inicial, es decir

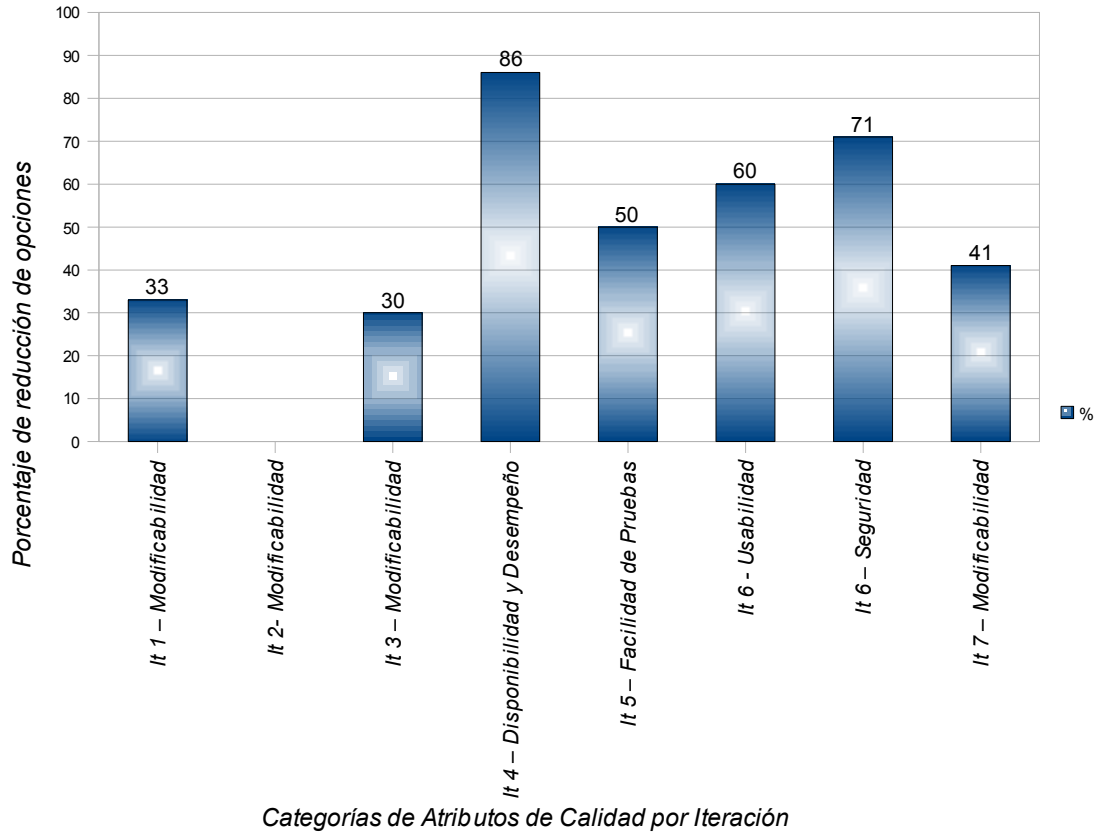
hubo un porcentaje de reducción.

2. Considerando que únicamente se tuvo una iteración donde se utilizaron dos atributos de calidad y los demás casos fueron con un atributo, el porcentaje de reducción fue mejor que en el ejemplo anterior, dado que en el ejemplo anterior los porcentajes cuando se utilizó un solo atributo fueron menores a 45% y para este caso fueron menores de 71%. Aún así prevalece el resultado de que a más atributos de calidad, el Asistente proporciona una cantidad menor de patrones opcionales.
3. También se observo que donde el porcentaje de reducción fue mayor, estuvieron involucrados atributos diferentes a Modificabilidad y Facilidad de Pruebas. Ésto se debe a que existe mayor cantidad de patrones que ayudan a estos atributos a diferencia de al restante de ellos.
4. Del total de ocasiones donde se utilizó el Asistente para obtener una lista de patrones como opción para satisfacer de alguna forma las necesidades o escenarios de atributos de calidad, sólo en una ocasión no hubo reducción, es decir se mostró la lista con el total de patrones relacionados con el patrón de inicio (segunda iteración). Cabe señalar que para este ejemplo no se obtuvo en ninguna ocasión un resultado con una única opción.

Los resultados obtenidos se plasmaron en las siguientes gráficas. En la gráfica 3 se puede apreciar que nuevamente se obtuvo una relación entre el número de atributos de calidad con el porcentaje de reducción de opciones proporcionadas por el Asistente. En este caso particularmente se observa que la variación de reducción entre la utilización de uno y dos atributos de calidad no fue muy amplia debido a que hubo un caso donde se obtiene una reducción del 71% utilizando un solo atributo de calidad, porcentaje cercano al caso donde se utilizan dos atributos donde hubo una reducción del 86%. Cabe hacer referencia al punto 3, que se mencionó anteriormente y con base en ese análisis se obtuvo la gráfica 4. En esta gráfica se observa la relación de los atributos de calidad de Modificabilidad y Facilidad de Pruebas, en las distintas iteraciones donde fueron utilizados, y el poco porcentaje de reducción de opciones.



Gráfica 3: Relación entre el número de atributos y el porcentaje de reducción de opciones



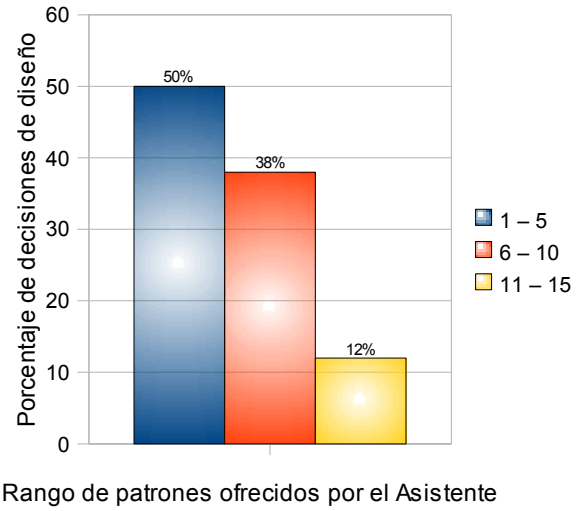
Gráfica 4: Relación Atributos de Calidad contra porcentaje de reducción de opciones en las distintas iteraciones (It) realizadas en el ejemplo anterior

Siguiendo con la gráfica 4, se observa que cuando se utiliza Modificabilidad (en las distintas iteraciones

realizadas en el ejemplo con Modificabilidad), el porcentaje de reducción es menor, aún más que cuando se utiliza Facilidad de Pruebas. Por lo contrario se observa que utilizando Usabilidad la reducción es mayor, de un 60% al igual que con Seguridad donde la reducción es de un 71%. Como se había mencionado anteriormente, ésto se debe a que para los últimos atributos mencionados no existen tanto patrones relacionados como con Modificabilidad y Facilidad de Pruebas.

Al igual que en la gráfica 2 (Rangos de patrones ofrecidos por el Asistente) del ejemplo anterior (Sistema de Gestión del Proceso de Almacenamiento), se consideró el porcentaje de iteraciones del actual ejemplo (Sistema de Gestión de Redes de Dispositivos) en relación al rango de patrones ofrecido por el Asistente, obteniendo la gráfica 5. Del total de ocho tomas de decisiones de diseño, una (12%) tuvo un rango alto de opciones proporcionadas por el Asistente (11 - 15); tres (38%) tuvieron un rango medio de opciones (6 - 10); y cuatro (50%) tuvieron un rango bajo (1 - 5) de opciones ofrecidas por el Asistente.

Como conclusión se puede decir que la reducción de patrones para este ejemplo fue mayor que para el ejemplo anterior. La razón está en que en el ejemplo se utilizaron atributos de calidad como Seguridad y Usabilidad que no tienen tantos patrones que los satisfagan, a diferencia de Modificabilidad y Facilidad de Pruebas. Otro punto a concluir es que se sigue apoyando la idea de que si se tienen más atributos de calidad de entrada para el Asistente, el porcentaje de reducción de opciones va a ser mayor. Finalmente se observó que al igual que en el ejemplo anterior, el arquitecto sigue tomando decisiones sobre el nodo de inicio, así como cuál patrón utilizar de las opciones ofrecidas por el Asistente-MATDDS y aunque éste presenta la ayuda, es necesario que el arquitecto selecciones el patrón.



Gráfica 5: Rangos de patrones ofrecidos por el Asistente (bajo, medio, alto)

4.4 Análisis General de MATDDS

Se realizó un primer análisis con los resultados obtenidos en los ejemplos anteriores de forma general, arrojando las siguientes observaciones:

1. De forma general siempre hubo un porcentaje de reducción en las opciones ofrecidas por el Asistente, con excepción de dos casos donde se utilizaron dos y un atributo de calidad correspondientes al ejemplo uno y al ejemplo dos respectivamente donde la reducción fue nula. Estos casos específicamente fueron:

   - En la segundo necesidad del primer ejemplo (ejemplo POSA 4), donde se utilizan los atributos de calidad de Modificabilidad y Facilidad de Pruebas, tomando como patrón de inicio a Layers. En este caso no se tuvo reducción debido a que todos los patrones hijos de Layers ayudan a la satisfacción de los dos atributos de calidad, por lo tanto el valor heurístico es el mismo ellos.
   - En el segundo ejemplo (dentro del contexto de ADD), específicamente cuando se está haciendo la estructuración del sistema, donde sólo se utiliza el atributo de calidad de Modificabilidad y como patrón de inicio a Layers. La razón es la misma que en el caso anterior, ya que todos los patrones relacionados con Layers satisfacen a Modificabilidad.

   Cabe mencionar que el porcentaje de reducción está ligado a la forma en que se obtiene el valor heurístico (ver fórmula 1), ya que éste es el que va a decidir cuales patrones se van a ofrecer como opciones en el caso de que varios patrones tengan el mismo valor, debido a que siempre se presentan como opción los que tienen el mayor valor heurístico.

2. Con base en el punto anterior y en otros casos, se observó que cuando se utilizaron los atributos de calidad de Modificabilidad y Facilidad de Pruebas como atributos de entrada en el Asistente, huvó en general poca reducción en el número de opciones ofrecidas por el Asistente. Caso contrario, al utilizar los atributos de calidad de Usabilidad y Seguridad la reducción de opciones fue mayor que con los dos primeros atributos mencionados. Ésto se debe a que Modificabilidad y Facilidad de Pruebas están cubiertos por un número mayor de patrones, después están los atributos de Disponibilidad y Desempeño con un número mediano de patrones que los cubran y finalmente están Usabilidad y Seguridad con un número menor de patrones. Se usa la palabra cubrir para referir un nivel positivo de satisfacción de los patrones sobre los atributos de calidad.

3. Cabe resaltar que el porcentaje de reducción de opciones está directamente relacionado con el número de patrones hijos del patrón padre, ya que si el número de patrones hijos es mayor, el porcentaje de reducción podría ser mayor. En cambio si el número de hijos en menor, posiblemente el porcentaje también sea menor.

4. Se observa que en ocasiones es necesario seleccionar en una misma iteración del Asistente más de una lista de patrones debido a que con la primera no se consideran satisfechos todos los atributos de calidad de entrada. También se podría dar el caso de que se tengan que elegir aún más listas (más iteraciones

del Asistente). Ésto es determinado por el Asistente, por lo tanto es el Asistente el que ofrece las siguientes listas. En su defecto, cuando el patrón no tiene hijos, el Asistente no presentará mas que los resultados con los atributos satisfechos con los patrones que los satisfacen.

5. En ambos ejemplos, se observa que es necesario tener una estructuración inicial del sistema. Logrando tal objetivo con un recorrido en anchura del grafo, ésto es, se toma en cuenta el primer nivel de profundidad del grafo, a partir de la raíz (Domain Model), las veces que sean necesario. Ésto se considera parte del método, ya que es recomendable que cuando se inicie el diseño de arquitectura de software se realice en primer lugar una estructuración del sistema.
6. A medida que se va recorriendo el grafo de patrones en profundidad, el nivel de detalle en el diseño es mayor. Ésto se debe a que en los primeros niveles, especialmente en el primer nivel, los patrones son estructurales. En cambio, en los últimos niveles se encuentran patrones dedicados a resolver cuestiones y problemáticas más específicas.

El segundo análisis se realizó tomando en cuenta las características y propiedades [38] expuestas en el capítulo de "Marco Teórico y Estado del Arte", específicamente la sección 2.6 de "Comparación de Herramientas". Se decidió hacer el análisis de MATDDS con base en estos puntos debido a que son igualmente utilizados para analizar otros métodos y herramientas de la misma categoría que MATDDS, los cuales también fueron presentados en la tabla 2 de comparación de herramientas en el capítulo mencionado anteriormente.

Debido a que MATDDS se puede ver como método o como programa, se decidió tomar de las herramientas comparadas en el capítulo mencionado anteriormente aquellas que sean sólo métodos y compararlas con MATDDS-Método y después tomar las herramientas que sean software y compararlos con MATDDS-Asistente. La finalidad es tener un perspectiva más amplia de MATDDS (Método y Asistente).

En primer lugar se realiza, como ya se mencionó, la comparación entre métodos. En este caso se hace la comparación de MATDDS-Método con el Modelo Cognitivo de Hinojosa, dicho Modelo es un método para obtener patrones de diseño a través de la aplicación de Prolog, es decir, un lenguaje interpretado. Esta comparación se muestra en la tabla 22.

Características y Propiedades	MATDDS – como método	Modelo Cognitivo de Hinojosa
Rastreo	No realiza rastreo, ya que únicamente es el método	No lo permite por ser sólo un modelo
Indexar	No se puede indexar por el mismo motivo del rastreo	No están clasificados
Búsqueda	La búsqueda se realiza por medio del valor heurístico de forma manual, es decir aplicando la fórmula o función heurística determinada	Se hace a través de situaciones que se quieren resolver
Soporte de Selección	El método permite obtener una lista de nombre de patrones, en caso de tener heurísticas iguales	No está definida la información en la documentación
Recomendaciones	Se obtiene una lista de los patrones que tienen influencia positiva en los atributos con los cuales se está trabajando	Sí hace sugerencias
Despliegue y Detalle	No muestra el detalle de la lista de patrones en caso de tener que elegir alguno de ellos, ya que única se obtiene el nombre de los patrones	No hace ningún tipo de despliegue y muestra detalles
Navegación	Se puede navegar por el grafo de patrones, de forma manual	No permite la navegación entre los patrones
Conexión	No permite hacer ninguna conexión	No permite conexión entre los patrones
Colaboración	Se puede realizar colaboración en el caso donde un usuario aumente el grafo o lo modifique para mejorarlo	No se reporta que acepte colaboración
Dominio de Manejo	Patrones de diseño y arquitectónicos	Once patrones de diseño
Requiere de Pre-procesamiento de Soluciones	Sí es necesario que se haga un procesamiento de los patrones que se van a utilizar, ya que es necesario obtener la tabla de prioridades así como la tabla de necesidades de cada patrón. En caso de que se vayan a incluir nuevos patrones es necesario incluirlos en el grafo de patrones relacionándolos con los ya existentes a través de las necesidades y obteniendo sus correspondientes tablas	Es necesario que los patrones estén expresados en clausulas de Horn, para poder ser usados por Prolog
Método	El método utiliza el algoritmo de búsqueda en profundidad, adaptado para su uso con patrones	El método de búsqueda es el uso de reglas del tipo "Si en verdad el antecedente, entonces es verdad el consecuente"
Guía de Soporte	La actual tesis es una guía de apoyo para el uso del método	Artículo
Fácil de Integrar a Métodos	El método se puede usar el cualquiera de los procesos de diseño de arquitectura mencionado en esta tesis, ya que apoya en la elección de patrones, solamente	Es fácil de integrar ya que no es necesario hacer cambios en los procesos para poder ser utilizado

Tabla 22: Comparación entre MATDDS y el Modelo Cognitivo de Hinojosa

Al hacer un análisis de las características con las cuales cumple cada uno de estos métodos se observa que MATDDS cumple con características importantes con las cuales no cumple el Modelo Cognitivo de Hinojosa, como son Soporte de Selección, Navegación, Colaboración. También se observa que en cuestión de propiedades, específicamente con respecto al Dominio MATDDS maneja un mayor número de patrones con respecto a la que maneja el Modelo de Hinojosa. Al igual que con MATDDS como método, MATDDS como programa (Asistente) se analizo a través de una comparación con las herramientas más afines, mostradas en la tabla 2 de la sección 2.6 del Estado del Arte.

Características y Propiedades	REBUILDER	Stylebase	PAKME	ADDSS	Web of Patterns	ArchE	Open Pattern Repositories	MATDDS - Programa
Búsqueda	Por medio de consultas a la base de datos	Por el nombre, alguna palabra o atributos de calidad	Por palabras clave o por operadores lógicos	Por medio de consultas a la base de datos	Por marcadores o servicios de búsqueda	Por medio de los escenarios de atributos de calidad	La búsqueda se hace por palabras clave	Por medio del mayor valor heurístico, determinado a través de los atributos de calidad
Soporte de Selección	Selección de alternativas por medio de CBR, mostrando una lista de soluciones	La selección se hace con base en la búsqueda sobre la base de datos	Selecciona varias alternativas y las muestra	Muestra una lista de soluciones, aunque no sean las más adecuadas	Muestra una lista de patrones	Muestra una lista de soluciones	Muestra las alternativas sugeridas por la selección	Muestra una lista de opciones para seleccionar y la información de cada una de ellas
Recomendaciones	La lista está clasificada de acuerdo con la consulta	No hace más recomendaciones que la búsqueda	No hace más sugerencias que las de la búsqueda	No realiza recomendaciones	No hace alguna recomendación	Hace recomendaciones de acuerdo a los escenarios	No hace una recomendación en particular	La lista de patrones mostrada contiene los patrones más recomendados
Despliegue y Detalle	Despliega la lista con diagramas UML	Despliega las soluciones con opción de poder ver su detalle	Despliega el resultado con opción de ver el detalle	Despliega la lista de soluciones con detalle y una mini-gráfica de cada solución	Despliega la lista pero no muestra detalle de los patrones	Muestra detalle de la recomendación hecha	Despliega una lista de posibilidades con opción de ver el detalle	Despliega la lista de opciones posibles y permite ver información de cada uno de los patrones desplegados
Navegación	No se puede ir de una solución a otra, ya que los casos son experiencias concretas	No tiene relaciones entre las soluciones	No se muestra la relación entre las soluciones	Sólo se pueden ligar las decisiones tomadas anteriormente con la arquitectura resultante	No se puede ir de una solución a otra	Se puede ir de una solución a otra por medio de la información que se va ingresando	Sí permite la navegación entre los patrones relacionados (no en el demo)	No permite navegar de un patrón a otro aunque estén relacionados, ésto solo lo puede hacer el programa por medio de los recorridos que se hacen en el grafo
Conexión	No se puede hacer ésto ya que se utiliza una base de datos	No puede hacer conexiones	No puede hacer conexiones	No se puede hacer conexiones hacia otros repositorios	Sólo con algunos repositorios (Swoogle, wopper server))	No permite hacer conexiones con otros repositorios	Permite la conexión sólo con el repositorio de la herramienta	No permite hacer conexiones, ya que el resultado se considera independiente en cada iteración
Colaboración	No menciona en la fuente el poder tener colaboración	Permite la colaboración de tal forma que se puede añadir o eliminar soluciones	Permite la colaboración de los usuarios para añadir y modificar las soluciones	No se permite compartir diseños a menos que el usuario creador decida colaborar con otros proyecto	Permite que los usuarios puedan tener colaboración	Permite la colaboración, ya que es un sistema experto	Permite la colaboración añadiendo y editando las soluciones	Se puede realizar colaboración en el caso donde un usuario aumente el grafo o lo modifique para mejorarlo. También permite hacer uso de otro grafo de patrones
Dominio de Manejo	Patrones de Diseño	Patrones arquitectónicos, de diseño, referencias arquitectónicas y micro y macro arquitecturas	Escenarios arquitectónicos, arquitecturas específicas, patrones y tácticas	Patrones y estilos arquitectónicos	Patrones de diseño	Patrones y tácticas de diseño	Patrones y tecnologías útiles	Patrones de Diseño y Arquitectónicos
Requerir de Pre-procesamiento de Soluciones	Cada solución se debe transformar en "caso" para ser almacenada	Las soluciones se deben almacenar en la base de datos	Se deben de almacenar las soluciones con su descripción	Se requiere almacenar los conocimientos en la base de datos	Es necesario que se almacenen los patrones con cierto formato	Sí se requiere de un pre-procesamiento de las soluciones	Se requiere almacenar los datos en el repositorio de la herramienta y los diagramas de patrones	Es necesario incluir cada patrón en el documento de texto utilizado para crear el grafo. También es necesario crear un archivo PDF con el mismo nombre del patrón, el cual contendrá su información
Método	CBR – Case_Based Reasoning	No provee algún método	No provee un método, utiliza la herramienta llamada Hipergate	No provee un método	Jane – Framework de Web Semántica	Rule-based (sistemas de predicción basados en reglas)	La fuente no menciona algún método	Utiliza MATDDS internamente
Guía de Soporte	Artículo	Guía de usuario	Artículo	Artículo	Páginas de Internet	Guías	Guía	Manual de usuario
Fácil de Integrar a Métodos	Es necesario adaptar el proceso de diseño para utilizar la herramienta	Es necesario hacer el proyecto en eclipse para utilizar la herramienta	No es necesario que el proceso de diseño se adapte para usar esta herramienta	Es de fácil integración ya que sólo intervine en la búsqueda y selección de soluciones	Es necesario tener el proyecto en Eclipse para poder hacer uso de ella	No es fácil de integrar a otros métodos que no usen escenarios de atributos de calidad	Es de fácil integración	Es de fácil integración, ya que no es necesario modificar los procesos

Tabla 23: Comparación entre MATDDS como programa con otras herramientas

Tomando en cuenta la tabla 23 de comparación de herramientas, se observa que MATDDS-Asistente aún tiene algunas deficiencias ya que le falta cubrir algunas características como son la Navegación, ya que ésto no lo pueden hacer los usuarios, sino sólo el Asistente; la conexión, aunque esta característica no la realizan mas que dos herramientas y únicamente con algunos repositorios. Con respecto a las propiedades, específicamente el Dominio, MATDDS-Asistente sólo maneja patrones de diseño y arquitectónicos, aunque muchas de las demás herramientas manejan un dominio menor que éste. A pesar de todo lo mencionado, MATDDS-Asistente se puede considerar como una herramienta de aprendizaje, ya que con su aplicación se puede aprender a usar los patrones que se encuentran en el grafo de patrones o más aún, se conocen los patrones, su aplicación de forma general y los niveles de satisfacción que tienen con respecto a los atributos de calidad.

Para concluir, MATDDS Método y Asistente cumplen con la mayoría de las características y propiedades que son sugeridas en el artículo "A Survey of Existing Approaches for Pattern Search and Selection" de Birukou [38]. También se observa que los resultados obtenidos a través de dos ejemplos, son satisfactorios a pesar de haber sido obtenidos por una persona sin experiencia con respecto a patrones y su aplicación en el diseño de la arquitectura.

Conclusiones y Trabajo Futuro

I. Síntesis

En este proyecto se analizaron algunos método auxiliares que guían la creación del diseño de la arquitectura de software, al igual que algunas herramientas que ayudan a hacer toma de decisiones de diseño. Se propuso un método (MATDDS) de fácil aplicación y fácil entendimiento para realizar esta toma de decisiones de diseño enfocada en patrones, donde influyen solamente atributos de calidad. El método utiliza la información contenida en el catálogo de patrones POSA 4 [17], esencialmente uniendo los mapas que se muestran en éste para generar un grafo de patrones. MATDDS realiza búsqueda de patrones sobre el grafo utilizando el algoritmo de búsqueda en profundidad, por medio de valores (heurísticas) que determinan caminos sobre él. A este algoritmo se le propuso un refinamiento en la búsqueda para facilitar su uso. Las variables utilizadas para determinar el valor heurístico de cada patrón se obtuvieron por medio del análisis de alrededor de cien patrones del mismo catálogo a través de una operación sencilla en cada uno de ellos.

MATDDS fue desarrollado en forma de método o serie de pasos a aplicar sobre un problema y como software (Asistente) para poder ser aplicado de igual forma en problemas donde se viera involucrada la obtención de patrones. Para comprobar su aplicabilidad se utilizó (como Asistente) en dos ejemplos, obteniendo resultados aceptables tomando en cuenta las condiciones en las cuales fue aplicado. Para realizar estos ejemplo se consideraron dos vertientes de diseño de arquitectura de software, la primera es la aplicada en el libro POSA 4 [17] y la segunda fue el método ADD propuesto por el SEI.

Finalmente se realizó una comparación entre las herramientas mencionadas en el capítulo de Estado del Arte y Marco Teórico, específicamente en las secciones 2.5 y utilizando como base las características y propiedades sugeridas en el artículo "Open Source Base Tools for Sharing and Reuse of Software Architectural Knowledge" [20] mencionadas en la sección 2.6 del mismo capítulo. Al hacer esta comparación se analizaron las características y propiedades con las cuales MATDDS cumplen, ya sea como Método o como Asistente.

II. Conclusiones

Idealmente la arquitectura de software es diseñada por personas con experiencia al respecto, que cubren cierto

nivel se conocimiento para poder hacer toma de decisiones de diseño, tales como saber sobre qué plataforma se implementará el sistema, tecnologías que ayudan a satisfacer necesidades, así como patrones arquitectónicos, de diseño y tácticas. Ciertamente la tarea de hacer diseño de arquitectura no es trivial, al contrario es complicada y aún más complicado es el contar con personas con experiencia, en este caso arquitectos que reúnan estas características, sin contar que la demanda de arquitectos con esta experiencia es grande. Por tal motivo existen métodos que guían el desarrollo del diseño de la arquitectura, pero éstos igualmente refieren la necesidad de la experiencia de los arquitectos en la toma de decisiones de diseño. Para hacer la toma de decisiones de diseño existen algunas herramientas igualmente auxiliares de los arquitectos, como son las mencionadas en esta tesis (sección 2.5 de Estado del Arte). Algunas de ellas sugieren hacer cambios en los procesos que se siguen para hacer el diseño y así poder hacer uso de dicha herramienta, lo cual incrementa el tiempo de inversión en la construcción del diseño de la arquitectura. Por estas razones se detecto la necesidad de encontrar una solución más factible a la problemática de toma de decisiones de diseño en el diseño de la arquitectura de software, creando un método (MATDDS) que ayude a la toma de decisiones de diseño donde se vean involucrados los patrones.

Con base en los resultados obtenidos con el presente proyecto de tesis, se concluye que se cumplió con el objetivo general de esta tesis, donde se propone la creación de un método para el diseño de arquitecturas de software que ayude a hacer toma de decisiones de diseño. Ésto se logró a través de los objetivos específicos:

1. Identificar la influencia que tiene cada patrón sobre varias categorías de atributo de calidad – Se identificó la influencia que tienen los patrones contenidos (alrededor de 100) en el libro o catálogo POSA 4 sobre los atributos de calidad.

2. Encontrar una estrategia para poder determinar el conjunto de patrones que satisfagan los atributos de calidad de un Sistema de Software dado – La estrategia encontrada para poder determinar un conjunto de patrones que satisfagan los atributos de calidad de un sistema de software dado fue, a través de la utilización de un grafo de patrones para lograr encontrar un conjunto de ellos que ayuda a satisfacer las necesidades de los involucrados, expresadas por medio de atributos de calidad. Para ésto se utilizó como motor de búsqueda el algoritmo en profundidad, adaptado de tal forma que el valor heurístico que utiliza este algoritmo es determinado por medio de la influencia que tiene cada patrón con respecto a los atributos de calidad y también de una ponderación asignada a las necesidades (atributos de calidad). Ésto se plasma en el capítulo III de esta tesis, específicamente la sección 3.2 de "Propuesta".

3. Desarrollar dicha estrategia para obtener un método que pueda ser aplicado – Teniendo la estrategia se logro desarrollar un método para poder ser utilizado como auxiliar en la toma de decisiones de diseño relacionadas con la arquitectura de software.

4. Crear una herramienta (Asistente) que facilite la aplicación del método obtenido y aplicarla a un ejemplo, de tal forma que se pueda hacer un análisis de resultados – Con base en el método obtenido se creó un

Asistente que fue aplicado a dos ejemplos de los cuales se analizaron sus resultados, identificando con este análisis pros y contras del mismo.

5. Evaluar los resultados obtenidos – Los resultados obtenidos se consideraron satisfactorios, tomando en cuenta que la aplicación de éste fue hecha por un arquitecto sin experiencia en el diseño de arquitecturas de software.

Con ésto se concluye que los objetivos, tanto el objetivo general como los específicos fueron cumplidos, a través de los cuales se obtuvieron algunos logros como son:

- Utilizar un algoritmo sencillo de búsqueda sobre un grafo, ésto permitió que el entendimiento de la forma de aplicar el método fuera de igual manera sencillo.
- La aplicación de MATDDS sobre el proceso de diseño ADD resulta casi de manera natural, sin necesidad de hacer cambios drásticos a la forma en que se aplica el proceso.
- Se observó que MATDDS se puede aplicar a otros procesos de diseño, como los que se presentan en esta tesis, ya que la intervención de éste es únicamente en la selección de patrones que satisfacen a los atributos de calidad. Por lo tanto, no afecta al proceso que se utilice para generar el diseño.
- Se creo un programa (Asistente) que facilita la aplicación de MATDDS, conteniendo la funcionalidad de éste, aumentado la rapidez con la cual se obtienen resultados.
- Con el Asistente se logró incluir una ayuda visual, es decir, se facilita la consulta de los patrones en la misma aplicación, al tener una lista de opciones.

En contraparte se observó:

- MATDDS apoya en la selección de los patrones, pero aún así es necesario hacer toma de decisiones de diseño, ya que sólo se hace la propuesta de patrones. El arquitecto es el que decide cuál patrón utilizar, la forma de aplicarlos o instanciarlos y él es el que debe decidir qué tanto se han satisfecho los atributos de calidad.
- La experimentación con MATDDS se vio restringida por el hecho de no haberse probado en un ambiente laboral, siendo los resultados obtenidos totalmente académicos

Finalmente se concluye que con la ayuda de MATDDS, ya sea como método o programa, es posible generar diseños de arquitectura de software, aún sin la experiencia en la aplicación de patrones y sin afectar el proceso que se esté utilizando o del cual se auxilie para hacer el desarrollo del diseño de la arquitectura de software. El nivel de ayuda que ofrece MATDDS es considerable, ya que va mostrando un camino para poder hacer la selección de patrones de diseño.

III. Trabajo Futuro

Como trabajo a futuro se propone:

- Incluir otro tipo de decisiones de diseño, como pueden ser tácticas de diseño o frameworks – Cuando se hace el diseño de la arquitectura se toman en cuenta varias decisiones, por lo tanto, es necesario incluirlas para tener un Método más robusto.
- Hacer pruebas con otros algoritmos de búsqueda sobre grafos – A pesar de que los resultados obtenidos al utilizar el algoritmo de búsqueda en profundidad fueron satisfactorios, se considera necesario experimentar con otros algoritmos que podrían brindar un mejor resultado. Así como incluir información contextual para apoyar de mejor forma la selección de patrones.
- Hacer pruebas en un ambiente laboral para observar los resultados – Debido a que no se tuvo el contacto necesario con alguna empresa, no se pudieron realizar pruebas de este tipo. Al mismo tiempo que no es tan factible realizar este tipo de pruebas debido a los tiempos que se manejan en empresas de desarrollo de software. De igual forma no todas las empresas llevan metodologías de diseño de arquitecturas, por lo tanto la integración de MATDDS no es tan factible ya que esta básicamente en dirigido a ser usado con metodologías de creación de diseño de arquitecturas de software.
- Hacer una base de datos con razones por las cuales se tomaron las decisiones tomadas en los ejemplos – Ésto ayudaría a que en un futuro se pudieran revisar las razones y tener otro punto para poder tomar decisiones. Haciendo ésto, no se perdería la experiencia de otros arquitectos al utilizar MATDDS-Asistente. Esta información se podría expresar de alguna manera que pueda ser tomado en cuenta por el Asistente para seleccionar patrones.

Bibliografía

[1] Mary Shaw and Paul Clements, "The Golden Age of Software Architecture", IEEE Software vol.23, no. 2, 2006.

[2] Ruth Malan and Dana Bredemeyer, "Defining Non-Functional Requirements", Architecture Resources for Enterprise Advantage, 2001.

[3] Peter Eeles, "Capturing Architectural Requirements", IEEE Computer Society Vancouver Chapter, Febrero 2006.

[4] Rick Kazman and Len Bass, "Categorizing Business Goals for Software Architectures", Software Engineering Institute, Carnegie Mellon University, Diciembre, 2005.

[5] Anthony J. Lattanze, "Architecting Software Intensive Systems: A Practitioners Guide", Carnegie Mellon University, Pittsburgh Pennsylvania, USA, Auerbach Publications, Noviembre, 2008.

[6] Ian Sommerville, "Ingeniería de Software", Pearson Addison Wesley, 2005.

[7] Philippe Kruchten, "The 4+1 Views: Model of Software Architecture", IEEE Software, vol 12, no. 6, 1995, pp. 12-50.

[8] "Software Engineering Institute", Carnegie Mellon University, http://www.sei.cmu.edu/architecture/

[9] Stephen Stelting, Olav Maassen, "Patrones de diseño aplicados a Java", Pearson Addison Wesley, 2003.

[10] Len Bass, Paul Clements, Rick Kazman, "Software Architecture in Practice", Addison Wesley, 2003.

[11] IBM, Rational Unified Process (RUP)

[12] "OpenUP", Eclipse Process Framework Project, http://epf.eclipse.org/wikis/openup/index.htm

[13] Rob Wojcik, Felix Bachmann, Len Bass, Paul Clements, Paulo Merson, Robert Nord, Bill Wood, "Attribute-Driven Design (ADD)", Technical Report CMU/SEI-2006-023, Software Engineering Institute, Carnegie Mellon University, Software Architecture Technology Initiative, Noviembre 2006.

[14] Anthony J. Lattanze, "The Architecture Centric Development Method", School of Computer Science, Carnegie Mellon University, Pittsburgh, Febrero 2005.

[15] Paul Clementas, Rick Kazman, Mark Klein, "Evaluating Software Architectures: Methods and Case Studies", Addison Wesley, 2003,.

[16] W. Edwards Deming, "Out of the Crisis", The MIT Press, 1986.

[17] Frank Buschmann, Kevlin Henney, Douglas C. Schmit, "Pattern-Oriented Software Architecture: a pattern language for distributed computing. Volume 4", Wiley & Sons, 2007.

[18] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissdes, "Design Patterns, Elements of Reusable Object-Oriented Software", Addison Wesley, 1995.

[19] Patricia Lago, Paris Avgeriou, "First Workshop on Sharing and Reusing Architectural Knowledge", ACM SIGSOFT Software Engineering Notes, Septiembre 2006.

[20] Katja Henttonen, Mari Matinlassi, "Open Source Base Tools for Sharing and Reuse of Software Architectural Knowledge", VVT Technical Research Centre of Finland, Octubre 2009.

[21] Felix Bachmann, Len Bass, Mark, Klein, "Preliminary Design of ArchE: A SoftwareArchitecture DesignAssistant", Technical Report CMU/SEI-2003-TR-021, Carnegie Mellon University, Software Engineering Institute , Septiembre, 2003.

[22] Christodoulos A. Floudas, Panos M. Pardalos, editores, "Encyclopedia of Optimization", Springer, 2nd Edición, 2009.

[23] William J. Raynor Jr., "The International Dictionary of Artificial Intelligence", The Glenlake Publishing Company, Ltd., 1999.

[24] Martin Fowler, "Patterns of Enterprise Application Architecture", Addison-Wesley, 2002.

[25] Deepak Aluk, John Crupi, Dan Malks, "Core J2EE Patterns Best Practices and Design Strategies", Prentice Hall / Sun Microsystems Press, 2nd Edición, 2003.

[26] Gregor Hohpe, Bobby Woolf, "Enterprise Integratión Patterns", Addison-Wesley, 2002.

[27] P. Gomes, F. C. Peereira, P. Paiva, N. Seco, J. L. Ferrera, C. Bento, "REBUILDER: A CBR Approach to Knowlede Management in Software Design", University of Coimbra, Portugal.

[28] "WordNet: A lexical database for English", Encyclopedia of Language and Linguistics, 2nd Edición, Oxford: Elservier, 2005, Princeton University, http://wordnet.princeton.edu/

[29] "Tigris.org: Open Source Software Engineering Tools", VTT Technical Research Centre of Finland, 2008, http://stylebase.tigris.org/

[30] "STYLEBASE FOR ECLIPSE – userguide", Technical Research Center of Finland, Free Software Foundation, http://www.tigris.org/files/documents/4528/41113/userguide.pdf

[31] M. A. Babar, X. Wang, I. Gorton, "PAKME: A Tool for Capturing and Using Architecture Design Knowledge", Empirical Software Engineering, National ICT Australian Ltd & CSE, UNSW, MRO Software.

[32] Rafael Capilla, Francisco Nava, Sandra Pérez, Juan C. Dueñas, "A Web-based Tool for Managing Architectural Design Decisions", Universidad Rey Juan Carlos, Universidad Politécnica de Madrid, 2006.

[33] "The Web of Oatterns Project", Jens Dietrich, Institute of Information Sciences and Technology, New Zealand, 2004, http://www-ist.massey.ac.nz/wop/

[34] "ISWRL: A Semantic Web Rule Language Combining OWL and Rule ML", Boley H, National Research Council of Canada, Network Interface, and Stanford University, 2004, http://www.w3.org/Submission/SWRL/

[35] "Open Pattern Repository", Fontys University of Applied Sciences, Software Engineering and Architecture (SEARCH) University of Groningen, http://patternrepository.com/welcome.iface

[36] "Open Pattern Repository", Fontys University of Applied Sciences, Software Engineering and Architecture (SEARCH) University of Groningen, http://code.google.com/p/openpatternrepository/

[37] A. Hinojosa, "A Cognitive Model of Design Pattern Selection", Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology.

[38] A. Birukou, "A Survey of Existing Approaches for Pattern Search and Selection", Proceedings of the 15th European Conference on Pattern Languages of Programs (EuroPLoP'2010), Alemania, 2010.

Apéndice A

I. Requerimientos

Para lograr la recolección y el análisis de requerimientos se utilizan algunas técnicas o métodos como guías, por ejemplo el modelo de clasificación *FURPS+* o el taller *QAW*. Estas técnicas señalan los requerimiento funcionales, atributos de calidad y restricciones que se deben de obtener de acuerdo a las necesidades solicitadas por los involucrados.

FURPS+

Este es un modelo de clasificación de requerimientos funcionales que afectan a la arquitectura, atributos de calidad y restricciones (requerimientos no-funcionales), cuyo nombre se deriva del acrónimo por sus siglas en ingles de:

- Funcionalidad (Functionality): Se refiere a los requerimientos funcionales que afectan a la arquitectura y representan las características del producto, describe lo que el usuario puede hacer a través del sistema.
- Usabilidad (Usability): Se refiere a las características que permiten el fácil uso del sistema, como son las interfaces del usuario con el sistema.
- Confiabilidad (Reliability): Se refiere a las características del sistema como disponibilidad, exactitud de cálculos, habilidad del sistema para recuperarse de fallos, en pocas palabras es la solidez del sistema en tiempo de ejecución.
- Rendimiento (Performance): Son características o requerimientos del sistema que tienen que ver con la velocidad de respuesta, velocidad de recuperación, velocidad de encendido y apagado del sistema.
- Soporte (Supportability): Son las características que tienen que ver con la facilidad para hacer pruebas al sistema, que tan adaptable es, que tan fácil de mantener, la facilidad de configuración, la escalabilidad, etc.

El "+" del nombre de FURPS+ representa a las restricciones del sistema:

- Requerimientos de Diseño: Se refieren a las limitaciones para hacer el diseño (opciones del sistema – tipo de BD).

- Requerimientos de Implementación: Especifican o limitan la construcción del sistema (lenguajes, reglas de programación, estándares a usar).
- Requerimientos de Interfaz: Limita formatos u otros factores externos que usen interfaz.
- Requerimientos Físicos: Limitaciones para el hardware.

QAW

QAW es un método utilizado para capturar, colectar y organizar los atributos de calidad formando con ellos escenarios, haciendo esto en una etapa temprana del ciclo de vida del desarrollo de la arquitectura (antes de que la arquitectura sea diseñada y creada). Este método facilita la participación de un grupo de involucrados del sistema como son algunos miembros del equipo de desarrollo, miembros de la organización que solicita el sistema y el arquitecto. Los participantes aportan información e ideas sobre el sistema y sus atributos de calidad para poder obtener escenarios priorizados y refinados.

Los escenarios son el elemento principal del taller QAW, estos escenarios son situaciones medibles que se puede presentar en el sistema. Los escenarios están compuestos por:

- Estímulo del escenario – Es el estimulo o condición que se tiene para provoca el escenario.
- Fuente del estímulo – Que o quien (humano o sistema) genera el estímulo, puede ser interna o externa.
- Entorno – Condiciones bajo las cuales ocurre el estímulo.
- Artefacto estimulado – El sistema completo o partes de él que son afectadas por el estímulo.
- Respuesta – Que hace el sistema ante la llegada del estímulo.
- Medición de respuesta – Cuando la respuesta ocurre debe ser medida de alguna manera para que el requerimiento pueda ser probado.

QAW se lleva a cabo en tres fases, en la primera (*Fase de Preparación*) se hace un acercamiento entre el grupo de desarrollo (líder) con miembros de la organización solicitante (involucrados) por teléfono o se llega a acuerdos por medio de correos electrónicos. La segunda (Fase de Construcción) es donde se reúnen los involucrados y los miembros del equipo de desarrollo, tiene una duración aproximada de 1 a 1.5 días. En la tercera fase (*Fase de Reporte*) el contacto se realiza vía telefónica o medio correos electrónicos.

II. Etapas de ACDM

Las etapas de ACDM se describen brevemente a continuación:

- Descubrimiento de Drivers Arquitectónicos – Se realiza una junta con los involucrados para descubrir y

documentar los drivers arquitectónicos.

- Establecer los Alances del Proyecto – Se crea un Tratado de Trabajo (Statement of Work) y un Plan de Proyecto Preliminar (Preliminary Project Plan).
- Crear una Arquitectura Conceptual – Se crea una arquitectura inicial, a la cual se le llama conceptual debido a que no se ha construido y solo es el diseño de la arquitectura, la cual incluye una vista de procesos (run-time view), una vista de desarrollo (code view) y una vista física (physical view).
- Revisión de la Arquitectura – Revisión de la arquitectura conceptual para describir y documentar riesgos y problemas.
- La Producción Va o No Va – Se priorizan los riesgos y problemas que se descubrieron en la revisión de la arquitectura conceptual para decidir si la arquitectura se produce o es necesario que se refine un poco más.

Si la arquitectura necesita ser refinada entonces, las dos siguientes etapas serán diferentes de cuando la arquitectura esta lista para ser construida. Cuando la arquitectura necesita ser refinada las etapas son las siguientes:

- Planteamiento de Experimento – Se crean experimentos o pruebas en el diseño (prototipo) para mitigar riesgos y defectos que se descubrieron en la revisión de la arquitectura.
- Ejecución del Experimento y refinamiento de la Arquitectura – Se lleva a cabo el experimento y se documentan los resultados. La arquitectura es refinada con base en los resultados obtenidos.

Cuando la arquitectura no necesita ser refinada, se llega directamente a las etapas de construcción:

- Planeación de Producción – Se crea un plan detallado para la construcción del sistema basado en la arquitectura refinada, cada elemento de la arquitectura tiene un responsable el cual se encargará de que el elemento sea completado.
- Producción – Se ejecuta el plan de producción que incluye la construcción de cada elemento de la arquitectura hasta su integración para completar la arquitectura, así como las pruebas para cada elemento y las pruebas del sistema ya integrado.

III. Especificación del SEI para Escenarios de Atributos de Calidad

Para cada uno de los escenarios que representan a algún atributo de calidad, el SEI sugiere que contengan seis partes [10], estas son:

- Estímulo: Es una condición que es necesario considerar al momento que éste llega al sistema.
- Fuente de Estímulo: Es algún tipo de entidad, ya sea humano, otro sistema o algún otro actor, el cual

genera el estímulo.

- Entorno: Son las condiciones bajo las cuales ocurre el estímulo.
- Artefacto: Es aquel artefacto que es estimulado, éste puede ser todo el sistema o alguna de sus partes.
- Respuesta al Estímulo: Es la actividad que es llevada a cabo por el artefacto al llegar el estímulo.
- Medida de la respuesta: Cuando la respuesta ocurre, ésta debe de ser medible para de esta forma poder probar el requerimiento.

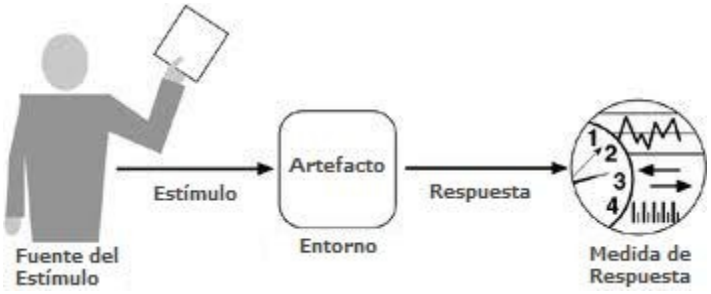


Figura 49: Partes sugeridas para representar un escenario de atributos de calidad [10]

IV. Ponderaciones de Patrones

A continuación se presentan las ponderaciones obtenidas con base en las descripciones de 101 patrones contenidos en el libro POSA 4 [17], correspondientes a los seis atributos de calidad utilizados en esta tesis.

Domain Model	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Layers	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

MVC	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	1
Seguridad	0
Facilidad de pruebas	1

Presentation Abstract Control	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	1
Seguridad	0
Facilidad de pruebas	1

Microkernel	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	1
Seguridad	0
Facilidad de pruebas	0

Reflection	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	1
Seguridad	0
Facilidad de pruebas	0

Pipes and Filters	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Shared Repository	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	1
Seguridad	0
Facilidad de pruebas	0

Blackboard	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Domain Object	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	1
Facilidad de pruebas	1

Messaging	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	1
Facilidad de pruebas	1

Message Channel	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	1
Facilidad de pruebas	0

Message Translator	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	-1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Acceptor-Connector	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Asynchronous Completion Token	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Message Router	Ponderación
Modificabilidad	1
Seguridad	0
Disponibilidad	0
Facilidad de pruebas	0
Desempeño	-1
Usabilidad	0

Publisher-Subscriber	Ponderación
Modificabilidad	1
Seguridad	-1
Disponibilidad	0
Facilidad de pruebas	0
Desempeño	1
Usabilidad	0

Explicit Interface	Ponderación
Modificabilidad	1
Facilidad de pruebas	0
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0

Extension Interface	Ponderación
Modificabilidad	1
Facilidad de pruebas	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0

Broker	Ponderación
Modificabilidad	1
Seguridad	0
Disponibilidad	0
Facilidad de pruebas	1
Desempeño	1
Usabilidad	0

Client Proxy	Ponderación
Modificabilidad	1
Seguridad	0
Disponibilidad	0
Facilidad de pruebas	1
Desempeño	0
Usabilidad	0

Introspective Interface	Ponderación
Modificabilidad	1
Facilidad de pruebas	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0

Dynamic Invocation Interface	Ponderación
Modificabilidad	1
Facilidad de pruebas	1
Disponibilidad	0
Desempeño	-1
Usabilidad	0
Seguridad	0

Requestor	Ponderación
Modificabilidad	0
Seguridad	1
Disponibilidad	0
Facilidad de pruebas	1
Desempeño	0
Usabilidad	0

Invoker	Ponderación
Modificabilidad	1
Seguridad	-1
Disponibilidad	0
Facilidad de pruebas	0
Desempeño	-1
Usabilidad	0

Proxy	Ponderación
Modificabilidad	1
Facilidad de pruebas	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	1

Business Delegate	Ponderación
Modificabilidad	0
Facilidad de pruebas	1
Disponibilidad	1
Desempeño	0
Usabilidad	0
Seguridad	0

Reactor	Ponderación
Modificabilidad	0
Seguridad	0
Disponibilidad	0
Facilidad de pruebas	1
Desempeño	1
Usabilidad	0

Proactor	Ponderación
Modificabilidad	0
Seguridad	0
Disponibilidad	0
Facilidad de pruebas	0
Desempeño	1
Usabilidad	0

Facade	Ponderación
Modificabilidad	1
Facilidad de pruebas	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0

Combined Method	Ponderación
Modificabilidad	0
Facilidad de pruebas	0
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0

Iterator	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Encapsulated Implementation	Ponderación
Modificabilidad	1
Disponibilidad	1
Desempeño	0
Usabilidad	1
Seguridad	0
Facilidad de pruebas	1

Whole-Part	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Composite	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Half-Object Plus Protocol	Ponderación
Modificabilidad	1
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Master-Slave	Ponderación
Modificabilidad	0
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Replicated Component Group	Ponderación
Modificabilidad	-1
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Page Controller	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	1
Seguridad	1
Facilidad de pruebas	0

Front Controller	Ponderación
Modificabilidad	1
Disponibilidad	-1
Desempeño	-1
Usabilidad	1
Seguridad	1
Facilidad de pruebas	0

Command Processor	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	1
Seguridad	1
Facilidad de pruebas	1

Template View	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	1
Seguridad	0
Facilidad de pruebas	0

Firewall Proxy	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	1
Facilidad de pruebas	0

Authorization	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	1
Facilidad de pruebas	0

Half-Sync/Half-Async	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Leader and Followers	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Active Object	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Monitor Object	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Future	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Thread-Safe Interface	Ponderación
Modificabilidad	0
Disponibilidad	1
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Double-Checked Locking	Ponderación
Modificabilidad	0
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Strategized Locking	Ponderación
Modificabilidad	0
Disponibilidad	1
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Scoped Locking	Ponderación
Modificabilidad	0
Disponibilidad	1
Desempeño	-1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Message	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Bridge	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Thread-Specific Storage	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Copied Value	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	-1
Usabilidad	0
Seguridad	1
Facilidad de pruebas	0

Object Adapter	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	-1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Chain of Responsibility	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	-1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Observer	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Mediator	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	-1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Interpreter	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Interceptor	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	-1
Usabilidad	0
Seguridad	-1
Facilidad de pruebas	0

Command	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Memento	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	-1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Visitor	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Decorator	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Context Object	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Data Transfer Object	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Execute-Around Object	Ponderación
Modificabilidad	1
Disponibilidad	1
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Template Method	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Strategy	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Null Object	Ponderación
Modificabilidad	1
Disponibilidad	-1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Wrapper Facade	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Objects for States	Ponderación
Modificabilidad	1
Disponibilidad	-1
Desempeño	-1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Methods for States	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Collections for States	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Container	Ponderación
Modificabilidad	1
Disponibilidad	1
Desempeño	0
Usabilidad	0
Seguridad	1
Facilidad de pruebas	0

Component Configurator	Ponderación
Modificabilidad	1
Disponibilidad	1
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Object Manager	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Lookup	Ponderación
Modificabilidad	-1
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Virtual Proxy	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Lifecycle Callback	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Task Coordinator	Ponderación
Modificabilidad	1
Disponibilidad	1
Desempeño	-1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Resource Pool	Ponderación
Modificabilidad	1
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Lazy Acquisition	Ponderación
Modificabilidad	0
Disponibilidad	-1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Eager Acquisition	Ponderación
Modificabilidad	0
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Activator	Ponderación
Modificabilidad	0
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Evictor	Ponderación
Modificabilidad	0
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Leasing	Ponderación
Modificabilidad	1
Disponibilidad	1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Automated Garbage Collection	Ponderación
Modificabilidad	0
Disponibilidad	-1
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Counting Handle	Ponderación
Modificabilidad	0
Disponibilidad	-1
Desempeño	-1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Abstract Factory	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Builder	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Factory Method	Ponderación
Modificabilidad	1
Disponibilidad	1
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Disposal Method	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0

Database Access Layer	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	0
Facilidad de pruebas	1

Data Mapper	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	1
Facilidad de pruebas	0

Row Data Gateway	Ponderación
Modificabilidad	1
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	1
Facilidad de pruebas	1

Table Data Gateway	Ponderación
Modificabilidad	0
Disponibilidad	0
Desempeño	0
Usabilidad	0
Seguridad	1
Facilidad de pruebas	1

Active Record	Ponderación
Modificabilidad	-1
Disponibilidad	0
Desempeño	1
Usabilidad	0
Seguridad	0
Facilidad de pruebas	0