TRIPODE: UN CATÁLOGO DE PATRONES ASPECTUALES DE ANÁLISIS PARA EL DOMINIO POS BASADO EN MORENA

IVÁN DARÍO CISNEROS NARVÁEZ
GLORIA JANNETH ERAZO CALVACHE
GERMÁN DARÍO PEÑA ARTURO

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2010

TRIPODE: UN CATÁLOGO DE PATRONES ASPECTUALES DE ANÁLISIS PARA EL DOMINIO POS BASADO EN MORENA

IVÁN DARÍO CISNEROS NARVÁEZ
GLORIA JANNETH ERAZO CALVACHE
GERMÁN DARÍO PEÑA ARTURO

TRABAJO DE GRADO PRESENTADO COMO REQUISITO PARCIAL PARA OPTAR AL TÍTULO DE INGENIEROS DE SISTEMAS

ING. ALEXANDER BARÓN SALAZAR
DIRECTOR

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2010

“Las ideas y conclusiones aportadas en el trabajo de grado son responsabilidad exclusiva de sus autores”.

Artículo 1º. del acuerdo No. 324 del 11 de Octubre de 1966 emanado del honorable Consejo Directivo de la Universidad de Nariño.

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

_____

Director: Ing. Alexander Barón Salazar

_____

_____

_____

_____

_____

Jurado: Ing. Manuel Bolaños González

_____

_____

_____

_____

_____

Jurado: Ing. Sandra Vallejo Chamorro

San Juan de Pasto, 13 de abril de 2010

AGRADECIMIENTOS

Al Ingeniero Alexander Barón, nuestro director de tesis, quien nos dio la oportunidad de integrar el grupo de investigación GALERAS LIS de la Universidad de Nariño y de hacer parte de este gran proyecto; a él muchas gracias porque nos guió en este camino nuevo de conocimiento.

A nuestras familias y demás allegados, quienes nos han dado todo su apoyo durante nuestro proceso de formación, para ellos nuestra más grande manifestación de gratitud.

RESUMEN

El proceso de desarrollo orientado por aspectos es una evolución en el desarrollo de software, sobre el cual se están haciendo grandes esfuerzos para lograr que su adopción sea más rápida en el ambiente empresarial, para esto se están desarrollando estudios y creando nuevas metodologías. Este es el caso del proyecto MEDUSA “Marco Metodológico Para El Desarrollo Orientado a Aspectos”, su objetivo es ofrecer una más rápida adopción de la metodología de desarrollo por aspectos por parte de los desarrolladores.

Con el desarrollo de este marco metodológico surgen diversas áreas de acción, una de ellas el uso de catálogos para la reutilización de componentes, lo que genera la necesidad de un esquema para la gestión de catálogos de patrones de análisis - ESKEMA, el cual incluye el Modelo de Representación de Patrones Aspectuales de Análisis – MORENA.

MORENA debe garantizar su eficiencia y eficacia a la hora de representar patrones aspectuales de análisis, demostrando que es útil y aplicable dentro del proceso de desarrollo de software. Igualmente, es necesario determinar que este modelo sea correcto antes de realizar las fases de implementación y aplicación, para evitar que las posibles fallas sean heredadas.

Debido a lo anterior, surge este proyecto de tesis, el cual tiene como propósito principal el realizar la validación de MORENA a partir de la construcción de TRIPODE: un catálogo de patrones aspectuales de análisis para el dominio POS basado en MORENA. La construcción de TRIPODE provee el escenario apropiado para validar el modelo y generar propuestas de mejoramiento que retroalimenten MORENA.

ABSTRACT

Aspect-oriented development process is an evolution in software development on which are making great efforts to get its faster adoption in the enterprise environment, for this, studies are developing and creating new methodologies. This is the case of the “Methodological Framework for Aspect-Oriented Development” MEDUSA project, its goal is to provide a faster adoption of the aspect-oriented development methodology by the developers.

With the development of this methodological framework a variety of action areas arise, one of them is the use of catalogs for the component reuse, which creates the need for a management scheme for analysis pattern catalogs - ESKEMA, this one includes the representation model for aspectual analysis pattern - MORENA.

MORENA must ensure its efficiency and effectiveness in representing aspectual analysis patterns, proving to be useful and applicable in the software development process. It is also necessary to determine that this model is correct before making the application and implementation phases, to prevent possible failures are inherited.

Due to the above, this thesis project rise, which has as main purpose to validate MORENA from building TRIPODE: an aspectual analysis pattern catalog for the POS domain based on MORENA. The construction of TRIPODE provides the appropriate setting to validate the model and generate proposals for improvement to provide feedback for MORENA.

TABLA DE CONTENIDO

Página

LISTA DE TABLAS

Página

LISTA DE FIGURAS

Página

GLOSARIO

ABSTRACCIÓN: Características esenciales de una entidad que la distinguen de otras entidades. Una abstracción define una frontera desde la perspectiva del observador.

ACTIVO DE SOFTWARE: Elementos reutilizables de software que ayuden a detectar, describir y compartir las soluciones para los problemas que se presentan en el desarrollo de software.

ACTOR: Conjunto coherente de roles que juegan los usuarios de los casos de uso cuando interactúan con éstos.

ALCANCE: Contexto que da significado a un nombre.

AOD: Aspect-Oriented Design – Diseño orientado por aspectos

AOSD: Aspect oriented software development - Desarrollo de software orientado por aspectos.

ARTEFACTO: Pieza de información que es utilizada o producida por un proceso de desarrollo de software.

ASPECTO: Módulo que encapsula la implementación de un *crosscutting concern.*

ATRIBUTO: Propiedad con nombre de un clasificador que describe un rango de valores que pueden contener las instancias de la propiedad.

CAMBIO DURANTE LA VENTA: Hace referencia a los posibles flujos alternativos que se pueden presentar durante la venta (devolución, cancelación y/o adición de productos).

CARDINALIDAD: Número de elementos en un conjunto.

CÁTALOGO: Colección de activos de software en la cual éstos se compilan, organizan y relacionan.

CLASE: Descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.

COLABORACIÓN: Representa una interacción entre objetos cuyo propósito es la realización de alguna actividad, y en donde cada objeto participante juega un determinado rol.

CONCERN: Diferentes temas o asuntos de los que es necesario ocuparse para resolver el problema.

CONTEXTO: Conjunto de elementos relacionados para un objetivo particular, como especificar una operación.

CROSSCUTING CONCERNS: Concerns cuya funcionalidad afecta a varios módulos ya definidos.

DEPÓSITO: Similar a un atributo de cantidad, con una entrada añadida para cada cambio a su nivel.

DIAGRAMA: Representación gráfica de un conjunto de elementos; representado la mayoría de las veces como un grafo conexo de nodos (elementos) y arcos (relaciones).

DIAGRAMA DE CLASES: Muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Se utiliza para modelar la vista de diseño estática de un sistema.

DIAGRAMA DE SECUENCIA: Muestra la interacción de un conjunto de objetos mediante mensajes en una aplicación a través del tiempo y se modela para cada método de la clase.

DOMINIO: Sector económico donde se presentan problemas que las aplicaciones resuelven. Un sector económico puede ser dado por la actividad económica a la cual se dedica la organización.

ENTRADA: Mantiene un registro de cambios en un depósito. Acumula el valor de las transacciones realizadas para luego registrarlas en un depósito.

FACTURA: Documento que refleja la entrega de un producto o la provisión de un servicio, junto a la fecha de atención, además de indicar la cantidad a pagar como contraprestación.

FACTURA TEMPORAL: Información correspondiente a la venta que se guarda en RAM hasta que se haga la ejecución de la venta

INGENIERÍA DE DOMINIO: Proceso empleado para el desarrollo de aplicaciones para una familia de sistemas similares.

MENSAJE: Especificación de una comunicación entre objetos que transmite información con la expectativa de que se desencadenará actividad.

MÉTODO: Implementación de una operación.

MODELO: Simplificación de la realidad, creada para comprender mejor el sistema que se está creando.

MORENA: Modelo de Representación de Patrones Aspectuales de Análisis.

NIVEL: Representa el valor actual del depósito, es el efecto neto de todas las entradas vinculadas al depósito.

OBJETO: Manifestación concreta de una abstracción; entidad con unos límites bien definidos e identidad que encapsula estado y comportamiento; instancia de una clase.

OPERACIÓN: Implementación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre un comportamiento.

PARÁMETRO: Especificación de una variable que puede cambiarse, pasarse o ser devuelta.

PATRÓN: Activo de software que ha sido útil en un contexto práctico y probablemente será útil en los demás.

PATRÓN DE ANÁLISIS: Aquel patrón que es aplicable durante la fase de análisis de un sistema.

POS: (Point of Sale) ó Punto de Venta. Hace referencia al lugar dentro de una empresa o negocio en donde se efectúa el proceso de salida y cobro de la mercancía o productos que ofrece

PROCESO DE VENTA: En el proceso de venta se identifican tres tiempos, a saber: identificación y registro de productos, en donde se van enlistando los productos en la factura temporal, actualizando el total de la venta; pago, transacción económica que depende de la forma de pago; ejecución de venta, se hace persistencia de la factura temporal y se envía la información a los demás sistemas relacionados a través de la gestión de interfaces.

RELACIÓN: Conexión semántica entre elementos.

REQUISITO: Característica, propiedad o comportamiento deseado de un sistema.

ROL: Describe las propiedades estructurales y de comportamiento de un objeto en un contexto.

TEJEDOR (WEAVER): Encargado principal de tejer los diferentes mecanismos de abstracción y composición que aparecen tanto en los lenguajes de aspectos como en los lenguajes de componentes.

TEMPLATE: Elemento parametrizado, en UML es denotado con un cuadro punteado a lado superior derecho dentro del cual de describen los parámetros del Template.

THEME: Característica, concern, o requisito de interés que tiene que ser provisto para el sistema. Son constructores conceptuales y de diseño propios de la aproximación Theme/UML.

THEME/UML: Extensión de UML que enfatiza en apoyar al diseñador en la especificación de Themes que se traslapan unos con otros, y en las capacidades de composición, que están definidas.

TRANSACCIÓN: Todo proceso que genera intercambio de información entre depósitos de información. Puede estar compuesta por otras transacciones.

UML: Unified Modeling Language. Lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software.

VALIDACIÓN: Construcción de un modelo correcto; es el proceso de determinar si el modelo, como abstracción, es una buena representación para resolver los problemas que lo fundamentaron.

VENTA: Transferencia de un producto o servicio a un comprador mediante el pago de un precio convenido. Conjunto de transacciones.

VINCULAR: Indica que un ítem, cliente, cajero, caja, empresa o pago se agrega a la lista de la factura temporal.

INTRODUCCIÓN

El avance en la Ingeniería de Software se evidencia en la generación constante de nuevas propuestas metodológicas, técnicas y estrategias, como el Desarrollo de Software Orientado por Aspectos o en inglés Aspect Oriented Software Development – AOSD, la aplicación de patrones y la reutilización de artefactos software, las cuales buscan mejorar el proceso de desarrollo y la producción de software de calidad que responda a las exigencias organizacionales y sociales.

La reutilización de software surge como la estrategia que mediante el uso de cualquier tipo de artefacto creado con anterioridad (p. ej., requisitos, patrones, reglas de negocio, modelos, etc.), permite construir aplicaciones y sistemas de una manera más eficiente, económica, productiva y rápida.

La aplicación de patrones en las diferentes etapas del ciclo de vida, permite incrementar la calidad del producto, disminuir la complejidad del proceso y reducir el costo y el tiempo del desarrollo. Un patrón es un estándar o estilo de solucionar problemas que se presentan de forma recurrente en diferentes contextos.

Los patrones de análisis de software surgen como una estrategia de modelado que busca precisar comportamientos abstractos que pueden ser aplicados en diferentes contextos y de esta manera lograr mayor productividad en el desarrollo y específicamente en la fase de análisis de software.

La construcción de catálogos de patrones de análisis requiere de la definición de un esquema para la gestión de los mismos, el cual incluye el Modelo de Representación de Patrones Aspectuales de Análisis – MORENA.

Los patrones aspectuales de análisis en MORENA son activos que los desarrolladores pueden refinar en sus aplicaciones especificas. Los desarrolladores realizan su trabajo a partir de requisitos de software a nivel del dominio para producir una especificación de software igualmente genérica que puede ser adaptada a las aplicaciones particulares en áreas de negocio y en organizaciones específicas.

La importancia de validar MORENA, radica en garantizar su eficiencia y eficacia a la hora de representar patrones aspectuales de análisis, demostrando que es útil y aplicable dentro del proceso de desarrollo de software. Igualmente, es necesario determinar que este modelo sea correcto antes de realizar las fases de implementación y aplicación, para evitar que las posibles fallas sean heredadas.

En este trabajo se presenta la validación de MORENA a partir de la construcción de TRIPODE: un catálogo de patrones aspectuales de análisis para el dominio POS basado en MORENA, con el fin de verificar el cumplimiento de los objetivos que el modelo plantea. El documento se organiza de la siguiente manera: en el primer capítulo se realiza la descripción del problema, en el segundo capítulo se definen los objetivos tanto general como específicos, en el tercer capítulo se describe el marco teórico estudiado con fines de la realización del proyecto; en el cuarto capítulo se presentan los elementos, la estrategia y el plan de validación; en el quinto capítulo se especifica el dominio de aplicación POS (Point of Sale); con estos elementos, en el sexto capítulo se construye TRIPODE; a partir de los resultados de los procesos anteriores, en el séptimo capítulo se desarrolla el plan de mejoramiento de MORENA que permitirá retroalimentar y optimizar el modelo; por último, se presentan las conclusiones del proyecto y trabajos futuros.

El grupo de trabajo que desarrolla esta investigación hace parte del grupo de investigación Galeras Línea Ingeniería del Software de la Universidad de Nariño, como parte del proyecto: Un esquema para la gestión de catálogos de patrones de análisis - ESKEMA, que está integrado en el proyecto: MMEDUSA - Marco Metodológico para Desarrollo de Aplicaciones Utilizando la Aproximación de Aspectos, desarrollado con la participación del grupo de investigación en Ingeniería del software de la Universidad EAFIT y la empresa AVANSOFT S.A.

1. DESCRIPCIÓN DEL PROBLEMA

1.1. PLANTEAMIENTO DEL PROBLEMA

El paradigma objetual cuenta con gran acogida en la comunidad de desarrollo de software. Pese a la madurez demostrada por el desarrollo de software orientado a objetos, existen nuevos retos planteados por la ingeniería del software frente a los cuales este paradigma es insuficiente.

Como Constantinides, Elrad y Fayad[1] afirman, los problemas de la orientación por objetos se evidencian en la dispersión y enmarañamiento de intereses y en su incapacidad para lograr completamente la separación de los mismos. Estos fenómenos dificultan la trazabilidad del sistema e impiden analizar el impacto y evolución del cambio, al tiempo que se aumenta la complejidad.

La problemática descrita anteriormente la aborda el AOSD y para especificar los mecanismos de su aplicación, el grupo de investigación en ingeniería del software de la universidad EAFIT en convenio con la empresa AVANSOFT S.A. desarrolla el proyecto *MMEDUSA*[2]. El objetivo general del proyecto es proponer un marco metodológico para la aplicación de aspectos en el proceso de desarrollo de software y validar dicha propuesta en un caso de estudio real.

Con el desarrollo de este marco metodológico surgen diversas áreas de acción, entre ellas el uso de catálogos para la reutilización de componentes, lo que genera la necesidad de un esquema para la gestión de catálogos de patrones de análisis ESKEMA, propuesto por Barón[3], el cual incluye MORENA.

MORENA plantea la definición de los patrones de análisis a través de tres modelos: estructural, de alcance y de comportamiento; que son representados por diagrama de clases, colaboración y diagrama de secuencia respectivamente. Los modelos se encapsulan en un constructor de la especificación UML 2.0 llamado Template.

---

[1] CONSTANTINIDES, Constantinos, ELRAD, Tzilla, y FAYAD, Mohamed. Extending the object model to provide explicit support for crosscutting concerns. s.l: John Wiley & Sons, 2002

[2] ANAYA, Raquel. "Presentación de resultados proyecto MMEDUSA: Marco Metodológico para Desarrollo de Aplicaciones Utilizando la Aproximación de Aspectos", 2008

[3] BARÓN, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

El objetivo del presente proyecto es validar MORENA a partir de la construcción de TRIPODE, esto provee el escenario apropiado para validar el modelo y generar propuestas de mejoramiento que lo retroalimenten.

Para validar MORENA es necesario seguir el proceso de ingeniería de dominio que plantea ESKEMA para la creación del catálogo de patrones de análisis[4].

El problema que aborda esta investigación es la validación de MORENA, proceso que incluye identificar los elementos, definir la estrategia y aplicarla de tal manera que permita retroalimentar el modelo para su mejoramiento.

1.2. FORMULACIÓN DEL PROBLEMA

¿De qué manera se puede comprobar que MORENA integra los elementos que lo describen en las condiciones que su definición plantea y que su aplicación en la construcción de un catálogo de patrones aspectuales de análisis es funcional?

1.3. SISTEMATIZACIÓN DEL PROBLEMA

- ¿Cuáles son los elementos del modelo de representación de patrones aspectuales de análisis - MORENA, que deben ser validados?
- ¿Cuáles son las actividades que se deben desarrollar para validar el modelo de representación de patrones aspectuales de análisis - MORENA?
- ¿De qué manera se puede determinar las características y condiciones específicas del dominio de aplicación POS como espacio de validación del modelo de representación de patrones aspectuales de análisis - MORENA?
- ¿Cómo ejecutar la estrategia de validación en el dominio de aplicación POS del modelo de representación de patrones aspectuales de análisis - MORENA?
- ¿Cómo presentar los resultados de la validación del modelo de representación de patrones aspectuales de análisis - MORENA?

[4] Ibid

1.4. ALCANCE

El presente trabajo se centra en la validación de MORENA en el contexto del dominio de aplicación POS, aplicando una estrategia de validación que permita presentar sugerencias para la optimización del modelo. El proceso de validación de MORENA se hace a partir de su aplicación en la construcción de TRIPODE.

La implementación y la aplicación del catálogo son el objetivo de trabajos futuros de investigación.

1.5. JUSTIFICACIÓN

Integrar las fortalezas del AOSD a las potencialidades de los patrones de análisis en un catálogo de artefactos, permite capitalizar el tiempo invertido en la fase de requisitos, al generarse un valor ganado para proyectos futuros que hacen uso del catálogo. Los patrones de análisis, permiten a los desarrolladores de software agilizar el proceso, incorporando resultados positivos de experiencias anteriores aplicadas en un contexto específico que pueden ser adoptadas en otros contextos. Su uso en software es muy importante porque permite integrar el conocimiento y la experiencia de muchos desarrolladores, según Fernandez[5].

Contrario a los modelos genéricos que no guían al diseñador en la manera de aplicarlos a diferentes contextos, el catálogo de patrones de análisis provee modelos que pueden ser adaptados a cualquier contexto. Las facilidades ofrecidas por un catálogo de patrones de análisis siguiendo un enfoque aspectual, permiten introducir la reutilización en las tareas del desarrollador, especificando de manera precisa la forma como este comportamiento genérico puede ser tejido con la funcionalidad base.

La construcción de catálogos de patrones de análisis requiere de la definición de un esquema para la gestión de los mismos. El esquema es propuesto en ESKEMA[6], el cual incluye MORENA.

El escenario más apropiado para la validación de MORENA, es su aplicación en la construcción de un catálogo de patrones aspectuales de análisis; la presente investigación propone la construcción de TRIPODE, que permitirá verificar la

[5] FERNANDEZ, Eduardo B., “Building Systems Using Analysis Patterns”, Florida Atlantic University, 2005.

[6] BARÓN, Op. Cit.

eficiencia de MORENA en la construcción de un catálogo de patrones de análisis en el contexto aspectual.

La efectiva validación de MORENA a partir de la construcción de TRIPODE permitirá:

- Verificar que MORENA cumple con los objetivos planteados.
- Garantizar la eficiencia y eficacia de MORENA.
- Demostrar la utilidad y aplicabilidad de MORENA en la construcción de catálogos de patrones de análisis en el contexto aspectual.
- Evitar que las deficiencias de MORENA, identificadas en la etapa de definición, trasciendan a actividades posteriores como la implementación y aplicación.
- Evitar el incremento del tiempo y esfuerzos invertidos en la construcción de MORENA.
- Mejorar MORENA a partir de la identificación de las oportunidades de mejoramiento.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

Validar el modelo de representación de patrones aspectuales de análisis - MORENA a partir de la construcción de TRIPODE: un catálogo de patrones aspectuales de análisis para el dominio POS basado en MORENA.

2.2. OBJETIVOS ESPECIFICOS

- Determinar los elementos de validación del modelo de representación de patrones aspectuales de análisis - MORENA
- Definir la estrategia a utilizar para la validación del modelo de representación de patrones aspectuales de análisis - MORENA
- Especificar el dominio de aplicación POS en el cual se validará el modelo de representación de patrones aspectuales de análisis - MORENA
- Construir el catálogo de patrones aspectuales de análisis basado en MORENA para el dominio de aplicación POS.
- Elaborar el plan de mejoramiento del modelo de representación de patrones aspectuales de análisis - MORENA a partir del proceso de validación.

3. MARCO TEÓRICO

3.1. INTRODUCCIÓN

En este capítulo se presentan los elementos teóricos requeridos para el desarrollo de la investigación. Estos elementos son: ingeniería de dominio, catálogos de activos de software, patrones de software incluidos los patrones de análisis, AOSD, lenguaje unificado de modelado UML, THEME/UML, ESKEMA, MORENA y por último, validación de modelos.

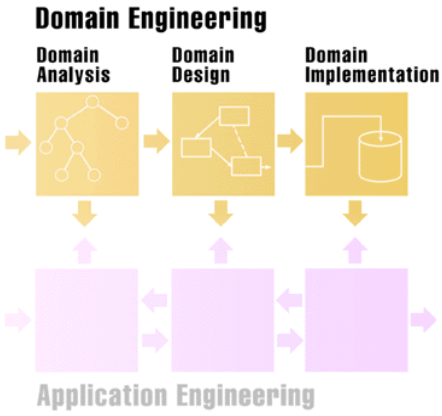
3.2. INGENIERÍA DE DOMINIO

El Software Engineering Institute presenta la siguiente definición:

> La ingeniería de dominio es el proceso empleado para el desarrollo de aplicaciones para una familia de sistemas similares. La Ingeniería de dominio incluye todas las actividades requeridas para la construcción de los activos principales del software. Estas actividades son: el análisis de dominio (identificación de uno o más dominios, capturar la variación dentro de un dominio), el diseño de dominio (construcción de un diseño adaptable) y la implementación del dominio (definir los mecanismos para trasladar los requisitos a sistemas creados para la reutilización). Los productos o activos de software de estas actividades son: modelo del dominio, modelo de diseño del dominio, lenguajes de dominio específico, código generado y componentes de código[7].

[7] Software Engineering Institute. Internet: (www.sei.cmu.edu/domain-engineering/domain_anal.html <http://www.sei.cmu.edu/domain-engineering/domain_anal.html>)

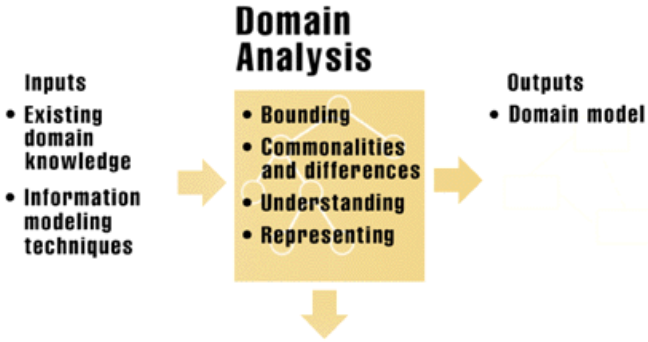
Figura 1: Proceso de ingeniería de dominio



Software Engineering Institute. Internet: (www.sei.cmu.edu/domain-engineering/domain_anal.html)

3.2.1. Análisis de dominio. "El análisis de dominio es el proceso de identificación, colección, organización y representación de la información relevante en un dominio, basado en el estudio de sistemas existentes y sus desarrollos históricos, conocimiento capturado de expertos del dominio, la teoría relacionada y la tecnología emergente dentro de un dominio[8]".

Figura 2: Proceso de análisis de dominio



Software Engineering Institute. Internet: (www.sei.cmu.edu/domain-engineering/domain_anal.html)

El Análisis de dominio describe detalladamente el dominio estudiado, considerando los elementos comunes y los elementos diferenciales, de los sistemas en el dominio, permite la organización y la comprensión de las relaciones

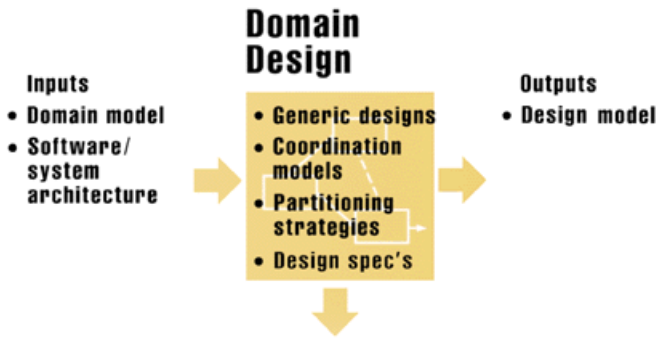
[8] Ibid.

entre los elementos del dominio y representa la información recolectada de manera que sea útil para el desarrollo de la aplicación actual y de las aplicaciones futuras que pertenezcan a la misma familia o dominio.

En la actualidad, existen numerosas técnicas de análisis de dominio. Cada una se centra en incrementar el nivel de comprensión del dominio, capturando la información y representándola en modelos formales.

3.2.2. Diseño de dominio. Actualmente muchos investigadores están estudiando y clasificando arquitecturas de software, con el fin de identificar estilos de arquitectura y patrones de sistemas que permitan la reutilización en el desarrollo de nuevas aplicaciones dentro de un dominio.

Figura 3: Proceso de diseño de dominio



Software Engineering Institute. Internet: (www.sei.cmu.edu/domain-engineering/domain_anal.html)

"El diseño de dominio es el proceso de desarrollo de un modelo de diseño a partir de los productos del análisis del dominio y del conocimiento adquirido a partir del estudio del reuso de los requisitos, diseños y arquitecturas genéricas de software de software[9]".

La clave para la reutilización, es el desarrollo y documentación de arquitecturas genéricas para las aplicaciones de un dominio. El modelo de diseño representa la arquitectura genérica desarrollada para el dominio analizado y provee el marco de trabajo para el desarrollo de los componentes reutilizables en la implementación del dominio.

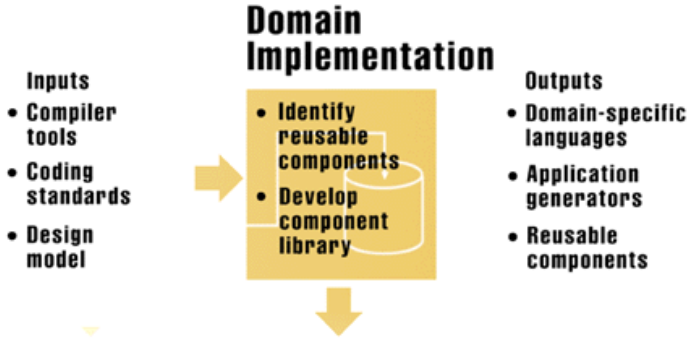
La arquitectura genérica es representada por un estilo arquitectónico que es el más apropiado para la familia de productos. Esta arquitectura define la plataforma

[9] Ibid

computacional para todas las aplicaciones de la familia y provee las pautas para la construcción de un diseño genérico. El resultado es una estrategia de partición para la asignación de características del dominio a elementos del software y un modelo de coordinación que describe como los elementos son activados y como comparten información.

3.2.3. Implementación de dominio. La creación de componentes reutilizables ha cobrado gran trascendencia en la comunidad de ingeniería de software actual, principalmente con el uso de librerías de componentes de código. Muchos de estos repositorios no son el resultado de esfuerzos en torno a la ingeniería de dominio, por tanto, activos reutilizables de alto nivel tales como modelos de dominio o arquitecturas de referencia. El trabajo en el área de librerías de componentes tiende a centrarse en cuestiones de cualificación, clasificación, recuperación y retiro de componentes.

Figura 4: Proceso de implementación de dominio



Software Engineering Institute. Internet: (www.sei.cmu.edu/domain-engineering/domain_anal.html)

El Software Engineering Institute afirma lo siguiente:

> La implementación del dominio es el proceso de identificar componentes reutilizables basados en el modelo del dominio y una arquitectura genérica. Usando el conocimiento obtenido durante el análisis del dominio, y la arquitectura genérica desarrollada durante el diseño del dominio, los ingenieros del dominio adquieren y, donde sea necesario, crean los activos reutilizables que son catalogados en una librería de componentes para el uso de los ingenieros de aplicación. Estos componentes reutilizables, así como los generadores de aplicación y los lenguajes de dominio son los

principales productos de esta fase en la ingeniería de dominio. La creación, gestión, y mantenimiento de un repositorio de activos reutilizables, son también tareas de las cuales se encarga la implementación de dominio[10].

3.3. CATÁLOGOS DE ACTIVOS DE SOFTWARE

Los catálogos han sido definidos de diversas formas, aquí se presentan algunas de ellas:

- "El catálogo es una lista en la que se registran, describen y ordenan, siguiendo determinadas normas, personas, cosas o sucesos que tienen algún punto en común"[11].

- El catálogo está diseñado para potenciar la reutilización de activos prefabricados en el desarrollo de nuevas aplicaciones

- El catálogo de activos se desarrolla principalmente debido a la necesidad de almacenar y recuperar la definición e implementación de estos y para automatizar el proceso de búsqueda de los mismos.

- "Para facilitar la aplicación de los activos, éstos se compilan, organizan y relacionan en lo que se conoce como catálogo de activos de software"[12].

De lo anterior, se puede deducir que un catálogo es una colección de activos de software en la cual éstos se compilan, organizan y relacionan, siguiendo determinadas normas; dotándoles de alguna estructura u organización para facilitar su selección.

Los activos de software que pueden contener los catálogos son elementos reutilizables de software que ayuden a detectar, describir y compartir las soluciones para los problemas que se presentan en el desarrollo de software. Estos elementos pueden ser: componentes, aspectos, patrones, concerns, etc.

Para la construcción de un catálogo de activos de software se debe establecer una estructura jerárquica similar a la de un sistema de ficheros, que puede ser determinada por el tipo de activos de software que este contendrá, además se

[10] Ibid.

[11] The free Dictionary. Internet: (es.thefreedictionary.com/cat%C3%A1logo http://es.thefreedictionary.com/cat%C3%A1logo)

[12] FOWLER, Martin. Analysis Patterns, Reusable objects models, Addison Wesley, 1997

debe implementar un conjunto de reglas que permitan realizar operaciones como: búsqueda, eliminación, adiciones y modificaciones sobre estos activos.

3.4. PATRONES DE SOFTWARE

Los desarrolladores de software, normalmente se enfrentan a problemas que ocurren reiteradamente. En estos casos, contar con algún instrumento que ayude a detectar, describir y compartir las soluciones más apropiadas para estos problemas sería de una gran utilidad. Los patrones de software desempeñan dicha misión en el marco de la ingeniería del software.

Antes de la aparición de los patrones de software, la reutilización se había centrado en el código. Los patrones permiten la reutilización del conocimiento empírico que implica la resolución de los problemas que reiteradamente se presentan en cada etapa del desarrollo de software.

Para obtener las ventajas que genera la reutilización a partir de los patrones de software, es primordial integrarlos a los procesos de desarrollo de software. Para ello, es necesario considerar los patrones como elementos de modelado de primer orden, representarlos y describirlos de manera adecuada, y contar con un catálogo de patrones que abarque todo el proceso y facilite la selección del más apropiado en cada momento.

3.4.1. Definición. Aunque el concepto de patrón puede ser más o menos intuitivo, ciertamente es difícil encontrar una definición comúnmente aceptada para el término patrón de software. Esto es así debido a que existen muchas comunidades interesadas en su estudio y con perspectivas muy diferentes, las cuales dependen fundamentalmente de su ámbito de aplicación.

Para nuestro caso retomamos una definición genérica propuesta por Martin Fowler[13]: “una idea que ha sido útil en un contexto práctico y probablemente será útil en los demás”. El termino idea se emplea para poner de manifiesto el hecho que un patrón puede ser cualquier cosa. Puede ser un grupo de objetos que colaboran, como los patrones del "Grupo de los Cuatro[14]" o los principios para el proyecto de organización de Coplien[15]. La frase contexto práctico refleja el hecho

[13] Ibid.

[14] GAMMA, E., R. HELM, R., Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Objects-Oriented Software Reading, MA: Addison Weley, 1995.

[15] COPLIEN, J.O. “Agenerative Development-process Pattern Language”. In Pattern Language of program design. J.O. Coplien and D.C. Schmidt, ED. Reading, MA:Addison-Weley, 1995.

que los patrones son desarrollados a partir de la experiencia práctica de un proyecto real.

A menudo se dice que los patrones son mas descubiertos que inventados. Esto es cierto en el sentido de que los modelos se convierten en patrones sólo cuando se da cuenta que pueden tener una cierta utilidad común. No todas las ideas de un proyecto son patrones; patrones son esas cosas que los desarrolladores piensan que pueden ser útiles en otros contextos.

3.4.2. Clasificación. Hay muchas formas posibles de clasificar los patrones, prácticamente tantas como tipos diferentes de problemas podemos encontrar a lo largo del ciclo de vida del software. No obstante, existen algunas taxonomías ampliamente conocidas que merece la pena comentar.

En primer lugar, se puede hablar de la clasificación realizada por Buschmann et al[16]. En ésta se diferencian tres tipos relacionados de patrones:

- **Patrones de arquitectura.** Expresan un esquema u organización básica de la estructura de un sistema software. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y pautas para organizar las relaciones entre ellos.

- **Patrones de diseño.** Proporcionan un esquema para refinar los subsistemas o componentes de un sistema software, o las relaciones entre ellos. Describen estructuras recurrentes de componentes que se comunican y resuelven un problema de diseño dado en un contexto particular.

- **Patrones de código (Idioms).** Son los patrones de más bajo nivel, específicos para un lenguaje de programación determinado. Un "idiom" describe cómo implementar aspectos de componentes o de relaciones entre ellos usando las características de un lenguaje dado[17].

Otra importante categorización en la que, hasta donde se sabe, se introduce por primera vez el término patrón conceptual, es la que realizan Riehle y Züllighoven[18]. Aquí los autores distinguen entre:

---

[16] BUSCHMANN, Frank, MEUNIER, Regine, ROHNERT, Hans, SOMMERLAD, Peter, STAL, Michael. (1996). Pattern-oriented software architecture: A system of patterns. Chichester, UK: John Wiley & Sons.

[17] COPLIEN, Op. cit.

[18] RIEHLE, Dirk. ZÜLLIGHOVEN, Heinz. (1996). Understanding and using patterns in software development. Theory and Practice of Object Systems.

- **Patrones conceptuales.** Usados para facilitar la construcción del modelo conceptual de un sistema. Se describen por medio de términos y conceptos de un dominio de aplicación particular (dominio del problema). Guían la percepción que se tiene de un dominio de aplicación y ayudan tanto a comprenderlo como a describirlo. Según los autores, éstos deberían apoyarse en metáforas del dominio de aplicación.

- **Patrones de diseño.** Usados en la construcción del modelo de diseño. Emplean conceptos pertenecientes al dominio de la solución. Varían su escala desde "macro-arquitecturas", como pueden ser los patrones de arquitectura según Buschmann et al[19], hasta "microarquitecturas", como los de Gamma et al[20]. Los autores sostienen que el paso del modelo conceptual al modelo de diseño se puede facilitar si es posible establecer una relación clara entre patrones de diseño y patrones conceptuales.

- **Patrones de programación.** Descrito por medio de constructores pertenecientes a algún lenguaje de programación concreto. Similar a los patrones de código o idioms. Asimismo, si éstos se relacionan con los de diseño, la etapa de implementación se podría agilizar.

En consonancia con la idea de patrón conceptual, Fowler[21] acuña el término patrón de análisis para referirse a aquellos patrones que son aplicables durante la fase de análisis de un sistema.
No obstante, los patrones también pueden ser clasificados en virtud de su dominio de aplicación, dando lugar, como es lógico, a una clasificación mucho más extensa. Aunque hay patrones que son de carácter general, esto es, independientes del dominio, muchos otros son creados para resolver problemas en ámbitos concretos de aplicación.

Otra interesante tipificación es la que atiende a la etapa o aspecto que se considera dentro del proceso de desarrollo. La estructuración de un catálogo a partir de este criterio podría facilitar en gran medida la selección y aplicación del patrón más adecuado en cada instante del proceso.

3.4.3. Descripción. Los patrones van más allá de la mera exposición de una solución para un problema conocido. Un patrón debe reflejar convenientemente las razones por las cuales puede interesar su uso. Además, para facilitar su

---

[19] BUSCHMANN, Op. cit.

[20] GAMMA, Op. cit.

[21] FOWLER, Op. cit.

aplicación y comparación, la descripción debería ser realizada de una manera comprensible, clara y estructurada.

Meszaros y Doble[22], presentaron un original e interesante trabajo sobre las recomendaciones que se deben seguir para la descripción de patrones. A través de un lenguaje de patrones, los autores describen y ayudan a aplicar las mejores prácticas utilizadas para su escritura. Según estos autores, la descripción de un patrón está formada por una serie de elementos obligatorios, junto con otros opcionales. Éstos se materializan en un conjunto de secciones que dotan de estructura la descripción. Así, los aspectos esenciales que deberían contemplarse son: nombre, contexto, problema, fuerzas, solución.

Entre los elementos considerados opcionales, los cuales pueden ser más o menos útiles dependiendo del patrón concreto que se describe, están: indicios, contexto resultante, patrones relacionados, ejemplos, código ejemplo, razón, alias, agradecimientos.

Coincidiendo con Riehle y Züllighoven[23], se piensa que la descripción de un patrón debería adaptarse a su tipo y dominio de aplicación específico, presentando de forma explícita sólo aquellas secciones que ayuden realmente a usar y entender mejor el patrón, y eliminando aquellas que se consideren superfluas o redundantes.

Uno de los formatos de descripción más famosos, conocido como formato del GoF, es el que proporciona Gamma et al[24]. Su estructura cuenta con las siguientes secciones: nombre, clasificación, intención, alias, motivación, aplicabilidad, estructura, participantes, colaboraciones, consecuencias, implementación, código ejemplo, usos conocidos y patrones relacionados.

En términos generales, la mayoría de los formatos empleados para escribir patrones constan de cuatro partes esenciales: una declaración del contexto en el que el patrón es útil, el problema que el patrón soluciona, las fuerzas que participan en la solución, y la solución que resuelve el problema. Este formato es conocido como el formato fijo Contexto-Problema-Fuerza-Solución. La estructura de este formato apoya la definición de un patrón como "una solución a un problema en su contexto". Existe otro formato más simple que incluye únicamente el par problema-solución.

---

[22] MESZAROS, Gerard; DOBLE, Jim (1998). A Pattern Language for Pattern Writing, Addison-Wesley

[23] RIEHLE, Op. cit.

[24] GAMMA, Op. cit.

Para muchos analistas el uso de un formato fijo, es un determinador de un patrón. El uso de un formato de patrón aceptado claramente marca al patrón como algo diferente a un objeto de escritura técnica.

El formato con el cual se deben escribir los patrones es un debate abierto en la comunidad de ingeniería del software, sin embargo, hay consenso en que los patrones deben ser nombrados. Una de las ventajas de trabajar con patrones es la forma en que puede enriquecer el vocabulario del desarrollo y comunicar ideas de diseño muy efectivamente a través de las nomenclaturas.

3.4.4. Integración en los procesos de software. Desarrollar software es un proceso indudablemente complejo. Esto es así por varias razones:

- El software es complejo por naturaleza.
- Normalmente es necesario trabajar en grupo.
- Suelen existir problemas de comunicación con los usuarios, incluso entre los propios miembros del equipo de desarrollo.
- El software evoluciona y se debe facilitar su mantenimiento.
- Los proyectos pueden tener un tamaño considerable.

Al mismo tiempo hay un creciente interés por construir software fiable, gastando la menor cantidad de tiempo y dinero posible. La ingeniería del software pretende precisamente satisfacer dicha demanda, estableciendo y aplicando principios y métodos propios de la ingeniería. La reutilización de software es una de las actividades encaminadas a la consecución de dicho objetivo.

Tradicionalmente esta reutilización ha estado basada fundamentalmente en el código. Sin embargo, otros tipos de artefactos (especificaciones y modelos de requisitos, diseños, documentación, etc.), no han sido reutilizados sistemáticamente; principalmente sabiendo que la construcción de éstos repercute enormemente sobre la calidad del producto final y el tiempo de desarrollo. La intención de los patrones es dar un paso más allá a través de la reutilización sistemática del conocimiento experto empleado en la creación de todos estos artefactos.

Aparte de la ventaja que supone la reutilización del conocimiento que el experto emplea para afrontar más ágilmente los problemas que se encuentra durante la construcción de todos estos artefactos, el equipo de desarrollo se beneficia, por otro lado, de la posibilidad de compartir un vocabulario común, compuesto por los nombres de los patrones, que favorece el entendimiento entre ellos y con los

usuarios finales. Adicionalmente, la inclusión de patrones mejora la documentación y, por ende, los modelos que se generan, desde el punto de vista de su legibilidad, comprensión y mantenimiento.

En definitiva, el desarrollo de software en base a patrones permite aliviar la complejidad intrínseca de su desarrollo y contribuye en la obtención de software más rentable y de mejor calidad.

3.4.5. Patrones de análisis. Durante las etapas tempranas, los desarrolladores elaboran modelos conceptuales que ayudan a entender los problemas que aparecen en el dominio de aplicación. Para agilizar y mejorar la construcción de estos modelos, es posible aplicar un tipo de patrones, conocidos como patrones de análisis, cuyo objetivo es facilitar la comprensión, comunicación y reutilización de aquellas abstracciones claves recurrentes en un determinado dominio del problema.

Fowler[25], acuña el término patrón de análisis para referirse a aquellos patrones que son aplicables durante la fase de análisis de un sistema. Los patrones de análisis capturan modelos conceptuales, vinculados a un dominio de aplicación concreto, que pueden ser reutilizados en distintas aplicaciones (reutilización del conocimiento del dominio). Su utilización tiene un objetivo doble, por una parte, acelerar el desarrollo de los modelos de análisis abstractos que capturan los principales requisitos del problema, y por otra, facilitar la transformación del modelo de análisis en el modelo de diseño.

- **Aplicación de patrones de análisis.** Entre los problemas más complejos y decisivos que es necesario abordar durante el desarrollo de un sistema software están la obtención, análisis y especificación de sus requisitos.

  A lo largo de este proceso, el analista desarrolla uno o varios modelos, denominados modelos de análisis o conceptuales, que ayudan a entender y simplificar los problemas que aparecen en el dominio de aplicación. Éstos complementan la especificación de requisitos realizada en lenguaje natural, representando de una forma más técnica y detallada, el contexto, estructura y comportamiento del sistema.

  Estos modelos tienen un gran valor, principalmente debido a que, por un lado, son una abstracción que representa la visión (modelo mental) que tiene el analista sobre el dominio de aplicación, y por otro, son usados como puente entre el análisis y el diseño del sistema a desarrollar. Por ello, es de máxima

[25] FOWLER, Op. cit.

importancia que dichos modelos estén bien construidos y sean válidos, es decir, que definan correctamente el sistema que desea el cliente.

Una prometedora técnica que permite acelerar y mejorar el desarrollo de estos modelos, a la vez que ayuda a reducir el número de errores cometidos, consiste en la aplicación de patrones de análisis.

Según Martin Fowler[26], "los patrones de análisis son grupos de conceptos que representan una construcción común en el modelado del negocio. Éstos pueden ser relevantes para un sólo dominio, o pueden abarcar muchos dominios".

Los patrones de análisis son también ampliamente conocidos como patrones conceptuales. Riehle y Züllighoven[27], introdujeron este término para referirse a aquellos patrones aplicables en la construcción de un modelo conceptual para un dominio de aplicación concreto.

Claramente, los patrones conceptuales están centrados en el qué (dominio del problema) y no en el cómo (dominio de la solución). Representan y documentan escenarios comunes que aparecen en el dominio que comparten

- **Beneficios que aporta la reutilización de patrones de análisis.** Las ventajas que supone la aplicación de patrones de análisis durante las etapas tempranas de modelado de un sistema software se pueden resumir en las siguientes:

  - ✓ Guían la percepción que se tiene de un dominio de aplicación y ayudan a encontrar, comprender y describir los problemas comunes que aparecen dentro de éste.

  - ✓ Proporcionan un lenguaje de comunicación o lengua franca que facilita el entendimiento entre el personal técnico y las personas relacionadas con el dominio del problema.

  - ✓ "Optimizan el análisis por medio de la reutilización de modelos conceptuales que ya han demostrado su validez en dominios semejantes, mejorando la calidad del software producido. Los modeladores sólo tienen que modificar y combinar estructuras existentes, más que crear nuevas desde cero"[28]

[26] FOWLER, Op. cit.

[27] RIEHLE, Op.cit.

[28] HAY, David. Data Model Patterns: Conventions of Thought. New York: Dorset House, 1996

- ✓ “Simplifican la construcción de los modelos conceptuales y facilitan su validación. Los patrones ayudan a reducir o eliminar los errores de modelado más gruesos, puesto que los elementos básicos del modelo están ya definidos”[29].

- ✓ Facilitan la comprensión, comunicación, documentación y mantenimiento de los modelos generados cuando los patrones utilizados se identifican claramente dentro de éstos.

- ✓ “Reducen el tiempo de desarrollo significativamente y mejoran la interfaz entre la etapa de análisis y de diseño”[30].

A pesar de todo, los patrones no están todavía ampliamente usados durante las etapas tempranas de modelado, a diferencia de lo que ocurre en etapas posteriores.

- **Catálogo de patrones de análisis de Martín Fowler.** Es necesario señalar que el número de publicaciones relacionadas con este tema es de momento escaso, sobre todo en comparación con la atención que se ha prestado a otros tipos de patrones, como por ejemplo los de diseño. Además, los trabajos existentes están muy disgregados, abordando la aplicación temprana de patrones desde muy diversas perspectivas. Aumentar y aunar los esfuerzos en este terreno es una tarea que no podemos eludir. No obstante, se han desarrollado algunos trabajos significativos en este terreno que, a continuación, vamos a presentar y analizar. Así, por ejemplo, hay autores que han concentrado sus energías en la exploración de métodos y herramientas para dar soporte al proceso de modelado conceptual en base a patrones.

  Este es el caso de Fowler[31], quizás la obra más relevante e influyente dentro del movimiento que está relacionado con el estudio de los patrones de análisis. Este libro presenta una colección de 85 modelos de objetos del negocio reutilizables en diferentes ámbitos (relaciones organizacionales, cantidades, medidas, responsabilidades, análisis financiero, inventarios, contabilidad, planificación, comercio, etc.). No obstante, gran parte de estos patrones no son exclusivos de un sólo dominio, pudiendo ser aplicados en varios de ellos, lo que eleva aún más su capacidad de reutilización.

  Martin Fowler presenta un catálogo de patrones de análisis para grupos de dominios de negocio. Muestra patrones para: contabilidad, observaciones y

[29] Ibid.

[30] GEYER-SCHULZ, Andreas; HAHSLER, Michael (2002). Software reuse with analysis patterns.

[31] FOWLER, Op. cit.

mediciones, observaciones para las finanzas corporativas, refiriéndose a los objetos, inventario y contabilidad, planificación, comercio, contratos, paquetes de comercio.

- ✓ **Modelos conceptuales.** El modelo conceptual es la base conceptual del catálogo de patrones de análisis de Fowler[32]. El modelo conceptual es una abstracción mental que permite entender y simplificar los problemas que se presentan en las organizaciones. Un modelo conceptual es un artefacto humano que representa una situación del mundo real. Los modelos conceptuales son efectivos, en términos de ingeniería, porque nos permiten entender fácil y rápidamente lo que está pasando en el mundo real. Para representar una situación real se puede usar más de un modelo, esto implica un proceso de análisis para elegir la mejor opción, encontrando modelos que son más apropiados para determinadas circunstancias. Lo anterior ilustra un principio importante: "Los modelos no son buenos o malos; son más o menos útiles"[33].

  Para desarrollar software que cumpla con los requisitos que lo motivaron, es necesario un modelo conceptual apropiado. Normalmente el modelo más simple es el más recomendado. La elección del modelo afecta la flexibilidad y la reutilización del sistema. La tendencia es a construir modelos que provean máxima flexibilidad, esto puede ser contraproducente, ya que construir demasiada flexibilidad en un sistema puede hacerlo muy complejo, difícil de mantener y costoso, aspectos entre los cuales la ingeniería busca balance. En tal sentido los desarrolladores deben tener en cuenta que la flexibilidad debe ser añadida únicamente si es estrictamente necesaria y de frecuente uso.

3.5. DESARROLLO DE SOFTWARE ORIENTADO POR ASPECTOS - AOSD

Actualmente, el *Object Oriented Software Development* (OOSD) es el modelo más utilizado en el desarrollo de software. Este paradigma ha permitido un gran avance de la ingeniería del software, haciendo posible el desarrollo de sistemas y aplicaciones de alta complejidad. El paradigma orientado a objetos da solución adecuada al comportamiento funcional, pero la solución ofrecida para el comportamiento no funcional no es la más apropiada. El paradigma objetual

[32] Ibid.

[33] ALEXANDER, Christopher. ISHIKAWA, Sara. SILVERSTEIN, Murray. JACOBSON, Max. FIKSDAHL-KING, Ingrid. ANGEL, Shlomo. A Pattern Language. New York: Oxford University Press, 1977

permite realizar una separación adecuada del comportamiento no transversal, pero el comportamiento transversal no es tratado de manera apropiada.

El AOSD se constituye en la alternativa más visionaria para el mejoramiento del proceso de desarrollo del software, su evolución ha permitido identificar las falencias de los paradigmas precedentes y proponer estrategias técnicas de solución que se incorporan en todo el ciclo de vida del software. La orientación a aspectos se presenta como un modelo que soporta la separación de los comportamientos que representan la funcionalidad base de aquellos comportamientos que pueden ser transversales a más de un módulo. La implementación de aplicaciones de software utilizando técnicas de AOSD permite mejores estructuras de implementación que tienen impacto positivo sobre características de calidad del software tales como escalabilidad, reusabilidad y la reducción de complejidad.

3.5.1. Separación de concerns. AOSD es un paradigma de ingeniería del software que conduce a reducir la complejidad de los sistemas software a través de la separación de *concerns* o intereses. Los *concerns* son los diferentes temas o asuntos de los que es necesario ocuparse para resolver el problema. Por ejemplo, una función específica que debe realizar una aplicación, pero también surgen otras como por ejemplo: distribución, persistencia, replicación, sincronización, etc. Para Dijkstra: "La separación de *concerns* se refiere a la habilidad para identificar, encapsular y manipular de forma aislada aquellas partes del software que son relevantes para un concepto dado, objetivo o propósito"[34], centrada en un principio en la etapa de implementación y posteriormente aplicada a todas las etapas del ciclo de vida del software.

Una de las corrientes más relevantes dentro de AOSD es MDSOC[35] (*Multidimensional Separation of Concerns*) que promueve la encapsulación de diferentes tipos de *concerns* organizados en dimensiones y la integración de las mismas para formar el sistema completo. Cada una de estas dimensiones está compuesta por *concerns* del mismo tipo encapsulados de forma independiente.
Estas piezas separadas (artefactos que encapsulan *concerns*) serán integradas a través de una serie de reglas de composición para formar el sistema completo. Todas las dimensiones son tratadas de forma arbitraria, posibilitando que un desarrollador se centre en determinadas características del software sin tener que conocer las demás. Por *concerns* del mismo tipo se pueden entender propiedades que descomponen el software atendiendo a un determinado criterio.

[34] DIJKSTRA, Edsger W., A Discipline of Programming, Prentice-Hall,1976.

[35] OSSHER, H. TARR, P.: "Multi-Dimensional Separation of Concerns and The Hyperspace Approach." In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. Kluwer, 2000.

3.5.2. Crosscutting concerns. Son *concerns* cuya funcionalidad afecta a varios módulos ya definidos. Estos *concerns* requerirán su implementación en muchas clases o módulos, sin la técnica apropiada, produce un código enmarañado (*tangling*) y/o mezclado (*scattering*), el cual será difícil de modificar y entender.

Se entiende el *scattering,* como la necesidad de diseminar un conjunto de requisitos a través de muchos componentes del sistema. El *tangling*, por el contrario, consiste en la necesidad de hacer que un sólo componente del sistema tenga que realizar todo un conjunto de requisitos.

Estos dos fenómenos, que se hacen presentes en el paradigma orientado a objetos, dificultan la reutilización, aumentan el acoplamiento entre componentes al tiempo que aumentan la complejidad del sistema y hacen que el software pierda capacidad para evolucionar.

En pocas palabras, *tangling* y *scattering*, atentan contra los atributos de calidad a los que apunta la ingeniería de software: aumentar la calidad del software, reducir sus costos y facilitar su mantenimiento y evolución.

AOSD a través de la separación *Crosscutting Concerns* o *concerns* transversales o cruzados desde las etapas tempranas del ciclo de vida del software hasta la implementación, elimina los fenómenos de *tangling* y *scattering* permitiendo la trazabilidad del sistema, el análisis del impacto de los cambios y la evolución del mismo, al tiempo que disminuye la complejidad e incrementa la comprensión de los diversos artefactos de requisitos, análisis, arquitectura diseño y código.

3.5.3. Descomposición aspectual. Para modularizar los *crosscutting concern*, los desarrolladores de software necesitan conocer y aplicar diferentes técnicas de descomposición. Muchos lenguajes de programación existentes, incluidos los orientados a objetos, los procedurales y los lenguajes funcionales, tienen en común que sus mecanismos claves de abstracción y composición, se basan en una forma de procedimiento.

La orientación a aspectos propone un nuevo tipo de modularización, que va más allá que los procedimientos: El aspecto. Un aspecto es un módulo que encapsula la implementación de un *crosscutting concern*. La clave de esta técnica de modularización radica en los mecanismos de composición modular. Contrario a los procedimientos que invocan explícitamente el comportamiento implementado por otros procedimientos, los aspectos tiene un mecanismo de invocación implícito. El comportamiento de un aspecto es invocado implícitamente en la implementación de otros módulos. Consecuentemente, los desarrolladores de estos otros módulos pueden ser inconscientes del *crosscutting concern*.

3.5.4. La base del paradigma: clases y aspectos. Un sistema orientado a aspectos puede verse como una combinación de la funcionalidad básica y el comportamiento aspectual, los cuales pueden ser combinados por medio de puntos de enlace.

- **Funcionalidad básica.** El paradigma orientado a aspectos utiliza toda la potencia de la orientación a objetos para soportar la funcionalidad base del sistema. Por medio de objetos se implementa la funcionalidad principal del sistema, tal como puede ser la gestión de inventarios, el pago de una nomina, entre otros. La funcionalidad básica está determinada por el dominio del problema.

- **Comportamiento aspectual.** El enfoque original con el cual surgió la orientación a aspectos identificaba el comportamiento aspectual sólo con conceptos técnicos tales como la persistencia, la gestión de errores, la sincronización o la comunicación de procesos. Hoy, el comportamiento aspectual es llevado más allá de las características técnicas del sistema y empiezan a identificarse conceptos del dominio del problema que tiene comportamiento aspectual y que cruzan el sistema.

3.5.5. Pointcut y advice. El mecanismo de invocación implícito requiere que el aspecto especifique por si mismo donde o cuando necesita ser invocado. La implementación de un aspecto consiste en dos partes conceptualmente diferentes: el código de la funcionalidad aspectual y el código de la aplicación aspectual. El código de la funcionalidad aspectual no es esencialmente diferente del código regular y es ejecutado donde el aspecto es invocado. Esta invocación del aspecto es determinada por el código de aplicabilidad del aspecto. Este código contiene las declaraciones donde el aspecto necesita ser invocado. En la terminología de aspectos, el código de aplicabilidad del aspecto es referido como un *pointcut* y el código de la funcionalidad des referido como el *advice* del aspecto. Desde un simple aspecto puede consistir de múltiples y diferentes funcionalidades que necesitan ser invocados desde diferentes sitios en el código. La implementación de un aspecto puede consistir de varios segmentos de código de *pointcuts* y *advices*.

3.5.6. Tejedor (weaver). La interacción entre clases y aspectos se hace posible a través de un tercer componente, el cual conocemos como tejedor. El tejedor es el encargado de realizar la mezcla de la funcionalidad base con el comportamiento aspectual. Las clases y los aspectos se pueden mezclar de dos formas distintas: de manera estática y de manera dinámica.

- **Tejido estático.** Implica modificar el código fuente de una clase insertando sentencias en estos puntos de enlace. Es decir, que el código del aspecto se introduce en el de la clase. Dos características importantes del tejido estático, son: se evita un impacto negativo en el rendimiento de las aplicaciones, pero se hace difícil identificar los aspectos en el código una vez ya se ha tejido.

- **Tejido dinámico**. El entrelazado dinámico requiere que los aspectos existan y estén presentes de forma explícita tanto en tiempo de compilación como en tiempo de ejecución. A partir de una interfaz de reflexión, el tejedor es capaz de añadir, adaptar y borrar aspectos de forma dinámica, si así se desea, durante la ejecución.

3.5.7. Aplicaciones bajo el paradigma orientado a aspectos. Para especificar un sistema orientado a aspectos tenemos tres elementos principales:

- Clases, las cuales modelan la funcionalidad base

- Aspectos

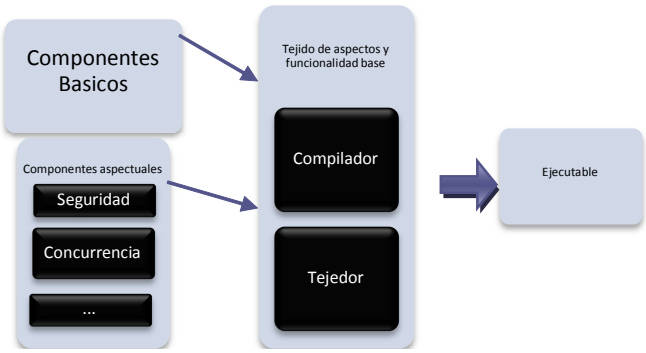
- Tejedor

En el esquema tradicional, la construcción de la aplicación se realiza a través de un compilador o intérprete, el cual simplemente toma el código de la funcionalidad base y lo traduce a un lenguaje entendible por la maquina. La construcción de aplicaciones bajo este esquema se muestra en la Figura 5.

Figura 5: Construcción de aplicaciones enfoque tradicional

La construcción de una aplicación bajo el enfoque orientado a aspectos requiere la combinación del código de la funcionalidad base con los distintos módulos que implementan los aspectos. Esta combinación es realizada por el tejedor, cada aspecto codificado con un lenguaje distinto. La construcción de aplicaciones bajo este esquema se muestra en la Figura 6.

Figura 6: Construcción de aplicaciones orientado por aspectos



El paradigma aspectual nos permite ver el sistema como un conjunto conformado por un modelo de objetos y varios modelos de aspectos. El modelo de objetos es el encargado de implementar la funcionalidad base del sistema, mientras que el modelo de aspectos nos permite implementar las características no funcionales (distribución, manejo de errores, sincronización entre otras) y según los enfoques más recientes, también nos permite implementar aspectos del dominio del problema.

3.6. LENGUAJE UNIFICADO DE MODELADO UML

Un lenguaje proporciona un vocabulario y reglas para combinar palabras de ese vocabulario con el objetivo de posibilitar la comunicación. Un lenguaje de modelado es un lenguaje cuyo vocabulario y reglas se centran en la representación conceptual, lógica y física de un sistema. El lenguaje unificado de modelado UML es un lenguaje estándar para escribir planos de software. UML puede utilizarse para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

El modelado proporciona una comprensión de un sistema. Nunca es suficiente con un único modelo. Para sistemas con gran cantidad de software se requiere un lenguaje que cubra las diferentes vistas de arquitectura de un sistema mientras evoluciona a través de un ciclo de vida de software.

De acuerdo al concepto dado por Booch, Rumbaugh y Jacobson:

> UML es solo un lenguaje y por lo tanto es tan solo una parte de un método de desarrollo de software. UML es

independiente del proceso, aunque su origen está estrechamente relacionado con RUP, el cual es un marco metodológico dirigido por los casos de uso, centrado es arquitectura, interactivo e incremental. Un proceso bien definido guiará a los usuarios al decidir los artefactos a producir, que actividades y que personal que se emplea para crearlos y gestionarlos, y como usar estos artefactos para medir y controlar el proyecto de forma global. Sin embargo UML es algo más que un conjunto de símbolos gráficos. Detrás de cada símbolo hay una notación UML bien definida. De esa manera, un desarrollador puede escribir un modelo, y otro desarrollador o incluso una herramienta puede interpretar el modelo sin ambigüedad e implementar la solución[36].

3.6.1. UML 2.0. Las características de las especificaciones UML 1.x, llevaban a problemas como excesivo tamaño, alta complejidad de uso debido a la amplia cantidad de conceptos, semántica imprecisa no apta para la generación de código ejecutable o modelos, soporte inadecuado para desarrollos basados en componentes, falta de soporte para intercambio de diagramas. Como respuesta a estos problemas OMG realiza la versión UML 2.0 que tiene como objetivos principales hacer el lenguaje más flexible, y permitir la validación y ejecución de modelos. La Versión UML 2.0 se compone de 4 estándares, como se observa en la siguiente figura:

Figura 7: Paquete completo UML 2.0

Object Management Group, Unified Modeling Language: Superstructure, Versión 2.1.1, 2007

- **UML 2.0 superestructura.** Contiene la definición formal de los constructores de UML. En este documento se especifican nuevas características y diagramas con el fin de mejorar los modelos para responder a las necesidades de mejorar el soporte para desarrollos basados en componentes, mejorar el modelado de

[36] BOOCH, G. RUMBAUGH, J. JACOBSON, I. El Lenguaje Unificado de modelado, Pearson Rentice Hall, 2006.

procesos de negocios y aumentar el soporte para arquitecturas de tiempo de ejecución (comparar modelos ejecutables) incluyendo la especificación de estructuras jerárquicas y comportamientos dinámicos.

- **UML 2.0 infraestructura.** Contiene los constructores básicos para la definición y adaptación de UML, y proporciona los mecanismos de extensión de UML. Surge como respuesta a las propuestas para mejorar las bases arquitectónicas de UML, incluyendo su núcleo y sus mecanismos de extensión.

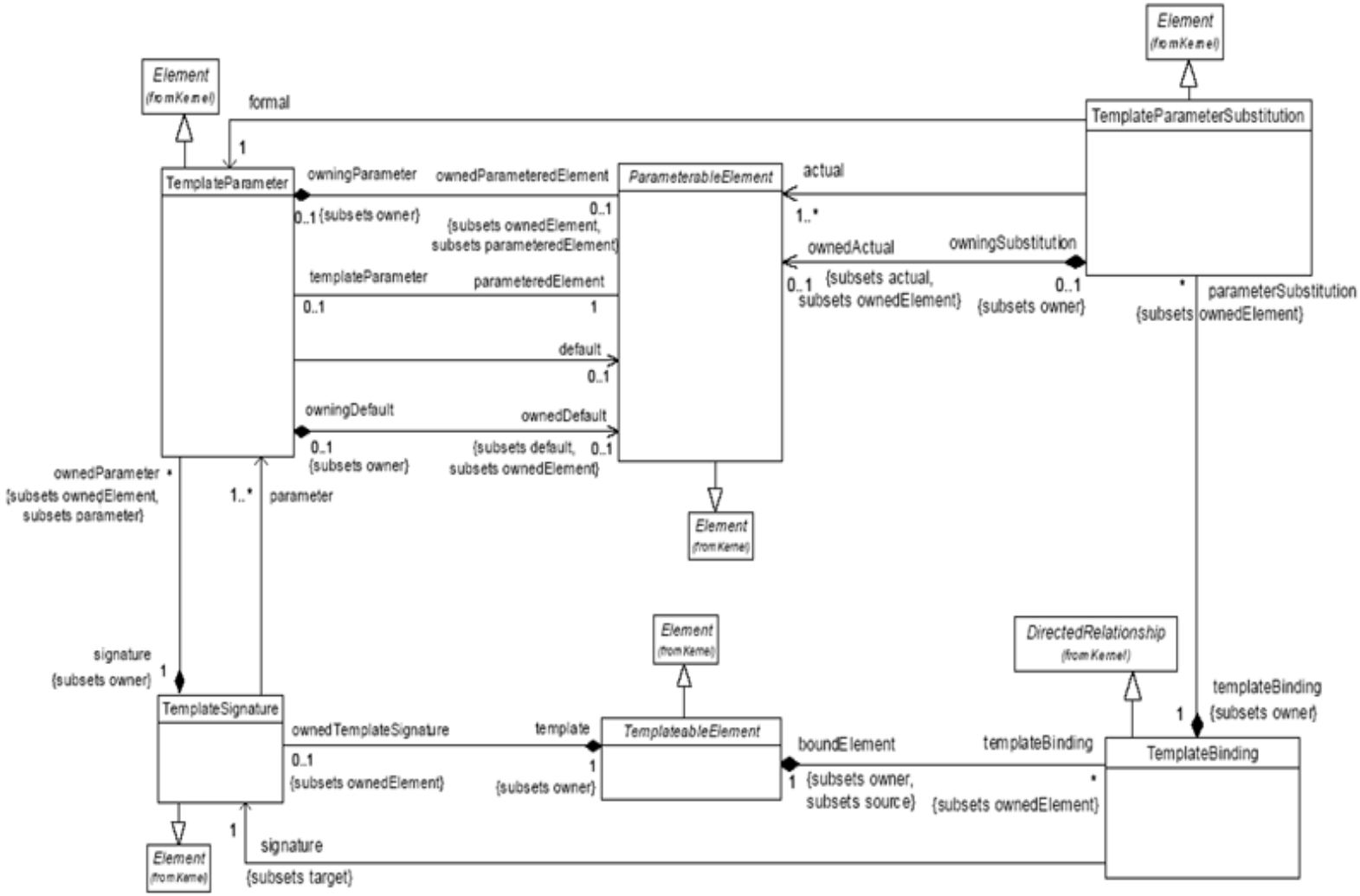
- **Estándar para intercambio de diagramas.** Permite compartir diagramas entre diferentes herramientas de modelado. Surge como respuesta a las propuestas que definieran un metamodelo para el intercambio de elementos de diagramas entre herramientas UML. Este metamodelo soporta el intercambio de características tales como la posición de los elementos, el agrupamiento de elementos, la alineación de elementos, las configuraciones de las fuentes, los caracteres y los colores.

- **Lenguaje de restricción de objetos (OCL).** Contiene la definición de un metamodelo de Lenguaje de Restricciones de Objetos (OCL) acorde al metamodelo de UML, para definir en un diagrama invariantes, precondiciones, poscondiciones y restricciones.

3.6.2. Templates de la especificación UML 2.0. Un Template es un constructor descrito en la especificación de Superestructura de UML 2.0, como un elemento de modelado el cual es parametrizado por otros elementos del modelo y pueden ser de tipo clasificador (*Classifiers*), paquete (*Packages*), y operación (*Operations*). Para especificar su parametrización un elemento Template posee una firma o *Signature*. Un Template también describe e identifica un patrón para un grupo particular de elementos. Los Templates se pueden usar para diseñar un único elemento de modelado que puede funcionar con diferentes tipos de datos. Un Template puede ser utilizado para generar otro modelo de elementos utilizando relaciones *TemplateBinding*. En esta relación los parámetros del *Signature* del Template que se especifican como parámetros formales serán sustituidos por parámetros actuales o los parámetros predeterminados en el enlace.

Dentro de la especificación de Superestructura de UML los constructores Templates usan 6 clases, estas determinan sus características. A continuación se hará una descripción de estas clases y se presentan ejemplos gráficos.

Figura 8: Clases que brindan las características de los constructores Template



Object Management Group, Unified Modeling Language: Superstructure, Versión 2.1.1, 2007

- **ParameterableElement.** Es un elemento de tipo clasificador, valor especifico, Propiedad u Operación que puede ser expuesto como un parámetro formal de un *Template*, o especificado como parámetro actual en un *TemplateBinding* dependiendo de la clase que lo referencie. Esta clase de define como una metaclase abstracta.

  Un *ParameterableElement* puede ser referenciado por la clase *TemplateParameter* cuando éste va a ser definido como un parámetro formal de un *Template*, en este caso este *ParameterableElement* es usado como restricción para los argumentos actuales que se especificarán en la relación *TemplateBinding* de modo que los parámetros actuales que esta relación contenga deberán respetar el tipo de elemento que representa el *ParameterableElement*.

  El *ParameterableElement* expuesto como un *TemplateParameter* puede ser usado en el *Template* como cualquier otro elemento del tipo definido en el espacio de nombres del *Template* y el *ParameterableElement* no podrá ser utilizado en otras partes del modelo.

  Un *ParameterableElement* también puede ser referenciado por la clase *TemplateParameterSubstitution* donde será usado como un parámetro actual dentro de una relación *TemplateBinding*.

- **TemplateableElement.** Se describe como un elemento que puede ser definido opcionalmente como una plantilla (*Template*) o en un elemento vinculado (*Bound Element*). Un *TemplateableElement* puede contener un *Templatesignature*, el cual especifica los parámetros formales de un *Template*. Un *TemplateableElement* que contiene un *Templatesignature* después es referenciado como plantilla o *Template*.

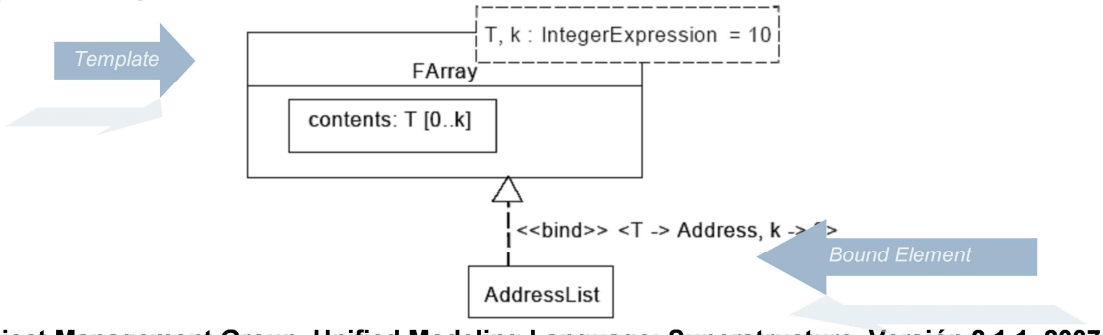
  Un *TemplateableElement* puede contener un *TemplateBinding*, el cual describe como los parámetros formales son remplazados por los parámetros actuales. Un *TemplateableElement* que contenga un *TemplateBinding* es después referenciado como un elemento vinculado o *Bound Element*.

  Si un *TemplateableElement* tiene parámetros de *Template* (*TemplateParameter*), un pequeño rectángulo de línea punteada se superpone en el símbolo del *TemplateableElement*, normalmente en la esquina superior derecha. El rectángulo punteado contiene una lista de parámetros de *Template* denominados formales. La lista de parámetros no puede estar vacía, aunque podría ser suprimida en la presentación. La lista de parámetros formales del *Template* puede ser mostrada como una lista separada por comas, o se puede presentar un parámetro formal de *Template* por línea. Un elemento vinculado tiene la misma notación gráfica que otros elementos del tipo al que corresponda por ejemplo una clase, un paquete etc. (ver Figura 9)

Un elemento vinculado puede tener múltiples *TemplateBinding*, posiblemente al mismo *Template*. Adicionalmente, un elemento vinculado puede contener elementos diferentes a los del *Binding*, cada relación *TemplateBinding* es mostrada usando la notación que se describe en la especificación para ella.

Un *TemplateableElement* puede contener al mismo tiempo un *TemplateBinding* como un *Templatesignature*. Así un *TemplateableElement* puede ser al mismo tiempo un *Template* y un elemento vinculado.

Figura 9: TemplateableElement



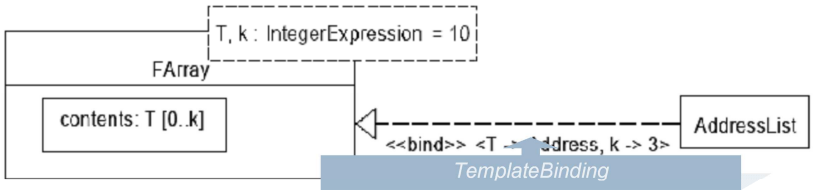
Object Management Group, Unified Modeling Language: Superstructure, Versión 2.1.1, 2007

- **TemplateBinding.** Representa una relación entre un elemento vinculado y un *Template*. Un *TempleteBinding* especifica la sustitución de los parámetros actuales por los parámetros formales de un *Template*. A un *TemplateBinding* le pertenece un conjunto de *TemplateParameterSubstitutions*.

  La semántica de una relación *TemplateBinding* es equivalente al modelo de elementos que se derivan de copiar el contenido del *Template* dentro de un elemento vinculado, remplazando todos los elementos expuestos como *TemplatesParameter* o parámetros formales con los elementos correspondientes especificados como parámetros actuales en el *TemplateBinding*. Si no se especifica el parámetro actual en el *TemplateBinding* para un parámetro formal, entonces el valor por defecto definido en el parámetro formal del *Template* es usado automáticamente. (Ver Figura 10)

  Un *TemplateBinding* es mostrado como una flecha punteada con la punta en el *Template* y la cola en el elemento vinculado y el estereotipo <<bind>>. La información de la relación se muestra como una lista separada por comas.

Figura 10: TemplateBinding



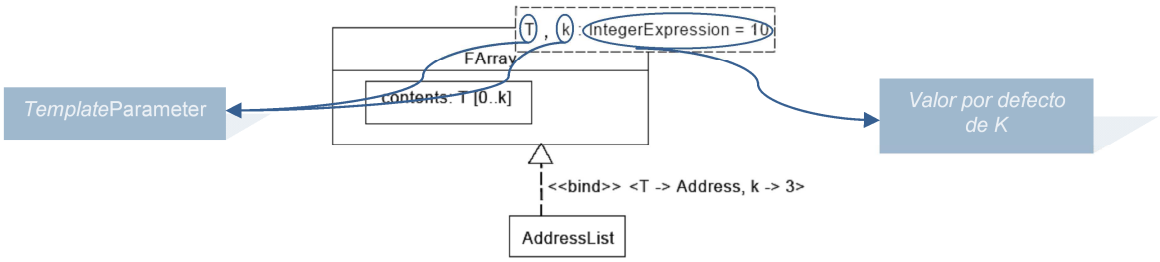
Object Management Group, Unified Modeling Language: Superstructure, Versión 2.1.1, 2007

- **TemplateParameter.** Un *TemplateParameter* expone un *ParameterableElement* como un parámetro formal de *Templates*. Este *ParameterableElement* sólo tiene sentido dentro del *Template* o de otras *Templates* que pueden tener acceso a su funcionamiento interno (por ejemplo, si el *Template* soporta especialización). Los *ParameterableElement* expuestos como *TemplateParameter* no pueden ser utilizados en otras partes del modelo. (Ver Figura 11)

  Cada elemento expuesto restringe a los elementos que puedan ser sustituidos por los parámetros actuales en una relación de *Binding*.

  A un *TemplateParameter* se le asignar un valor por defecto que sustituirá el parámetro formal en una relación *Binding*, en caso que esta relación no prevea de manera explícita la sustitución el parámetro formal.

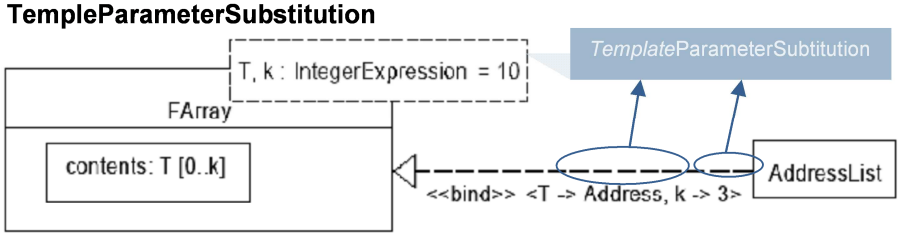
Figura 11: TemplateParameter



Object Management Group, Unified Modeling Language: Superstructure, Versión 2.1.1, 2007

- **TemplateParameterSubstitution.** Un *Template*ParameterSubstitution asocia uno o más parámetros actuales con un *TemplateParameter* o parámetro formal de un *Template* en el contexto de un *TemplateBinding*.

Figura 12: TempleParameterSubstitution

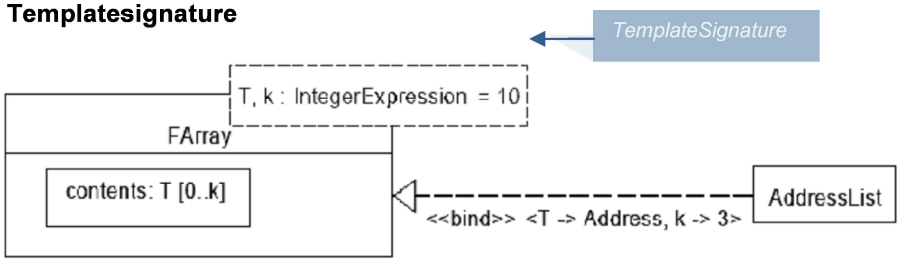


Object Management Group, Unified Modeling Language: Superstructure, Versión 2.1.1, 2007

- **Templatesignature.** El *Templates*ignature es una propiedad de un *TemplateableElement* y la cual contiene uno o más *Template*Parameters que definen el *Signature* para la relación entre *Templates* y elementos vinculados.

  Un *Templates*iganature especifica un conjunto *TemplateParameters* o parámetros formales que están asociados a un *TempleteableElement*. Los parámetros formales especifican los elementos que pueden ser sustituidos en un *TemplateBinding*.

Figura 13: Templatesignature



Object Management Group, Unified Modeling Language: Superstructure, Versión 2.1.1, 2007

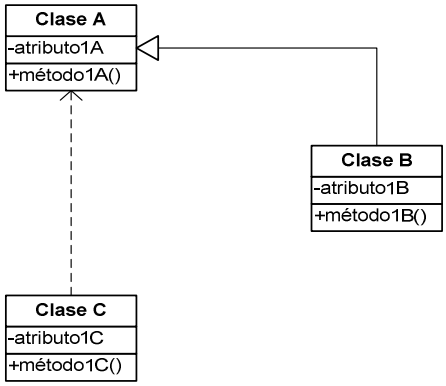
Teniendo en cuenta las clases anteriores, se puede decir que: un *Template* es un *TempleteableElement*, el cual define una lista de uno o más *TemplateParameter* que se especifican para producir un elemento de modelado para un sistema puntual. Este conjunto de *TemplateParameter* que se conocen como parámetros formales, son presentados dentro de un *Templatesignature*, el cual se representa con un rectángulo puenteado al borde superior izquierdo del *TempleteableElement*.

Un elemento vinculado también es un *TemplateableElement*. Al relacionar un elemento vinculado con un Template, se genera una relación de tipo

TemplateBinding. Esta relación contiene los valores específicos que remplazaran a cada uno de los parámetros formales que se encuentran en el *Templatesignature* del Template. Estos valores recibe el nombre de *TemplateParameterSubstitution*.

3.6.3. Diagramas de clases. Los diagramas de clases son los más utilizados en el modelado de sistemas orientados a objetos. Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Gráficamente, un diagrama de clases es una colección de nodos y arcos, como se muestra en la figura 14.

Figura 14: Diagrama de clases



Booch, G. Rumbaugh, J. Jacobson, I. El Lenguaje Unificado de modelado, Pearson Rentice Hall, 2006

Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema. Principalmente, esto incluye modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas. Los diagramas de clases también son la base para un par de diagramas relacionados: los diagramas de componentes y los diagramas de despliegue.

Los diagramas de clases son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa o inversa.

Los diagramas de clases contienen normalmente los siguientes elementos:

- Clases
- Interfaces
- Relaciones de dependencia, generalización y asociación

Al igual que los demás diagramas, los diagramas de clases pueden contener notas y restricciones.

"Los diagramas de clases también pueden contener paquetes o subsistemas, los cuales se usan para agrupar los elementos de un modelo en partes más grandes. A veces se colocarán instancias en los diagramas de clases, especialmente cuando se quiera mostrar el tipo (posiblemente dinámico) de una instancia"[37].

- **Clases.** Las clases son los bloques de construcción más importantes de cualquier sistema orientado a objetos. Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Una clase implementa una o más interfaces. Gráficamente, se representa como un rectángulo.

  Las clases se pueden utilizar para capturar el vocabulario del sistema que se está desarrollando. Estas clases pueden incluir abstracciones que formen parte del dominio del problema, así como las clases que constituyan una implementación. Se pueden utilizar las clases para representar cosas que sean software, software o puramente conceptuales.

  "Las clases bien estructuradas están bien delimitadas y forman parte de una distribución equilibrada de responsabilidades en el sistema[38]".

  Las partes más importantes de una clase son:

  - ✓ **Nombre**: cada clase ha de tener un nombre que la distinga de otras clases. Un nombre es una cadena de texto. Ese nombre sólo se denomina nombre simple; un nombre de camino consta del nombre de la clase precedido por el nombre del paquete en el que se encuentra. El nombre de una clase debe ser único en su paquete.

  - ✓ **Atributos**: un atributo es una propiedad de una clase identificada con un nombre, que describe un rango de valores que pueden tomar las instancias de la propiedad. Una clase puede tener cualquier número de atributos o no tener ninguno. Un atributo representa alguna propiedad del elemento que está modelando que es compartida por todos los objetos de esa clase.

  - ✓ **Operaciones**: una operación es la implementación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre un comportamiento. En otras palabras, una operación es una abstracción de

[37] BOOCH. Op. cit.

[38] Ibid.

algo que se puede hacer a un objeto y que es compartido por todos los objetos de la clase. "Una clase puede tener cualquier número de operaciones o ninguna[39]".

- **Interfaces.** Las interfaces definen una línea entre la especificación de lo que una abstracción hace y la implementación de cómo lo hace. Una interfaz es una colección de operaciones que sirven para especificar un servicio de una clase o de un componente. Gráficamente, una interfaz se representa como un círculo; de forma expandida, una interfaz se puede ver como una clase estereotipada para mostrar sus operaciones y otras propiedades.

  "Las interfaces se utilizan para visualizar, especificar, construir y documentar las líneas de separación dentro de un sistema. Una interfaz bien estructurada proporciona una clara separación entre las vistas externa e interna de una abstracción, haciendo posible comprender y abordar una abstracción sin tener que sumergirse en los detalles de su implementación[40]".

- **Relaciones.** Al realizar abstracciones, uno se da cuenta de que muy pocas clases se encuentran aisladas. En vez de ello, la mayoría colaboran con otras de varias maneras. Por lo tanto, al modelar un sistema, no sólo hay que identificar los elementos que conforman el vocabulario del sistema, también hay que modelar cómo se relacionan estos elementos entre sí.

  Una relación es una conexión entre elementos. Gráficamente, una relación se representa como una línea, usándose diferentes tipos de línea para diferencia los tipos de relaciones.

  En el modelado orientado a objetos hay tres tipos de relaciones especialmente importantes: dependencias, que representan relaciones de uso entre las clases (incluyendo refinamiento, traza y ligadura); generalizaciones, que conectan clases generales con especializaciones, y asociaciones, que representan relaciones estructurales entre los objetos. Cada una de estas relaciones proporciona una forma diferente de combinar las abstracciones.

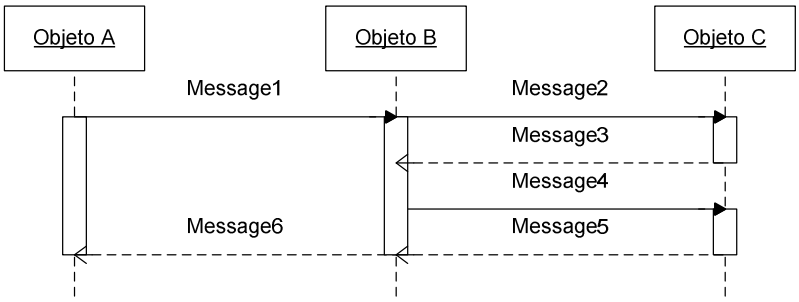
3.6.4. Diagramas de secuencia. Un diagrama de secuencia destaca la ordenación temporal de los mensajes. Como se muestra en la figura 15, se forma colocando en primer lugar los objetos que participan en la interacción en la parte superior del diagrama, a lo largo del eje X. Normalmente, se coloca a la izquierda el objeto que inicia la interacción, y los objetos subordinados a la derecha. A continuación, se colocan los mensajes que estos objetos envían y reciben a lo

[39] Ibid.

[40] Ibid.

largo del eje Y, en orden de sucesión en el tiempo, desde arriba hasta abajo. Esto ofrece al lector una señal visual clara del flujo de control a lo largo del tiempo.

Figura 15: Diagrama de secuencia



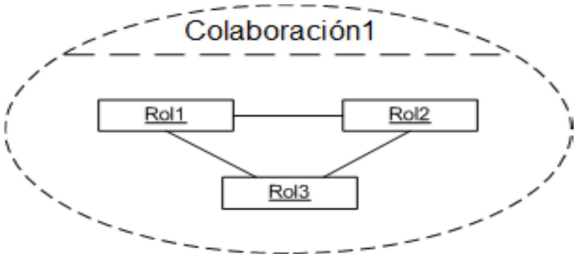
Booch, G. Rumbaugh, J. Jacobson, I. El Lenguaje Unificado de modelado, Pearson Rentice Hall, 2006

Los diagramas de secuencia tienen dos características que los distinguen. En primer lugar, está la línea de vida. La línea de vida de un objeto es la línea discontinua vertical que representa la existencia de un objeto a lo largo de un periodo de tiempo.

En segundo lugar, está el foco de control. El foco de control es un rectángulo delgado y estrecho que representa el periodo de tiempo durante el cual un objeto ejecuta una acción, bien sea directamente o a través de un procedimiento subordinado. La parte superior del rectángulo se alinea con el comienzo de la acción; la inferior se alinea con su terminación.

3.6.5. Colaboraciones. Una colaboración representa una interacción entre objetos cuyo propósito es la realización de alguna actividad, y en donde cada objeto participante juega un determinado rol. Gráficamente, una colaboración se representa como una elipse con el borde discontinuo, como lo muestra la figura 16.

Figura 16: Colaboración



Ortín, Maria José; Garcia Molina, José. "Modelado basado en roles con UML".Departamento de Informática, Lenguajes y Sistemas. Facultad de Informática - Universidad de Murcia

Una colaboración no es propietaria de ninguno de sus elementos estructurales, sino que los referencia o usa. Por esto un mismo elemento puede formar parte de más de una colaboración.

Una colaboración tiene dos aspectos: el aspecto estructural o estático consiste en un conjunto de roles y las relaciones estructurales entre ellos; el aspecto dinámico o de comportamiento consiste en una o más interacciones entre los roles participantes en la colaboración.

Se debe tener en cuenta el concepto de rol dado por María José Ortín y José García Molina quienes dicen:

> Un rol describe las propiedades estructurales y de comportamiento de un objeto en un contexto. Un objeto juega uno o más roles a lo largo de su existencia y un mismo rol puede ser jugado por diferentes objetos. Dado que un objeto finalmente será instancia de una clase, un rol es una proyección (vista parcial) de dicha clase, de modo que las propiedades del rol serán un subconjunto de las propiedades de la clase del objeto. Una clase implementará uno o más roles y un rol podrá ser implementado por una o más clases[41].

3.7. THEME/UML

[41] ORTÍN, Maria José; GARCIA MOLINA, José. "Modelado basado en roles con UML".Departamento de Informática, Lenguajes y Sistemas. Facultad de Informática - Universidad de Murcia.

Theme/UML es una aproximación de diseño orientada por aspectos. Es una extensión de UML que enfatiza en apoyar al diseñador en la especificación de Themes que se traslapan unos con otros, y en las capacidades de composición, que están definidas.

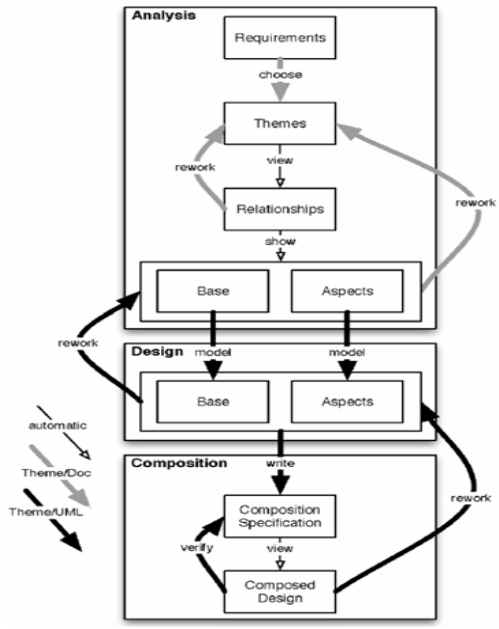
En todas las aproximaciones orientadas a aspectos, debe haber una manera para definir como relacionar los aspectos con el resto del sistema, para proveer esta capacidad, Theme/UML tiene definida una nueva clase de relaciones, llamada una relación de composición, que permite al diseñador identificar estas partes en el diseño del Theme y por lo tanto como debería ser compuesto. Para los Themes que entrecruzan otros, estos medios identifican cuando y donde en estos otros Themes, debe ocurrir el comportamiento adicional, para otras clases de traslapo, estos medios identifican elementos de diseño en el Theme que correspondan el uno con el otro y dicen cómo deben ser integrados.

Theme/UML también provee semánticas para composición de modelos basados en la especificación de relaciones de composición, este soporta un nivel de verificación que permite que el diseñador considere el diseño del sistema total, incluyendo la especificación de composición, para asegurar que tenga sentido Theme/UML es parte de una aproximación para análisis orientado a aspectos y diseño llamado Theme. Theme tiene dos partes: Theme/Doc es un conjunto de heurísticas y herramientas para la visualización y documentación de análisis de requerimientos de software con el propósito de encontrar el Theme a ser diseñado, Theme/UML es la segunda parte, la cual es una aproximación de diseño orientado por aspectos basados en UML.

3.7.1. Proceso THEME/UML. El proceso del Theme se describe por Chitchyan et al[42], como un proceso de tres fases, análisis, diseño y composición. En la fase de análisis, se identifican y se caracterizan los Themes. En la fase de diseño los Themes identificados y caracterizados se especifican en modelos técnicos del diseño. Finalmente, en la fase de la composición, se especifica la composición de Themes. El proceso completo se ilustra en la Figura 17.

Figura 17: Proceso completo Theme

[42] CHITCHYAN, R. RASHID, A. SAWYER, P. GARCIA, A. PINTO, M. BAKKER,J. TEKINERDOGAN, B. CLARKE, S. JACKSON, A. Survey of Aspect-Oriented Analysis and Design Approaches. AOSD-EUROPE. 2005



Chitchyan, R. Rashid,A. Sawyer, P. Garcia, A. Pinto, M. Bakker,J. Tekinerdogan, B. Clarke, S. Jackson, A. Survey of Aspect-Oriented Analysis and Design Approaches. AOSD-EUROPE. 2005

3.8. MODELO DE REPRESENTACIÓN DE PATRONES ASPECTUALES DE ANÁLISIS – MORENA

MORENA es el modelo de representación de patrones aspectuales de análisis propuesto por ESKEMA, por lo tanto se hace pertinente una breve descripción de ESKEMA.

3.8.1. ESKEMA. Barón[43], afirma que ESKEMA tiene como objetivo proveer mecanismos que permitan introducir de manera natural en las tareas del desarrollador, la reutilización como práctica sistémica. ESKEMA facilita a las empresas la incorporación de la reutilización en su proceso de desarrollo de software, permitiendo estandarizar las tareas de reutilización (identificación, definición, construcción, almacenamiento, etiquetado, documentación, búsqueda y gestión de activos de software reutilizables).

Para el cumplimiento de este objetivo, ESKEMA provee el catálogo de patrones de análisis representados mediante MORENA, con una estructura que facilita los mecanismos de búsqueda, recuperación e instanciación, además de un componente gestor de experiencias de uso de los patrones de análisis.

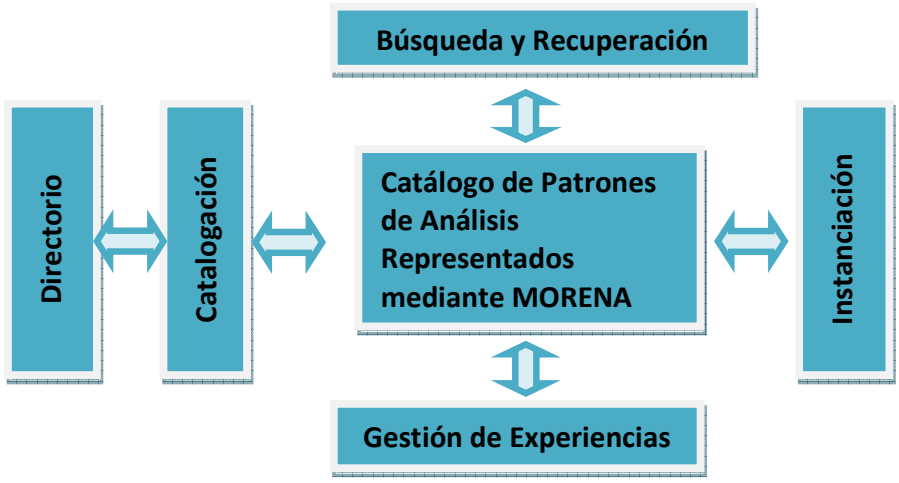
[43] BARÓN. Op. cit.

- **Modelo de ESKEMA.** En el trabajo Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual se afirma que:

> El modelo de ESKEMA se basa en la reutilización de activos de software integrados en un repositorio. En ESKEMA, los activos reutilizables son los patrones de análisis y el repositorio recibe el nombre de catálogo. Durante la ingeniería de dominio, se crean los patrones de análisis como soluciones a problemas recurrentes en diferentes contextos de un mismo dominio o de varios dominios. Durante la ingeniería de aplicación se instancian los patrones a las soluciones especificas. ESKEMA integra los mecanismos de gestión del catálogo: catalogación, búsqueda, recuperación e instanciación[44].

En la figura 18 se presentan los componentes de ESKEMA.

Figura 18: Componentes de ESKEMA



Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

- ✓ **Catálogo de patrones de análisis representados mediante MORENA.** El catálogo es el componente que permite la gestión de persistencia de los patrones de análisis. El catálogo provee una estructura que permite ejecutar

[44] Ibid.

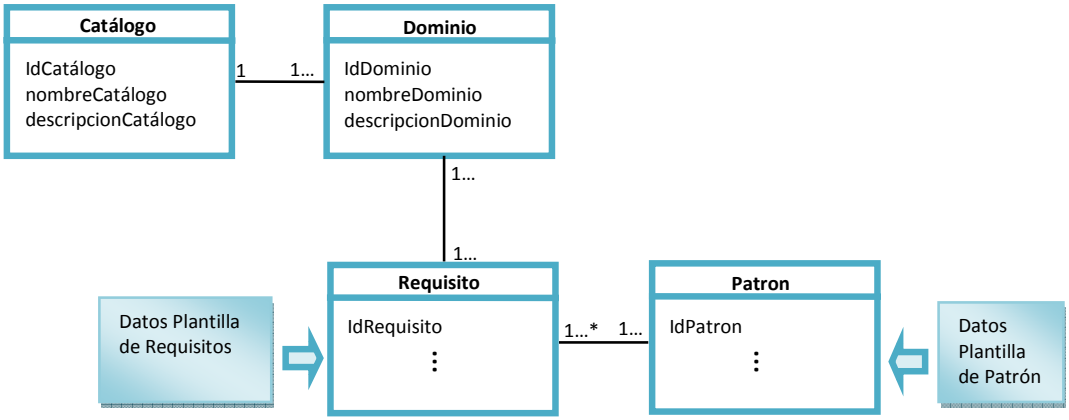
los mecanismos de gestión de los patrones de análisis (catalogación, búsqueda, recuperación, instanciación y retroalimentación).

El catálogo de patrones de análisis en ESKEMA esta divido en dominios. A cada dominio se asocian un conjunto de patrones de análisis y un conjunto de requisitos de dominio. Cada patrón de análisis y cada requisito de dominio pueden asociarse a un dominio o a varios dominios.

La asociación del patrón de análisis con los requisitos de dominio que resuelve y los dominios donde se aplica, hacen parte de la definición de los patrones de análisis en ESKEMA.

El catálogo de patrones de análisis de ESKEMA hereda los métodos y la arquitectura de la herramienta CASE a la cual se integra.

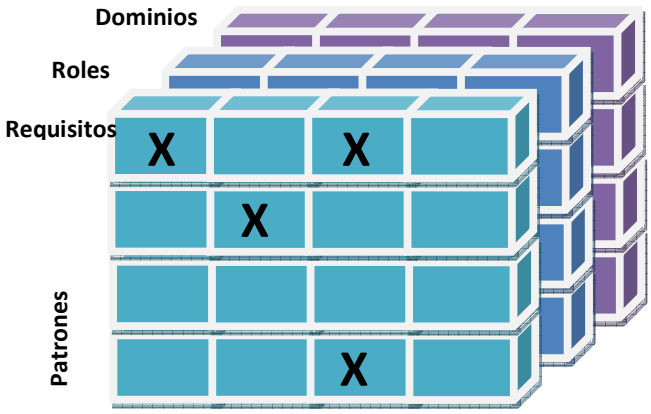
Figura 19: Estructura del catálogo de patrones de análisis



Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

- ✓ **Directorio del catálogo de patrones de análisis.** El directorio es el componente de ESKEMA que define el sistema de indexación, es decir, los criterios de organización lógica de los patrones de análisis en el catálogo. Este componente se fundamenta en la estructura del catálogo y en los elementos de definición del patrón de análisis. El directorio es consumido por los mecanismos de búsqueda y recuperación. Unificar los índices del directorio con los criterios de búsqueda y recuperación permite optimizar la gestión del catálogo de patrones de análisis.

Figura 20: Directorio del catálogo de patrones de análisis

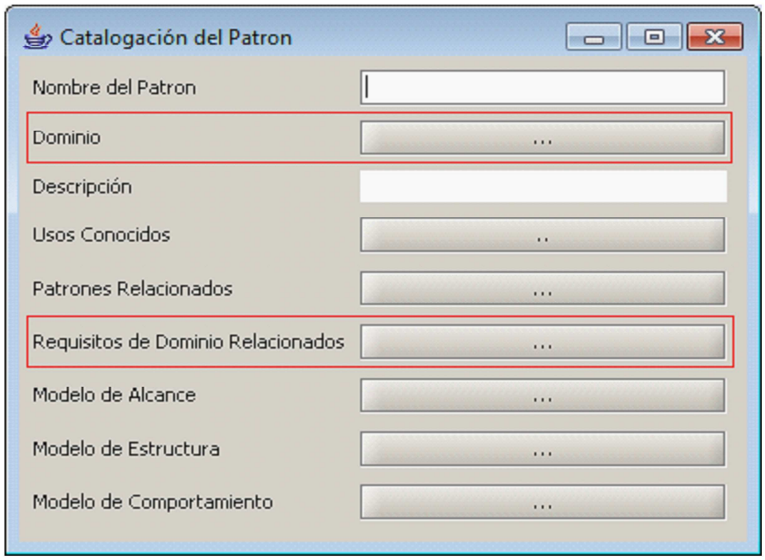


Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

- ✓ **Mecanismo de catalogación.** El mecanismo de catalogación es la estrategia que provee ESKEMA para integrar un patrón de análisis al catálogo, asignando los valores de indexación definidos en el directorio.

  La catalogación se desarrolla durante la definición del patrón indicando el requisito de dominio o los requisitos de dominio que resuelve, el dominio o los dominios en los cuales se aplica y los roles participantes. La catalogación de un patrón de análisis puede ser actualizada cuando el desarrollador indica una nueva asociación del patrón con otros requisitos de dominio o la participación de otros roles. En cualquier caso el catálogo se actualiza y ESKEMA provee la vista requerida para el uso de los patrones de análisis en las nuevas condiciones.

Figura 21: Mecanismo de catalogación



Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

- **Proceso de ESKEMA.** En ESKEMA se define un proceso de desarrollo que dispone de actividades lógicamente organizadas para gestionar un desarrollo basado en patrones de análisis.

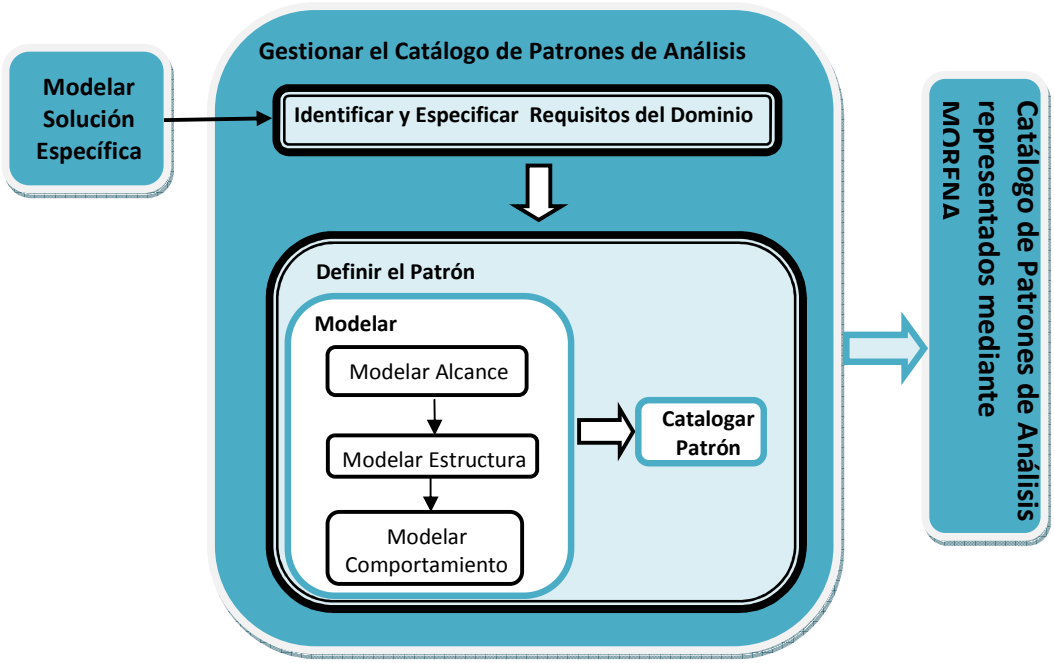
  Se distinguen dos grandes actividades: El análisis de dominio para definir los patrones que representan comportamiento genérico y el modelado de solución especifica a partir de estos patrones, como se observa en la Figura 22.

Figura 22: Vista general del proceso de desarrollo de ESKEMA

Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

- ✓ **Gestión del catálogo de patrones de análisis.** El objetivo de esta actividad es definir el proceso que guiará la construcción del catálogo a partir de la identificación y especificación de requisitos del dominio y la retroalimentación del catálogo a partir de requisitos de una solución específica generalizada. (ver figura 23).

Figura 23: Gestionar el catálogo de patrones de análisis



Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

- o **Identificar y especificar requisitos del dominio.** Esta acción busca definir los requisitos que especifican el dominio y que son insumo para la definición del patrón de análisis. Para la construcción inicial del catálogo, los requisitos se identifican a partir del análisis del dominio. Para la retroalimentación del catálogo, los requisitos se identifican a partir del análisis de una situación específica para la cual no se encontró solución en el catálogo, en este caso, el requisito debe ser generalizado para que se defina como requisito de dominio y sea insumo para la definición de un nuevo patrón.

Esta acción puede ser desarrollada aplicando aproximaciones de ingeniería de dominio como FODA, ODM, DARE o la de Martin Fowler. La ingeniería de dominios reconoce dos procesos distintos: la ingeniería de dominio y la ingeniería de aplicación.

La ingeniería de dominio se centra en el desarrollo de elementos reutilizables que formarán la familia de productos, mientras que la ingeniería de aplicación se orienta hacia la construcción de productos individuales, pertenecientes al dominio, y que satisfacen un conjunto de requisitos y restricciones expresados por un usuario específico, reutilizando, adaptando e integrando los elementos reutilizables existentes y producidos en la ingeniería de dominio. En este punto ESKEMA identifica la ingeniería de dominios como parte del proceso.

- **Definir el patrón.** Esta acción busca modelar el patrón e integrarlo al catálogo Con base en los requisitos se define el patrón de análisis. Esta acción incluye modelar y catalogar el patrón.

  Modelar el patrón es una tarea que busca generar los modelos que definen al patrón, desde tres vistas que se representan en UML: el modelado del alcance, representado por una colaboración; el modelado estructural, representado por un diagrama de clases y el modelado de comportamiento representado por un diagrama de secuencia.

  "Catalogar el patrón es la actividad que busca integrar el nuevo patrón al catálogo a partir de la instanciación de los elementos de descripción del patrón que son los criterios de organización del catálogo. La catalogación adecuada del patrón facilita su posterior búsqueda, recuperación e instanciación"[45].

  Actualmente existen varios modelos de catalogación que permiten clasificar y catalogar a los patrones, a efectos de facilitar su recuperación y posterior utilización. Así mismo, casi todas las herramientas de modelado incluyen funcionalidades para utilizar patrones en el modelado que permiten navegar por un catálogo de patrones y generar artefactos de software.

  Esta actividad es un ciclo que termina una vez se han catalogado los patrones de análisis a partir de los requisitos que han sido identificados y definidos ya sea como producto del análisis del dominio, o de la generalización de una solución especifica. Como resultado de este

[45] BARÓN. Op. cit.

trabajo, se especifican formalmente los requisitos, se modelan los patrones de análisis y se catalogan.

3.8.2. Descripción del patrón de análisis en MORENA. Existen varios formatos para la descripción de patrones de software. MORENA adopta algunos de los elementos del formato de descripción de patrones conocido como formato del GoF, descrito en Gamma et al[46]. Los elementos que describen un patrón de análisis son: nombre, dominio, descripción, solución, usos conocidos y patrones relacionados.

3.8.3. Representación del patrón de análisis en MORENA. Los modelos de alcance, estructural y de comportamiento que definen el patrón de análisis son representados por colaboración, diagrama de clases y diagrama de secuencia respectivamente. Los modelos se encapsulan en un constructor de la especificación UML 2.0 llamado Template.

- **Representación integrada.** El Object Management Group describe a continuación como Theme/UML hace uso de los Templates:

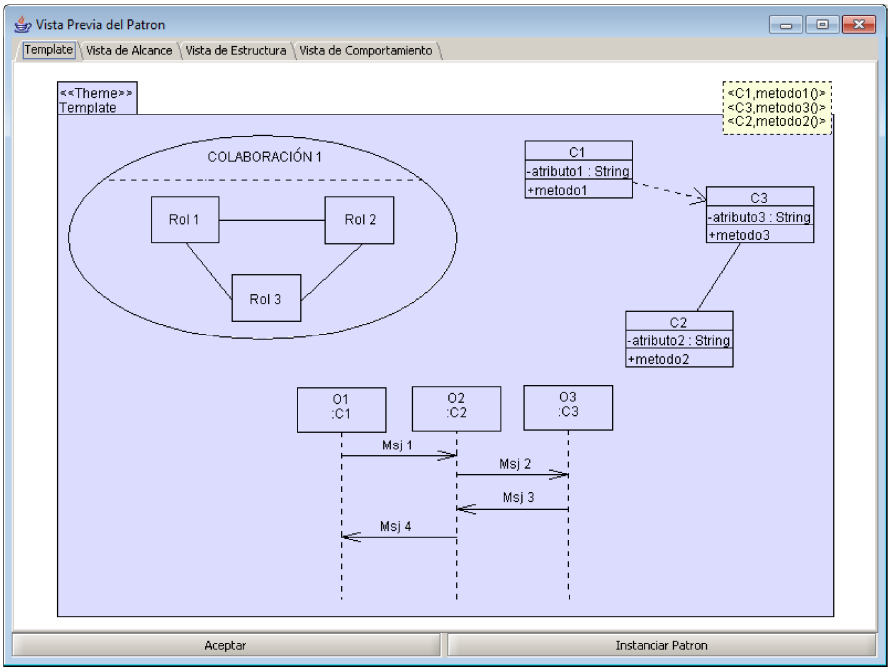
> El Template es un constructor descrito en la Superestructura de UML 2.0, como un elemento de modelado que es parametrizado por otros elementos del modelo y pueden ser de tipo clasificador, paquete u operación. Para especificar su parametrización el Template posee una firma o Signature. Un Template puede ser utilizado para generar otro modelo de elementos utilizando relaciones TemplateBinding. En esta relación los parámetros del Signature se especifican como parámetros formales que serán sustituidos por parámetros actuales.
>
> Theme/UML es una aproximación de desarrollo de software orientado por aspectos que utiliza el Template para representar el Theme Aspectual. En Theme/UML, el Template es tipo paquete, pues se utiliza como contenedor de los diagramas que representan el Theme aspectual. Los parámetros del signature reciben los valores para instanciar los modelos del theme aspectual. La relación Template

[46] GAMMA. Op. cit.

Binding es el canal por el cual los valores llegan al theme aspectual[47].

Se retoma el concepto de patrón de análisis de la aproximación Theme/UML. En MORENA, el Template es de tipo paquete pues se utiliza como contenedor de los diagramas que representan los modelos que definen el patrón de análisis. El Signature permite instanciar el patrón. Los parámetros del Signature reciben los valores del problema específico que instancian las variables del patrón de análisis en cada uno de los modelos. La relación TemplateBinding es el canal por medio del cual se envían al patrón de análisis los valores que lo instancian y que son recibidos en los parámetros de la Signature.

Figura 24: Representación integrada del patrón de análisis



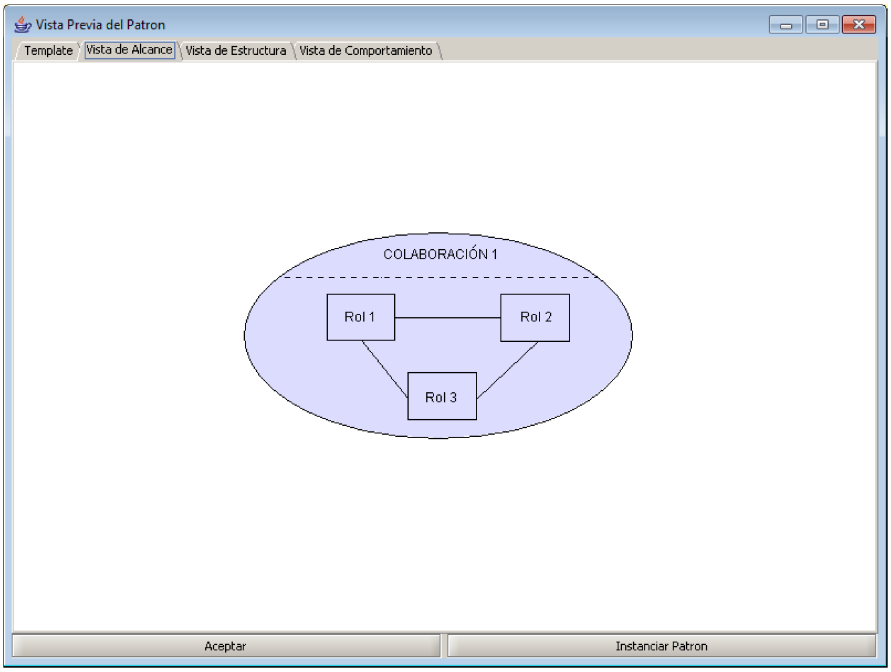
Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

- ✓ **Modelo de alcance.** El modelo de alcance permite definir como el patrón de análisis interactúa con su contexto y en especial con los usuarios de acuerdo con los roles que asumen.

[47] Object Management Group, Unified Modeling Language: Superstructure, Versión 2.1.1, 2007

En MORENA, el modelo de alcance es representado por una colaboración.

Figura 25: Representación del modelo de alcance



Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

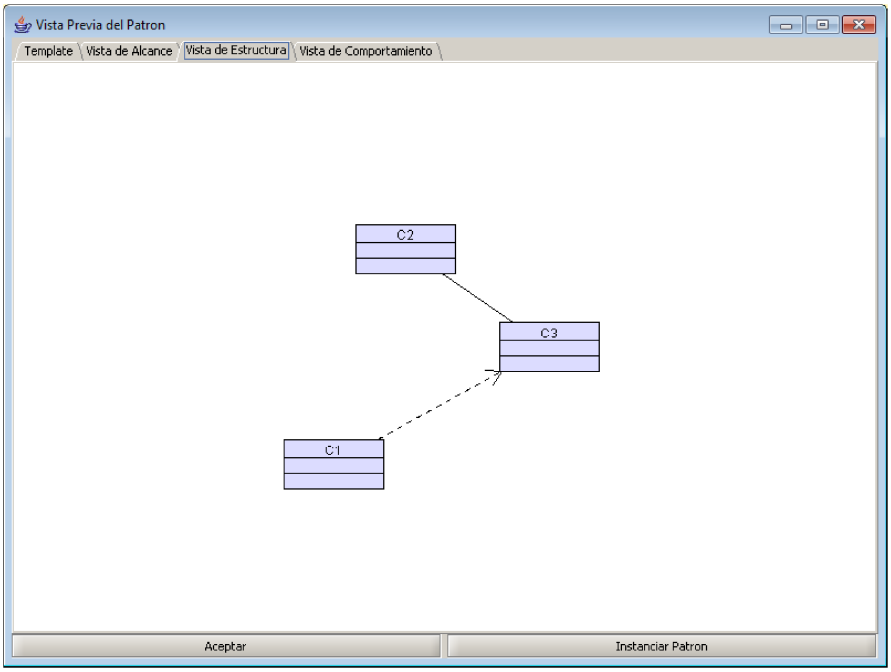
"La colaboración representa de manera adecuada como los roles asociados al patrón de análisis se relacionan entre sí. Las relaciones entre roles, permite generar el comportamiento del patrón de análisis en el contexto[48]".

- ✓ **Modelo estructural.** El modelo estructural permite representar como se compone y relaciona la estructura del patrón de análisis.

  En MORENA, el modelo estructural es representado por un diagrama de clases.

[48] BARÓN. Op. cit.

Figura 26: Representación del modelo estructural



Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

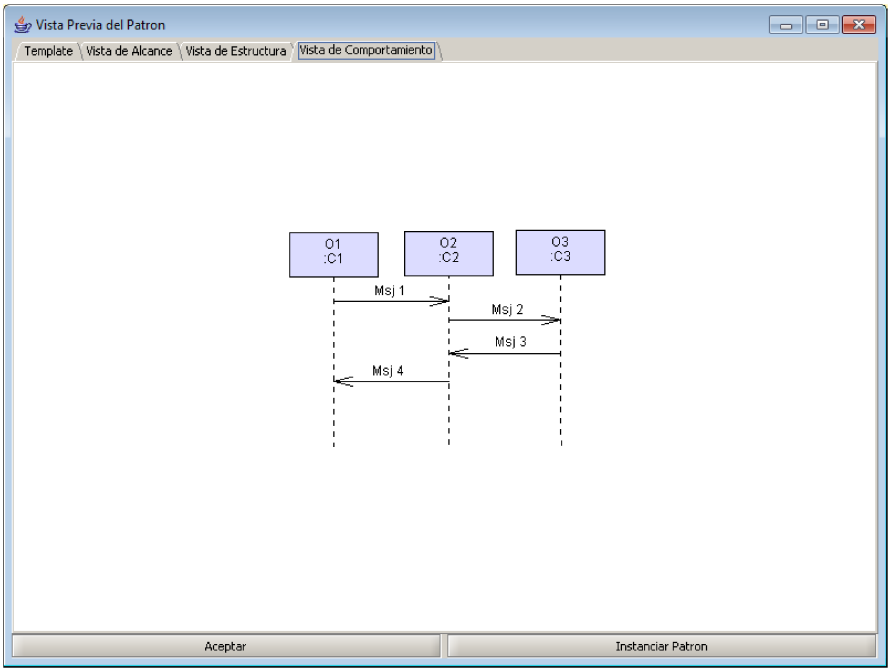
"El diagrama de clases permite representar la estructura conceptual del patrón de análisis indicando clases, atributos y relaciones. Los elementos del diagrama de clases en el patrón de análisis, reciben nombres genéricos que serán instanciados en la acción instanciar el patrón[49]".

- ✓ **Modelo de comportamiento.** El modelo de comportamiento define la parte dinámica del patrón de análisis. En MORENA, describe los aspectos del patrón de análisis que cambian con el tiempo.

  En MORENA, el modelo de comportamiento se representa por un diagrama de secuencia.

[49] Ibid.

Figura 27: Representación del modelo de comportamiento



Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

> "El diagrama de secuencia describe la interacción de los objetos del patrón de análisis a través del tiempo. Contiene detalles de implementación de la solución propuesta por el patrón, incluyendo los objetos y los mensajes pasados entre los objetos. Los elementos del diagrama de secuencia en el patrón de análisis, reciben nombres genéricos que serán instanciados durante el modelado de la solución específica[50]".

3.9. VALIDACIÓN DE MODELOS

La validación se refiere a la construcción de un modelo correcto, es el proceso de determinar si el modelo, como abstracción, es una buena representación para resolver los problemas que lo fundamentaron. De forma más sencilla, la validación trata sobre la cuestión ¿Se está construyendo el modelo correctamente? o responder a la pregunta: ¿Los datos de salida serán los esperados?

[50] Ibid.

La validación de modelos permite conocer si el comportamiento de un modelo es el esperado por el usuario y si un modelo cumple determinadas propiedades de calidad. Se debe realizar desde la primera etapa de desarrollo del modelo, esto permite detectar errores y no arrastrarlos a etapas posteriores. Aunque se critique que los métodos de validación consumen demasiado tiempo, más costoso es construir un modelo que no sea válido.

Se debe definir cuidadosamente, al principio del estudio, la cuestión de interés que en nuestro caso sería la forma en la que el modelo va a ser usado. Los modelos no son universalmente válidos, sino que son diseñados para propósitos específicos. Si la cuestión de interés no se especifica, no se puede saber si se ha conseguido el nivel de detalle deseado en el modelo.

En el curso de Computación Estadística dictado por la Universidad de Jaén[51], se afirmó que el desarrollo del modelo es un proceso iterativo en el que hay sucesivos refinamientos en cada etapa. El paso entre las distintas etapas está marcado por el éxito o fracaso al realizar la validación en las mismas, con esto se han de establecer una serie de criterios que sirvan para decidir si el modelo supera cada una de estas etapas de validación, en caso de no superarlas se realizaría la calibración respectiva.

Igualmente, los Laboratorios Docentes de la E.T.S.I[52], describen la calibración como el proceso de estimar los parámetros de un modelo, ésta es una forma de prueba y ajuste de los parámetros existentes y generalmente no incluye la introducción de otros nuevos, cambiando la estructura del modelo.

Por otra parte, para poder aplicar convenientemente cualquiera de las mejoras en el modelo es necesario establecer los objetivos del proceso de validación, así como las actividades, tareas, calendarios y recursos disponibles que permiten conseguir eficientemente el propósito de los mismos, de lo contrario puede ocurrir que durante esta etapa no se descubran fallos en el modelo, lo cual implicaría una nueva revisión de éste.

[51] Computación Estadística – Curso 2008-2009. Internet: (wwwdi.ujaen.es/asignaturas/computacionestadistica/pdfs/tema6.pdf <http://wwwdi.ujaen.es/asignaturas/computacionestadistica/pdfs/tema6.pdf>)

[52] Laboratorios Docentes de la E.T.S.I. Informatica. Internet: (jair.lab.fi.uva.es/~pablfue/leng_simulacion/materiales/v_v_0405.pdf <http://jair.lab.fi.uva.es/~pablfue/leng_simulacion/materiales/v_v_0405.pdf>)

3.9.1. Generalidades

- **Dificultades de la validación.** Está fuera de toda duda que existe un cierto número de problemas cuando se intenta llevar a cabo el proceso de validación.

  - ✓ No existe la validación general, cada modelo se valida con respecto a sus objetivos. No se puede decir que un modelo válido para un propósito lo tenga que ser necesariamente para otros.

  - ✓ Hay una escasez de investigación sobre validación.

  - ✓ La literatura sobre validación es bastante pobre.

  - ✓ No existen procesos, mediciones, test, o métodos que aseguren que el modelo es válido.

  - ✓ La validación requiere la participación de mucha gente.

  - ✓ No existe un método o un conjunto de técnicas para validar un modelo.

- **Perspectivas de validación.** En la validación de modelos se deben tener en cuenta ciertas premisas para llevar a cabo un plan de validación de forma correcta, estas son:

  - ✓ Comúnmente se puede aprender mucho sobre la construcción de modelos, pero muy poco acerca de cómo asegurar la correctitud de nuestros modelos.

  - ✓ No existe un modelo absolutamente válido.

  - ✓ La validación es tratada incorrectamente como una actividad para realizar al final de un proyecto.

  - ✓ Algunas de las mejores literaturas consideran a la validación como un proceso.

  - ✓ La validación debería ser iniciada al comienzo de un proyecto.

  - ✓ La validación es tanto un ejercicio en relaciones humanas como un esfuerzo técnico.

  - ✓ El proceso de validación es más un arte que una ciencia.

  - ✓ Cuando no existe un plan general de validación, hay que ser muy metódico en los intentos de validación.

3.9.2. Antecedentes. Existen trabajos muy importantes en torno a validación de modelos, los cuales plantean que esta parte del proyecto es una de las tareas más importantes y difíciles en el desarrollo de un modelo. Estos trabajos están orientados a la validación de modelos de simulación, donde se utilizan una serie de estrategias propias para llevar a cabo esta actividad. A continuación se describen brevemente los que se consideran más relevantes:

La Universidad de Jaén[53], plantea que existen tres pasos para la construcción de un modelo de simulación: la validación del modelo conceptual, la validación del modelo lógico y la validación del modelo de ordenador.

La validación del modelo conceptual consiste en establecer si con la abstracción que hemos realizado sobre el sistema real, se podrá responder a las cuestiones planteadas. Dado que no hay un método estándar para la validación del modelo conceptual, existen una serie de aproximaciones que son útiles para establecer si los aspectos del sistema real, recogidos en el modelo conceptual son los importantes para el propósito de la simulación.

- **Representación de los sucesos del sistema:** el grafo de sucesos es un método que sirve para describir de forma gráfica los sucesos y los elementos de un sistema de eventos discretos. Dicha representación también puede resultar útil para la validación del modelo conceptual.

  En dicha representación en forma de grafo, los nodos representan los sucesos y los arcos las conexiones entre sucesos. Dichos arcos pueden ser de dos tipos indicando si la ocurrencia del suceso es incondicional o está condicionada al cumplimiento de ciertos aspectos.

- **Identificación explícita de los elementos que han de estar en el modelo:** en la mayoría de los casos, el modelo no puede contener cada detalle del sistema real, pero sí debe incluir aquellos elementos que sean relevantes para responder a las cuestiones planteadas. Debemos identificar todos los sucesos, facilidades, equipamiento, reglas de operación, variables de estado, variables de decisión y medidas de ejecución que van a formar parte del modelo de simulación.

  El modelo lógico sirve como puente entre el modelo conceptual y el modelo de ordenador. Si el modelo conceptual se ha construido bien, la validación del modelo lógico no es un proceso complejo.

[53] Computación Estadística – Curso 2008-2009. Op cit.

- **Validación del procesamiento de los sucesos**: el modelo lógico, para ser válido, ha de contener todos los sucesos incluidos en el modelo conceptual, así como la lógica para una correcta planificación de los sucesos futuros. Se debe validar que el modelo lógico contiene todos los sucesos del modelo conceptual, así como verificar las conexiones entre ellos. Por último se ha de validar que todas las variables de estado cambian correctamente ante la ocurrencia del suceso que las afectan. Un método que se puede utilizar para esto, es una revisión en la que el desarrollador del modelo lógico debe explicar con detalle la lógica del modelo a los otros miembros del proyecto de simulación.

Una vez que el modelo lógico se ha validado, se determinará si el modelo de ordenador es una correcta representación del sistema real. Un test para validar el sistema es ver si las personas relacionadas con el sistema confían en el modelo y están dispuestos a utilizarlo. La importancia de la credibilidad en el modelo es la mayor razón del interés tan difundido en realizar una animación de la salida de la simulación. Un modelo de un determinado sistema puede ser válido para un propósito y no ser válido para otro.

Comparación de los resultados de salida del modelo con los del sistema real: Este método se podrá aplicar en aquellos casos en los que el sistema exista y se pueda experimentar con él de forma que se obtengan datos de salida del mismo. Este método consiste en ejecutar el modelo y obtener una serie de datos de salida y comparar éstos, mediante algún método estadístico, con resultados que se tengan del sistema. Debemos comparar dos conjuntos de datos, de alguna forma, para determinar si el modelo es una representación adecuada del sistema real.

Una primera aproximación sería utilizar uno de los test estadísticos clásicos (como Chicuadrado o Kolmogorov-Smirnov para dos muestras) para determinar si las distribuciones subyacentes a los dos conjuntos de datos se pueden ver como la misma. Sin embargo, los procesos de salida de la mayoría de los sistemas reales y simulaciones no son estacionarios y son autocorrelados, de forma que ninguno de estos tests es directamente aplicable.

- **Método Delphi:** este método se utiliza cuando no se tienen datos o se dispone de muy pocos, acerca del sistema que se está considerando. En este método, se selecciona un grupo de expertos los cuales deben llegar a un consenso en las respuestas que den acerca de una serie de preguntas que se les plantean. En un entorno de simulación los expertos pueden ser los administradores y usuarios del sistema y las cuestiones son acerca del comportamiento del sistema bajo ciertas condiciones de operación. El método Delphi excluye las discusiones cara a cara entre los miembros del grupo.

- **Test de Turing:** Alan Turing sugirió este método como un test de inteligencia artificial. En este test, a un experto, o grupo de expertos, se le presentan resúmenes o informes de resultados de ejecución del sistema y del modelo, a

los que se les ha dado el mismo formato. Estos informes se reparten aleatoriamente a los ingenieros y administradores del sistema, para ver si son capaces de discernir cuáles son los reales del sistema y cuales la imitación resultado de la simulación. Si los expertos no son capaces de distinguir entre ambos, se puede concluir que no hay evidencias para considerar inadecuado al modelo. Si descubren diferencias las respuestas sobre lo que encuentran inconsistente se puede utilizar para realizar mejoras en el modelo.

- **Validación de comportamientos en casos extremos:** ocasionalmente se puede observar el comportamiento del sistema bajo condiciones extremas. Esta es una situación ideal para recoger datos de las medidas de ejecución del sistema real de forma que luego se puedan comparar con los resultados de la simulación, una vez que se ejecute el modelo bajo situaciones similares. También es posible que los expertos del sistema puedan predecir el comportamiento del sistema bajo condiciones extremas y utilizar estas predicciones para validar el modelo.

Naylor y Finger[54] proponen una técnica de 3 pasos para validación. El primer paso consiste en desarrollar un modelo que parezca razonable a la gente que conoce el sistema bajo estudio. El segundo hace referencia a testear empíricamente las suposiciones realizadas acerca de la operación del sistema. Y por último, determinar el nivel de representatividad de los datos producidos por la simulación.

Las tareas a seguir en el primer paso son:

- Obtener información de los “expertos” del sistema mediante entrevistas, no siempre la información estará documentada.

- Hacer observaciones del sistema.

- Recurrir a la teoría existente, por ejemplo, en un sistema de colas los tiempos entre arribos se sabe que responden a un proceso Poisson.
- Relevar resultados de modelos similares.

- Usar la experiencia y la intuición.

- Para hipotetizar sobre algunos aspectos del sistema, sobre todo cuando éste no existe.

[54] NAYLOR, Thomas H. FINGER J. M. MCKENNEY, James L., SCHRANK William E. HOLT, Charles C. Verification of Computer Simulation Models, *Management Science*, Vol. 14, No. 2, Application Series (Oct., 1967)

Las tareas a seguir en el segundo paso son:

- Testear cuantitativamente las suposiciones hechas en las etapas iníciales del desarrollo del modelo.

- Verificar si las distribuciones de probabilidad de datos de entrada hipotetizadas son adecuadas (usar gráficos, test de bondad).

- Análisis de sensibilidad: determinar si la salida de la simulación cambia significativamente cuando cambia el valor de un parámetro de entrada.

Las tareas a seguir en el tercer paso son:

- Se debe preguntar: ¿Los datos de salida son los esperados?

- Si se ha modelado un sistema existente, comparar los datos de salida del modelo con los del sistema real.

- Generalmente la comparación se basa en tests estadísticos. Si la comparación es favorable, el modelo se considera válido.

4. VALIDACIÓN DE MORENA

4.1. INTRODUCCIÓN

En este capítulo se presenta la validación del modelo de representación de patrones aspectuales de análisis. Inicialmente se muestran los elementos de validación identificados de acuerdo a sus componentes, posteriormente la estrategia de validación de acuerdo a sus características y finalmente el plan de validación que da un orden adecuado a la validación

4.2. ELEMENTOS DE VALIDACIÓN

Al realizar el estudio correspondiente de MORENA, se evidencia la importancia de validar todos sus elementos debido a que son piezas fundamentales en el modelo y es necesario determinar si en conjunto, permiten realizar la representación de patrones aspectuales de análisis de una manera adecuada. Además, a raíz de esta investigación, se estableció como valor agregado a la validación del modelo, la validación del proceso y la estructura del catálogo planteados por ESKEMA. Estas actividades se realizan durante la construcción y simulación del catálogo.

En la siguiente tabla se muestran los elementos a validar, el objetivo y momento de su validación:

Tabla 1: Elementos de validación de MORENA

Tipo	Elemento	Objetivo de validación	Momento de la validación
Proceso - Gestión del catálogo de patrones de análisis	Identificar y especificar requisitos de dominio	Validar si esta actividad permite definir los requisitos que especifican el dominio y que son insumo para la definición del patrón de análisis.	Construcción del catálogo

Proceso - Gestión del catálogo de patrones de análisis	Modelar patrón	Validar si esta actividad permite generar los modelos que definen al patrón	Construcción del catálogo
	Catalogar patrón	Validar si esta actividad permite integrar el nuevo patrón al catálogo a partir de la instanciación de los elementos de descripción del patrón	Construcción del catálogo
Representación	Modelo estructural	Validar si el modelo estructural permite representar cómo se compone y relaciona la estructura del patrón de análisis.	Construcción del catálogo
		Validar si el diagrama de clases permite representar la estructura conceptual del patrón de análisis.	
	Modelo de comportamiento	Validar si el modelo de comportamiento define la parte dinámica del patrón de análisis.	Construcción del catálogo
		Validar si el diagrama de secuencia describe la interacción de los objetos del patrón de análisis a través del tiempo.	

Representación	Modelo de alcance	Validar si el modelo de alcance permite definir como el patrón de análisis interactúa con su contexto y en especial con los usuarios de acuerdo con los roles que asumen.	Construcción del catálogo
		Validar si la colaboración representa de manera adecuada, cómo los roles asociados al patrón de análisis se relacionan entre sí.	
	Template	Validar si el template sirve como contenedor de los diagramas que representan los modelos que definen el patrón de análisis.	Construcción del catálogo
		Validar si el Signature permite instanciar el patrón	Simulación de uso y de retroalimentación del catálogo
Catálogo	Estructura	Validar si la estructura permite ejecutar los mecanismos de gestión de los patrones de análisis	Simulación de uso y de retroalimentación del catálogo

4.3. ESTRATEGIA DE VALIDACIÓN

A partir del estudio realizado de distintas estrategias de validación de modelos, se ha determinado plantear una estrategia propia para la validación de MORENA, debido a que no existen estrategias de validación generales sino que cada una depende de las características del modelo.

La estrategia de validación de los elementos de MORENA, se basa en la construcción, simulación de uso y retroalimentación del catálogo siguiendo el proceso de ESKEMA denominado: “gestionar el catálogo de patrones de análisis”, la estructura del catálogo y el modelo de representación.

4.4. PLAN DE VALIDACIÓN

Para realizar la validación de los elementos de MORENA y teniendo en cuenta lo mencionado en el marco teórico de este documento referente a la validación, se ha decido dividir este proceso en etapas para ordenar y optimizar el trabajo.

Una vez identificados los elementos de validación y definida una estrategia de validación, se procede a clasificarlos en etapas dependiendo de su orden de aplicación dentro de MORENA. Estas etapas se describen a continuación**:**

ETAPA 1: Construcción del Catálogo

I. Identificar y especificar requisitos de dominio
II. Modelar patrón
III. Modelo de alcance
IV. Modelo estructural
V. Modelo de comportamiento
VI. Catalogar patrón
VII. Template (como contenedor)

ETAPA 2: Simulación de uso y de retroalimentación del Catálogo

I. Estructura del catálogo
II. Template - Signature

Durante el proceso de validación, se realizarán tablas para comprobar si los elementos de MORENA cumplen con el objetivo planteado, describir los resultados obtenidos y especificar cuáles generan una sugerencia de mejoramiento para MORENA.

5. DOMINIO DE APLICACIÓN POS

5.1. INTRODUCCIÓN

Uno de los sectores económicos más importantes a nivel mundial es el sector comercio, en cual se llevan a cabo procesos sistematizados y correctos. Dentro del sector comercio las organizaciones que realizan tareas de fabricación, distribución y mercadeo de productos, basan sus objetivos en la venta de los mismos, por esto, este proceso toma importancia.

Dentro del proceso de la venta, el POS como punto interno de la organización donde se desarrolla la venta de productos y/o servicios es de gran importancia, siendo éste el punto donde el cliente realiza la transacción final para obtener el producto. Por este motivo, este proyecto de investigación identifica el POS como el dominio de aplicación.

En este capítulo se define el sector económico comercio, el proceso de venta y finalmente, se describe el dominio de aplicación POS; indicando sus tipos, sistemas y requisitos.

5.2. SECTOR ECONÓMICO COMERCIO

Como bien lo dice Juan Esteban Giraldo[55], el sector comercio se encarga de la prestación de servicios de distribución, canalización y venta de todo tipo de bienes tanto, a nivel nacional como internacional; integra un conjunto de actividades que tienen como objetivo el intercambio de bienes y servicios, y que ponen en contacto al productor (oferta) y al consumidor (demanda).

El comercio nació como necesidad de intercambiar el excedente agrícola por otros productos, inicialmente se realizaba mediante “trueque” (intercambio de productos), luego surgió la moneda.

El comercio se clasifica según su ámbito en comercio interior y comercio exterior. El comercio interior se desarrolla dentro de las fronteras de un estado, tiene como

[55] GIRALDO, Juan Esteban. El comercio en la economía, 2002. Internet: (www.gestiopolis.com/recursos/documentos/fulldocs/eco/comeco.htm <http://www.gestiopolis.com/recursos/documentos/fulldocs/eco/comeco.htm>)

función distribuir la producción y las importaciones de un país y depende para su funcionamiento de un sistema de comunicaciones moderno; el comercio exterior es el intercambio de bienes y servicios entre los diferentes países del mundo.

Dentro del sector comercio, la venta es el proceso más importante ya que es donde se cumplen los objetivos de las organizaciones como tal.

5.3. PROCESO DE VENTA

En toda empresa existe un objetivo fundamental hacia el que están enfocados prácticamente todos los esfuerzos de la organización y a cuyo servicio se ponen todos los sistemas que dispone la empresa. Dicho objetivo no es otro que la venta de los productos y/o servicios. La venta activa toda una serie de procesos administrativos como impuestos pagos de bonificaciones por venta, inventarios, compras, etc.

Para la American Marketing Asociation[56], el proceso de venta es toda actividad que incluye un proceso personal o impersonal mediante el cual, el vendedor identifica las necesidades y/o deseos del comprador, genera el impulso hacia el intercambio y satisface las necesidades y/o deseos del comprador (con un producto, servicio u otro) para lograr el beneficio de ambas partes.

Dentro del proceso de la venta hay un punto fundamental que es llamado POS, en este punto se desarrollan todas las transacciones activadas por la venta.

5.4. DESCRIPCION DEL DOMINIO DE APLICACIÓN POS

5.4.1. Definición de dominio de aplicación POS. El POS hace referencia al lugar dentro de una empresa o negocio en donde se efectúa el proceso de salida y cobro de los productos o servicios que ofrece. En otras palabras es el lugar donde se culmina la operación de venta, es decir, donde el Cliente, paga el importe por un producto o servicio que está adquiriendo y se registra dicha transacción, dándole al cliente un recibo, comprobante o factura que ampara el pago dado por dicho producto o servicio. Son puntos de especial cuidado, porque en éstos están los soportes de ingreso de cualquier actividad económica lucrativa.

[56] American Marketing Asociation. MarketingPower.com, sección Dictionary of Marketing Terms. Internet (www.marketingpower.com <http://www.marketingpower.com/>)

Inmaculada José Martínez Martínez[57] afirma que el POS adquiere una dimensión superior al simple hecho de ser el lugar físico del intercambio comercial, el establecimiento por sí mismo es capaz de generar y transmitir emociones, sensaciones, sentimientos y experiencia, crear ambientes socioculturales, agudizar los sentidos con el objetivo último de favorecer la compra y el hábito de consumo. El POS en su globalidad actúa como un poderoso medio o canal de comunicación que transmite informaciones de forma ininterrumpida y es capaz de influir en su público. El cliente no sólo compra en un establecimiento, sino también se informa, se educa, se entretiene y, sobre todo, recibe toda clase de estímulos.

El POS es el lugar donde convergen los intereses de todos los actores implicados en la trama comercial, es decir la industria, los distribuidores y los compradores/consumidores finales.

Cada uno de los actores comerciales tiene unos intereses particulares: el fabricante pretende vender más productos de su marca, el distribuidor busca la mayor rentabilidad posible y el cliente busca una mejor información, una mayor calidad, un mejor precio, mayores servicios, etc. Por tanto se considera viable tomar el POS como dominio ya que se convierte en el centro convergente de los fabricantes con sus productos, los comerciantes con su gestión y los clientes con sus necesidades y deseos de compra.

5.4.2. Tipos de POS. Es posible realizar una clasificación de los POS existentes en el sector comercial de acuerdo a las funcionalidades requeridas, al tamaño de la organización y a la actividad comercial desempeñada por la empresa. Los POS pueden clasificarse en cuatro categorías:

- **Supermercados y autoservicios:** en este grupo se reúnen las necesidades de hipermercados, supermercados y autoservicios.

- **Tiendas y negocios de venta minorista:** se especializa en el sector de pequeños negocios, brindándoles un manejo eficiente de su actividad comercial. En esta categoría se pueden incluir negocios tales como droguerías, disco tiendas, librerías, etc.

- **Gastronomía:** orientado a negocios que trabajan con la preparación y venta de alimentos tales como restaurantes, bares y negocios de comidas rápidas.

- **Clubes:** en esta categoría se manejan los negocios de carácter social, cultural o deportivo en su mayoría de tipo privado. Los servicios y productos que estos

[57] MARTÍNEZ, I.J. (2005): La comunicación en el punto de venta: estrategias de comunicación en el comercio real y on-line. ESIC, Madrid.

negocios ofrecen son: restaurante, bar, gimnasio, hospedaje, membrecías, eventos, entre otros.

5.4.3. Sistemas POS. Los negocios cuya actividad económica es la venta de gran número de productos y variedades de manera directa a sus clientes, requieren de un mecanismo que les permita mantener un control permanente sobre las ventas, el inventario y la interacción con sus proveedores y clientes; este mecanismo es llamado Sistema POS.

El Sistema POS es un sistema compuesto por software y hardware, creado con el fin de automatizar y agilizar los procesos relacionados con ventas y las demás actividades que puedan depender de ella. Se puede afirmar que el objetivo de la implantación de este tipo de sistemas dentro de una empresa, es mejorar la funcionalidad de los procesos y poder tener un control sobre información clave de manera consistente y actualizada. Además el sistema POS debe permitir una interacción sistema-usuario a través de una interfaz agradable y de fácil uso.

Londoño[58] presenta algunos ejemplos de sistemas POS que actualmente se encuentran en el mercado de software, tales como: Aloha POS, el cual satisface los requisitos particulares de un restaurante o de un bar; RETAIL, que es una herramienta de software para venta al detal; Tina POS, aplicación útil para comercios detallistas; Checkout POS y Checkout Restaurant, que integran el punto de venta y la administración en una misma base de datos replicada.

- **Funcionalidades de los sistemas POS.** Las funcionalidades que el sistema brinde deben acoplarse a las necesidades de cada empresa. Estas funcionalidades se identificaron a partir de los sistemas POS mencionados anteriormente.

  - ✓ **Supermercados y autoservicios.** Dentro de las funcionalidades que un sistema POS debe brindar a este tipo de Puntos de Venta, encontramos:

    Facturación:

    - o Manejo de multiprecios (divisas)
    - o Manejo de descuentos por valor o porcentaje, por productos en promoción, por grupos de productos, por horas, por fechas, etc.
    - o Cotizaciones.
    - o Control de domicilio
    - o Cambios de productos
    - o Impresión de copias

[58] LONDOÑO, Luis Fernando. Definición de las Necesidades para la Construcción de un Sistema de Puntos De Venta, Universidad EAFIT, 2008.

- Anulación de facturas
- Devolución de clientes
- Control de mercancía separada
- Manejo de pedidos de clientes
- Control de gastos de caja
- Creación y configuración de de medio para venta (Domicilios, etc.)
- Creación y configuración de medio de pago (Tarjetas de Crédito, Débito, etc.)
- Manejo de bonos.

Inventario:

- Seguimiento rápido al movimiento de productos
- Control de mercancía
- Control de mercancía en consignación
- Control de mercancía separada
- Conversión de desempaque.
- Actualización de precios
- Control de vencimiento de productos

Cartera:

- Control de pagos y abonos
- Consulta de saldos

Estadísticas:

- Estadísticas de las funciones propias de cada negocio
- Generación de reportes de las funciones propias de cada negocio

Presupuesto:

- Proyecciones y control de logro de metas.

Contabilidad:

- Manejo de impuestos
- Cuadres diarios
- Cuentas por pagar
- Cuentas por cobrar
- Generación de informe fiscal (DIAN)

Empleados:

- Manejo de comisiones.

Clientes:

- Registrar venta por cliente
- Registrar información básica
- Manejo de programa de fidelización

Proveedores:

- Registrar información

Comunicaciones:

- Manejo de puntos de cadena
- Manejo de sucursales
- Manejo de dependencias

Información:

- Soporte de alto volumen de transacciones
- Administración de información con seguridad y confiabilidad
- Asignación de perfiles
- Auditoría
- Copias de seguridad.
- Consultas básicas de productos y clientes

- ✓ **Tiendas y negocios de ventas minoristas.** Entre las funcionalidades requeridas por este tipo de POS se tiene:

Facturación:

- Control de vencimiento de productos
- Manejo de remisiones
- Generar facturación
- Creación y configuración de tipos de medio para venta
- Control de domicilio
- Control de mercancía separada
- Manejo de bonos.
- Manejo de descuentos por valor o porcentaje, por productos en promoción, por grupos de productos, por horas, por fechas, etc.

Inventario:

- Agrupación de Productos
- Control de mercancía

Cartera:

- Control de pagos y abonos
- Consulta de saldos

Estadísticas:

- Generación de reportes de las funciones propias de cada negocio

Contabilidad:

- Arqueo de caja
- Cuentas por pagar

Clientes:

- Registrar información básica

Proveedores:

- Registrar información

Información:

- Consultas básicas de productos y clientes

✓ **Gastronomía**. Entre sus requisitos están:

Facturación:

- Manejo de multiprecios (divisas)
- Control de domicilio
- Emisión rápida de facturas
- Apertura, adición y cierre de mesas
- Creación y configuración de de medio para venta (Domicilios, etc.)
- Creación y configuración de medio de pago (Tarjetas de Crédito, débito, etc.)

Inventario:

- Seguimiento rápido al movimiento de productos
- Control de mercancía
- Conversión de desempaque.
- Actualización de precios
- Control de vencimiento de productos

Estadísticas:

- Estadísticas de las funciones propias de cada negocio
- Generación de reportes de las funciones propias de cada negocio

Producción:

- Control de costos en productos terminados de producción y proceso
- Administración eficiente de inventarios de materia prima y producto terminado
- Manejo de órdenes de producción

Presupuesto:

- Proyecciones y control de logro de metas.

Contabilidad:

- Cuentas por pagar
- Arqueo de caja

Clientes:

- Registrar información básica
- Manejo de programa de fidelización

Proveedores:

- Registrar información

Información:

- Consultas básicas de productos y clientes
- Auditoría

- ✓ **Clubes**. Para dar un adecuado manejo a los servicios y productos que estos negocios ofrecen, tales como restaurante, bar, gimnasio, hospedaje, membrecías, eventos, entre otros; se requiere de las siguientes características:

  Facturación:

  - o Manejo de multiprecios (divisas)
  - o Control de domicilio
  - o Emisión rápida de facturas
  - o Apertura, adición y cierre de mesas
  - o Facturación de membrecías, eventos y hospedaje.
  - o Creación y configuración de de medio para venta (Domicilios, etc.)
  - o Creación y configuración de medio de pago (Tarjetas de Crédito, débito, etc.)

  Inventario:

  - o Seguimiento rápido al movimiento de productos
  - o Control de mercancía
  - o Conversión de desempaque.
  - o Actualización de precios
  - o Control de vencimiento de productos

  Cartera

  - o Control de pagos y abonos
  - o Consulta de saldos

  Estadísticas

  - o Estadísticas de las funciones propias de cada negocio
  - o Generación de reportes de las funciones propias de cada negocio

  Producción

  - o Control de costos en productos terminados de producción y proceso
  - o Administración eficiente de inventarios de materia prima y producto terminado
  - o Manejo de órdenes de producción

Presupuesto

- Proyecciones y control de logro de metas.

Contabilidad:

- Cuentas por pagar
- Arqueo de caja
- Generación de informe fiscal (DIAN)
- Manejo de impuestos

Empleados

- Manejo de comisiones.

Clientes

- Registrar información básica
- Registrar venta por cliente
- Manejo de programa de fidelización
- Controlar posibles prospectos de miembros y beneficiarios

Proveedores:

- Registrar información

Comunicaciones:

- Manejo de dependencias

Información:

- Soporte de alto volumen de transacciones
- Administración de información con seguridad y confiabilidad
- Asignación de perfiles
- Auditoría
- Copias de seguridad.
- Consultas básicas de productos y clientes

5.4.4. Requisitos del dominio POS. A partir de los sistemas POS existentes en el mercado y las funcionalidades anteriormente descritas, se identificaron los requisitos de dominio, los cuales serán clasificados en: funcionales, no funcionales, de información y reglas de negocio.

- **Requisitos funcionales**

Tabla 2: Requisitos funcionales del dominio POS

Requisito	Descripción
Manejo de descuentos y promociones	Realizar la aplicación y el registro de los descuentos y las promociones que se apliquen en un centro de venta.
Registrar venta por cliente	Realizar el registro de los productos que compra cada cliente por venta, para futuros usos en promociones, descuentos o toma de decisiones.
Manejo Inventario	Realizar el envío de información para actualización de inventario.
Integración sistemas de pago con tarjeta	Realizar la integración de los diferentes sistemas con los sistemas externos que permiten el pago por medio de tarjeta debito o crédito.
Conversión a moneda extranjera	Calcular la equivalencia del costo de un producto comercializado en moneda extrajera.
Realizar los movimientos de caja	Llevar un control sobre los flujos de entrada y salida de caja, registrando la forma de éstos flujos (cheques, efectivo).
Generar reportes de ventas	Generar un reporte (impreso, archivo, pantalla) de los movimientos básicos de ventas.
Integrar la información	Integración de la información de los diferentes puntos de venta de la empresa con la administración central en tiempo real, en una misma base de datos replicada.
Consulta de productos	Realizar la búsqueda de productos por sus diferentes características, desplegando toda la información relacionada al producto, de esta forma se puede ver la disponibilidad de productos o búsqueda especifica de un producto.
Realizar arqueo de caja	Permitir un proceso en el que se totalice las ventas llevadas hasta ese momento y realizar un comparativo con el efectivo.
Reporte de la información totalizada	Realizar un reporte que contenga el total vendido, el total de costos, el total de impuestos y productos que se venden.
Realizar devoluciones	Realizar la respectiva devolución de un producto que ya fue vendido; enviando actualización a inventario, anulando factura y retornando pago.
Gestionar factura	Crear y hacer persistencia de la factura, la cual deberá contener la información que se considere

Requisito	Descripción
	pertinente. Al finalizar el proceso el sistema deberá emitir la factura siguiendo los parámetros de ley.
Validar forma de pago	Validar las diferentes formas de pago y realizar las operaciones correspondientes.
Gestionar acciones	Permitir programar e implementar acciones, las cuales podrán ser completadas, canceladas o suspendidas por un periodo de tiempo.

- **Requisitos no funcionales**

Tabla 3: Requisitos no funcionales del dominio POS

Requisito	Descripción
Soportar varios idiomas	Funcionamiento multi-idioma, de tal manera que puedan añadirse nuevos idiomas fácilmente.
Interfaz gráfica de usuario	El sistema debe ser gráfico, con uso mínimo del teclado para las funcionalidades que requieren atención al cliente.
Interfaz de fácil manejo	La interfaz debe ser de fácil manejo, para que así el sistema no requiera personal muy especializado para su uso.
Manejar un corto tiempo de respuesta	El sistema deberá entregar la información requerida y realizar las operaciones internas de manera ágil para ofrecer un buen servicio.
Soportar alto volumen de transacciones	Manejar un sistema gestor de base de datos que permita el soporte de un alto número de transacciones brindando integración y coherencia en el manejo de la información.
Manejo de información con altos niveles de seguridad	Brindar acceso a la información según el nivel de autoridad definido en la empresa (por medio de módulos de autorización incluidos en el sistema).
Realizar copias de seguridad de bases de datos	Realizar una copia de seguridad de los datos de manera periódica establecida por el usuario.

- **Requisitos de información**

Tabla 4: Requisitos de información del dominio POS

Requisito	Descripción
Almacenar los periodos especiales de venta	Almacenamiento de la información relacionada con los periodos de venta que tienen asociadas promociones o descuentos (Día de la madre, día del padre, navidad, etc.).
Almacenar los impuestos de venta	Almacenamiento de la información relacionada con los impuestos a las ventas de cada producto.
Almacenar información de ventas	Almacenamiento de información de productos vendidos, cantidades, días, fechas y horas.
Almacenar información de cliente y productos comprados	Almacenamiento de información de la relación Cliente y productos comprados.
Transmitir datos	Permitir la transmisión de datos entre los diferentes módulos del sistema.

- **Reglas de negocio**

Tabla 5: Reglas de negocio del dominio POS

Requisito	Descripción
Calculo de impuestos en la venta	Calcular el valor que tiene los impuestos asociados a los productos registrados en la venta.
Calculo del “regreso/cambio” del pago de la venta	Calcular el valor del “regreso/cambio” generado por el pago de la venta.
Descuentos	Definir cuáles son los porcentajes de descuento que se van a dar en cada promoción o evento que así lo amerite.

6. TRIPODE: UN CATÁLOGO DE PATRONES ASPECTUALES DE ANALISIS PARA EL DOMINIO POS BASADO EN MORENA.

6.1. INTRODUCCIÓN

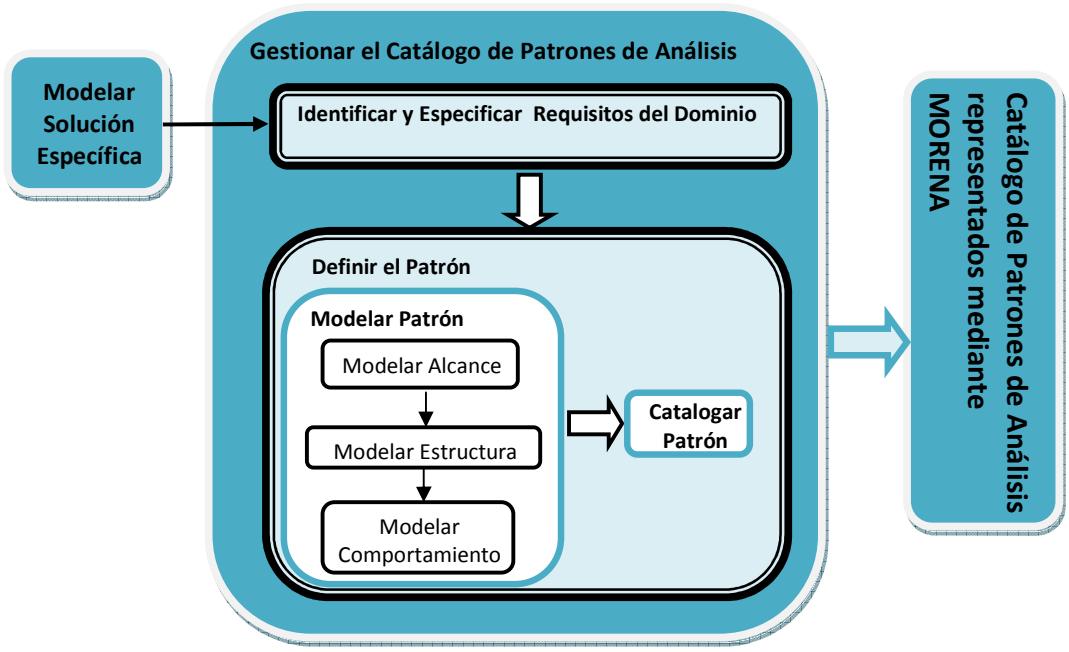
Para el logro del objetivo que plantea esta investigación, se seguirá el plan de validación de MORENA que se divide en dos etapas, descrito en el capítulo 4.

En este capítulo se muestra la construcción del catálogo siguiendo el proceso y estructura que propone ESKEMA, la simulación de uso y de retroalimentación del catálogo y los resultados de validación en cada una de estas etapas.

6.2. PROCESOS DE INGENIERIA DE DOMINIO PROPUESTO POR ESKEMA

El proceso de ingeniería de dominio propuesto por ESKEMA recibe el nombre de: “gestionar el catálogo de patrones de análisis”, el cual se describe en la siguiente figura:

Figura 28: Gestionar el catálogo de patrones de análisis



Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

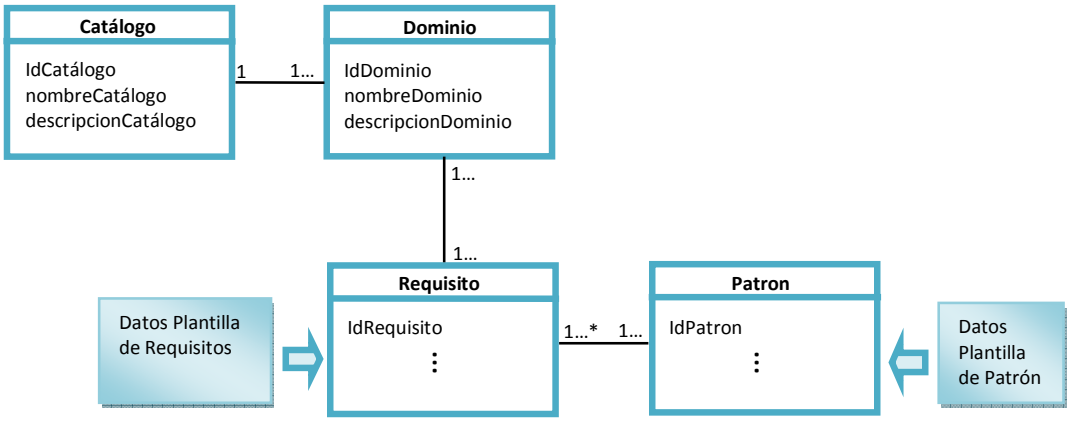
6.2.1. Identificar y especificar requisitos del dominio. Esta acción busca definir los requisitos que especifican el dominio y que son insumo para la definición del patrón de análisis.

6.2.2. Definir el patrón. Esta acción busca modelar el patrón e integrarlo al catálogo. Con base en los requisitos se define el patrón de análisis.

6.3. ESTRUCTURA DEL CATÁLOGO DE PATRONES DE ANÁLISIS

Para la estructura del catálogo de patrones de análisis, ESKEMA propone la siguiente:

Figura 29: Estructura del catálogo de patrones de análisis



Barón, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010

La figura 29, describe como nodo raíz el catálogo, en seguida sus dominios integrantes y para cada uno de ellos los patrones y requisitos de dominio asociados.

6.4. PROCESO DE CONSTRUCCIÓN DEL CATÁLOGO: PATRÓN FACTURA

Para la creación del patrón Factura se identifica y especifica el requisito de dominio Gestionar Factura, luego se modela el patrón por medio de la creación de la colaboración, diagrama de clases y diagrama de secuencia; finalmente se lo cataloga.

6.4.1. Identificar y especificar requisitos de dominio

Tabla 6: Descripción del requisito de dominio Gestionar factura

FRQ-011	Gestionar factura
Versión	1.1
Autores	Gloria Erazo, Germán Peña, Iván Cisneros
Fuentes	Supervisor de caja
Descripción	El sistema deberá crear y hacer persistencia de la factura, la cual deberá contener la información que se considere pertinente. Por ejemplo: • Código de la factura, nombre e identificación de la empresa, código y nombre del cajero, número de caja, fecha y hora de atención. • Ítem: nombre del producto o servicio, código, IVA, cantidad, valor unitario, valor total que corresponde al valor unitario por la cantidad de unidades. • Subtotal: diferencia entre el valor total de la factura y el valor total del IVA. • IVA: impuesto sobre el valor agregado de cada producto o servicio. • Total: sumatoria de los valores totales de los ítems • Pago: se indican las formas de pago y sus respectivos valores. • Cliente: se mostrará el número de identificación y el nombre del cliente**.** Al finalizar el proceso el sistema deberá emitir la factura siguiendo los parámetros de ley.
Dominios asociados	POS Comercio
Patrón solución	Factura
Estado	Pendiente de verificación
Estabilidad	Media
Comentarios	Ninguno

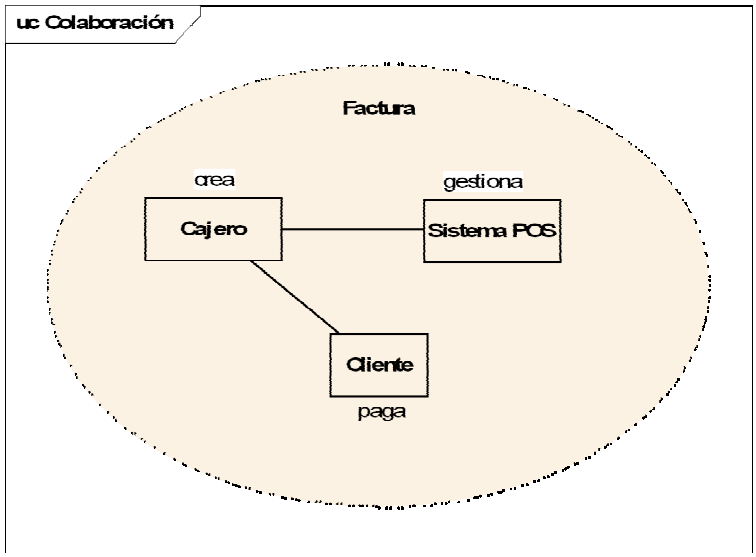
6.4.2. Definir el patrón

- **Modelar patrón**
  - ✓ **Modelar alcance**

Tabla 7: Asignación de responsabilidades patrón Factura

RESPONSABILIDAD	CLASE RESPONSABLE	ACTOR RESPONSABLE
Vincular ítem	Ítem	Cajero
Actualizar total factura	Factura	Sistema POS
Registrar pago	Pago	Cajero Cliente
Imprimir factura	Factura	Sistema POS

Figura 30: Colaboración patrón Factura



- **Actores**
  - **Cajero:** representa el funcionario que opera la caja donde se desarrolla la operación de venta.
  - **Cliente:** representa la persona u organización que adquiere los productos y/o servicios ofrecidos por la empresa.

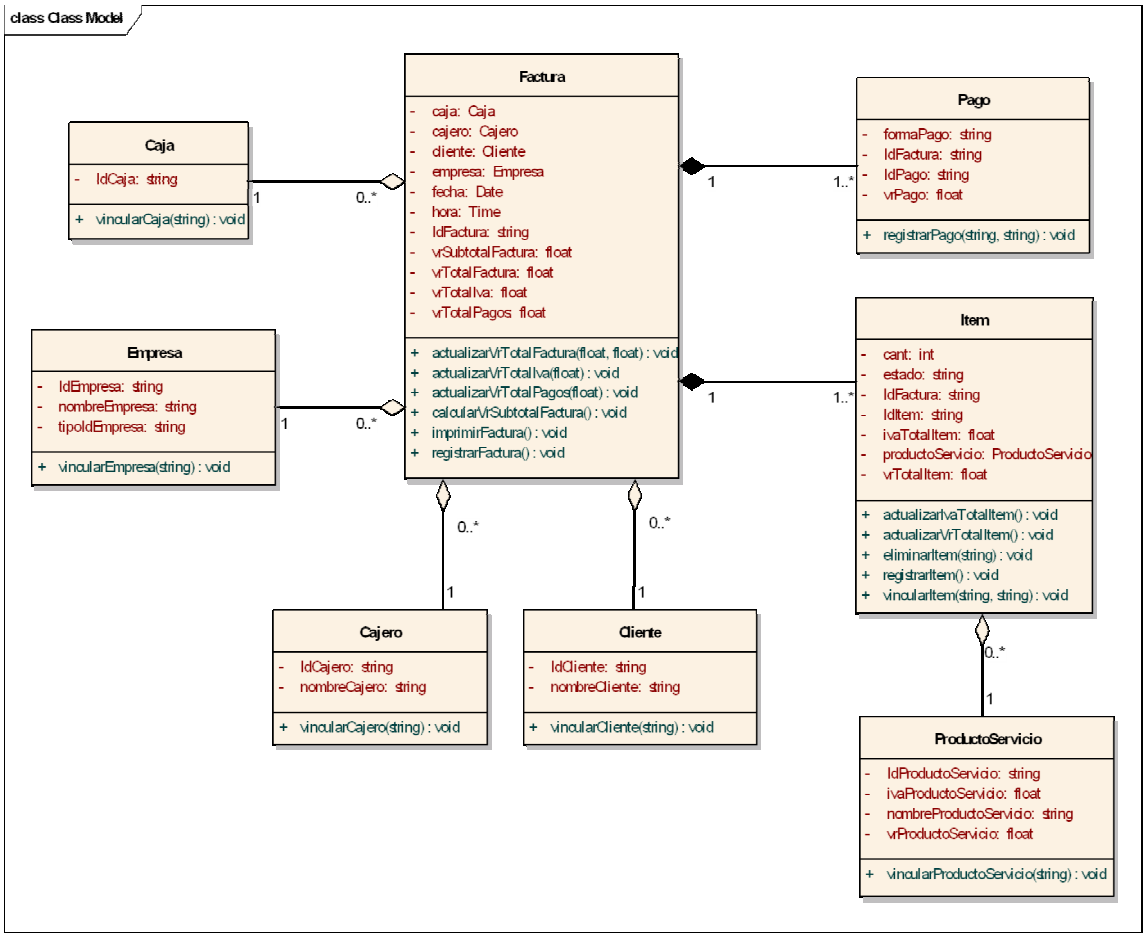
- **Sistema POS:** representa el software que gestiona el servicio de venta de productos y/o servicios en la empresa.

o **Roles**

- **Crear:** representa la función de recolectar la información de la operación de venta y configurar la factura.
- **Gestionar:** representa la función de recibir, procesar, almacenar e imprimir la información referente a la operación de venta.
- **Pagar:** representa la función de realizar un desembolso por la adquisición de productos y/o servicios.

✓ **Modelar estructura**

Figura 31: Diagrama de clases del patrón Factura



- **Clase Factura:** describe conceptualmente los documentos que contienen la información de una operación de venta de productos y/o servicios.

  Atributos

  - **caja:** representa el punto de venta donde se desarrolla la operación.
  - **cajero:** representa el funcionario que opera la caja donde se desarrolla la operación.
  - **cliente:** representa la persona u organización que adquiere los productos y/o servicios ofrecidos por la empresa.
  - **empresa:** representa la persona u organización que ofrece los productos y/o servicios.
  - **fecha:** representa la fecha de creación de la factura.
  - **hora:** representa la hora de creación de la factura.
  - **IdFactura:** representa el código de identificación de la factura.
  - **vrSubtotalFactura:** representa la diferencia entre el valor total de la factura y el valor total del IVA de la factura.
  - **vrTotalFactura:** representa la sumatoria de los valores totales de los ítems vendidos al cliente.
  - **vrTotalIva:** representa la sumatoria del IVA total de los ítems vendidos al cliente.
  - **vrTotalPagos:** representa la sumatoria de pagos realizados por el cliente en la operación de venta.

  Métodos

  - **actualizarVrTotalFactura:** representa la operación que permite acumular el valor total de los ítems vendidos al cliente.
  - **actualizarVrTotalIva:** representa la operación que permite acumular el valor total del IVA de los ítems vendidos al cliente.
  - **actualizarVrTotalPagos:** representa la operación que permite acumular el valor de los pagos realizados por el cliente en la operación.
  - **calcularVrSubtotalFactura:** representa la operación que permite restar el valor total del IVA de la factura al valor total de la factura.
  - **imprimirFactura:** representa la operación que permite imprimir la factura.
  - **registrarFactura:** representa la operación que permite hacer persistencia de la factura.

- **Clase Caja:** describe conceptualmente los puntos de venta donde se desarrollan las operaciones de venta.

Atributos

- **IdCaja:** representa el código de identificación de la caja.

Métodos

- **vincularCaja:** representa la operación que permite relacionar la caja con una factura.

o **Clase Empresa:** describe conceptualmente las personas u organizaciones que ofrecen los productos y/o servicios.

Atributos

- **IdEmpresa:** representa el código de identificación de la empresa.
- **nombreEmpresa:** representa el nombre de la empresa.
- **tipoIdEmpresa:** representa el tipo de identificación de la empresa. Por ejemplo: nit, cedula del representante, etc.

Métodos

- **vincularEmpresa:** representa la operación que permite relacionar la empresa con una factura.

o **Clase Cajero:** describe conceptualmente los funcionarios que operan las cajas donde se desarrollan las operaciones de venta.

Atributos

- **IdCajero:** representa el código de identificación del cajero.
- **nombreCajero:** representa el nombre del cajero.

Métodos

- **vincularCajero:** representa la operación que permite relacionar un cajero con una factura.

o **Clase Cliente:** describe conceptualmente las personas u organizaciones que adquieren los productos y/o servicios ofrecidos por las empresas.

Atributos

- **IdCliente:** representa el código de identificación del cliente.
- **nombreCliente:** representa el nombre del cliente.

Métodos

- **vincularCliente:** representa la operación que permite relacionar un cliente con una factura.

- **Clase Pago:** describe conceptualmente los desembolsos que realizan los clientes por la adquisición de los productos y/o servicios ofrecidos por las empresas.

  Atributos

  - **formaPago:** representa la manera en la que se realiza el pago. Por ejemplo: efectivo, tarjeta crédito, tarjeta debito, bono, etc.
  - **IdPago:** representa el código de identificación del pago.
  - **IdFactura:** representa el código de identificación de la factura a la cual está vinculado el pago.
  - **vrPago:** representa el valor del pago.

  Métodos

  - **registrarPago:** representa la operación que permite vincular el pago a una factura y almacenar su información.

- **Clase Item:** describe conceptualmente uno o un conjunto del mismo producto o servicio.

  Atributos

  - **cant:** representa la cantidad de productos o servicios por ítem.
  - **estado:** representa un estado que determina si el ítem ha sido adicionado o eliminado de la factura.
  - **IdFactura:** representa el código de identificación de la factura a la cual está vinculado el ítem.
  - **IdItem:** representa el código de identificación del ítem.
  - **ivaTotalItem:** representa la sumatoria del IVA de los productos o servicios del ítem.
  - **productoServicio:** representa el producto o servicio vinculado al ítem.
  - **vrTotalItem:** representa la sumatoria de los valores de los productos o servicios del ítem.

Métodos

- **actualizarIvaTotalItem:** representa la operación que permite acumular el IVA de los productos o servicios del ítem.
- **actualizarVrTotalItem:** representa lo operación que permite acumular el valor de los productos o servicios del ítem.
- **eliminarItem:** representa lo operación que permite cambiar el estado del ítem a eliminado y borrarlo de la factura.
- **registrarItem:** representa la operación que permite almacenar la información del ítem.
- **vincularItem:** representa la operación que permite relacionar el ítem con una factura.

o **Clase ProductoServicio:** describe conceptualmente los productos o servicios ofrecidos por las empresas.

Atributos

- **IdProductoServicio:** representa el código de identificación del producto o servicio.
- **ivaProductoServicio:** representa el IVA del producto o servicio.
- **nombreProductoServicio:** representa el nombre del producto o servicio.
- **vrProductoServicio:** representa el valor del producto o servicio.

Métodos

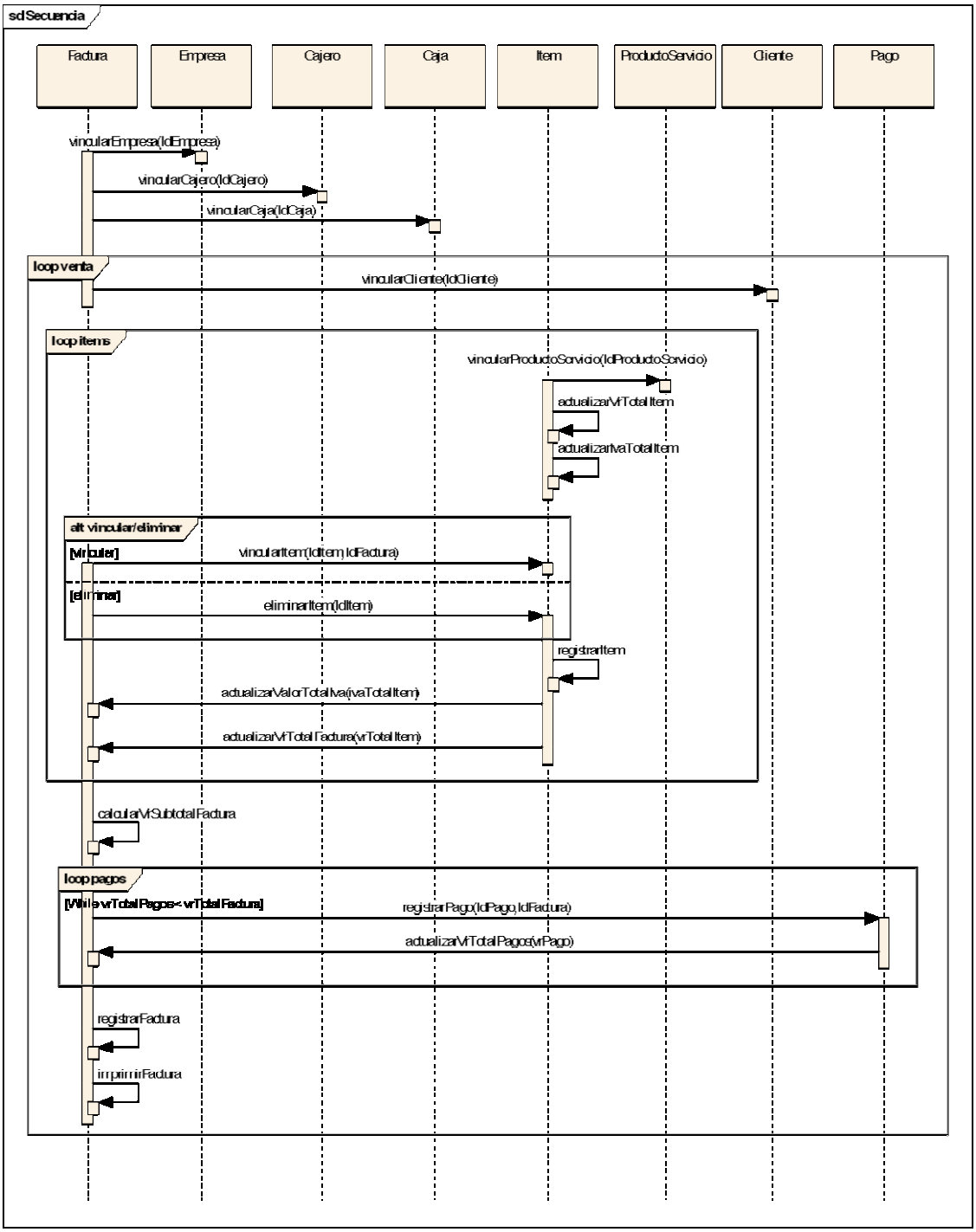
- **vincularProductoServicio:** representa la operación que permite relacionar un producto o servicio con un ítem.

o **Cardinalidad**

- Una factura tiene una caja, una caja puede tener cero o más facturas.
- Una factura tiene una empresa, una empresa puede tener cero o más facturas.
- Una factura tiene un cajero, un cajero puede tener cero o más facturas.
- Una factura tiene un cliente, un cliente puede tener cero o más facturas.
- Una factura tiene uno o más pagos, un pago tiene una factura.
- Una factura tiene uno o más ítems, un ítem tiene una factura.
- Un ítem tiene un producto o servicio, un producto o servicio puede tener cero o más ítems.

- ✓ **Modelar comportamiento**

Figura 32: Diagrama de secuencia del patrón Factura



- **Objetos**

  - **Factura:** representa el documento concreto que contiene la información de una operación de venta de productos y/o servicios.
  - **Empresa:** representa la organización o persona concreta que ofrece productos y/o servicios.
  - **Cajero:** representa el funcionario concreto que opera la caja donde se desarrollan las operaciones de venta.
  - **Caja:** representa el punto de venta concreto donde se desarrollan las operaciones de venta.
  - **Ítem:** representa concretamente uno o un conjunto del mismo producto o servicio.
  - **ProductoServicio:** representa el producto o servicio concreto ofrecido por la empresa.
  - **Cliente:** representa la persona u organización concreta que adquiere los productos y/o servicios ofrecidos por la empresa.
  - **Pago:** representa el desembolso concreto que realiza el cliente por la adquisición de los productos y/o servicios ofrecidos por la empresa.

- **Mensajes**

  - **actualizarIvaTotalItem:** representa el proceso de acumular el IVA de los productos o servicios del ítem. Este proceso se ejecuta por cada unidad de producto o servicio vendido.
  - **actualizarVrTotalFactura:** representa el proceso de acumulación del valor total de los ítems adquiridos en la operación de venta. Este proceso se ejecuta por cada ítem.
  - **actualizarVrTotalItem:** representa el proceso de acumular el valor de los productos o servicios del ítem. Este proceso se ejecuta por cada unidad de producto o servicio vendido.
  - **actualizarVrTotalIva:** representa el proceso de acumular el valor total del IVA de los ítems adquiridos en la operación de venta. Este proceso se ejecuta por cada ítem.
  - **actualizarVrTotalPagos:** representa el proceso de acumular el valor de los pagos realizados por el cliente en la operación de venta.
  - **calcularVrSubtotalFactura:** representa el proceso de restar el valor total del IVA de la factura al valor total de la factura. Este proceso se ejecuta una sola vez por cada venta.
  - **eliminarItem:** representa el proceso de cambiar el estado del ítem a eliminado y borrarlo de la factura.
  - **imprimirFactura:** representa el proceso de imprimir la factura una vez terminada la operación de venta.

- **registrarFactura:** representa el proceso de hacer persistencia de la factura.
- **registrarItem:** representa el proceso de almacenar la información del ítem vendido.
- **registrarPago:** representa el proceso de vincular el pago a la factura y almacenar su información.
- **vincularCaja:** representa el proceso de relacionar la caja con la factura. Este proceso se ejecuta una sola vez, al inicio de la jornada laboral.
- **vincularCajero:** representa el proceso de relacionar el cajero con la factura. Este proceso se ejecuta una sola vez, al inicio de la jornada laboral.
- **vincularCliente:** representa el proceso de relacionar el cliente con la factura en la operación de venta.
- **vincularEmpresa:** representa el proceso de relacionar la empresa con la factura. Este proceso se ejecuta una sola vez, al inicio de la jornada laboral.
- **vincularItem:** representa el proceso de relacionar el ítem vendido con la factura. Este proceso se ejecuta por cada ítem.
- **vincularProductoServicio:** representa el proceso de relacionar el producto o servicio con el ítem. Este proceso se ejecuta por cada producto o servicio.

o **Ciclos.** Los procedimientos de vinculación de caja, cajero y empresa se realizan únicamente al inicio de la jornada, por esto no son incluidos en los ciclos de facturación.

loop venta: representa el ciclo mayor, en el cual se realizan los procedimientos de:

- Vinculación de productos y/o servicios a los ítems.
- Actualización del valor total de los ítems.
- Actualización de IVA total de los ítems.
- Vinculación o eliminación de ítems en la factura.
- Registro de ítems.
- Actualización del valor total de la factura.
- Actualización del valor total del IVA de la factura.
- Cálculo del valor subtotal de la factura.
- Registro de los pagos.
- Actualización del valor total de los pagos.
- Registro e impresión de la factura.

loop items: representa el ciclo donde se vinculan los productos y/o servicios a los ítems y se actualizan sus correspondientes valores, se

vinculan o eliminan los ítems de la factura y se actualizan el valor del IVA y los valores totales de factura.

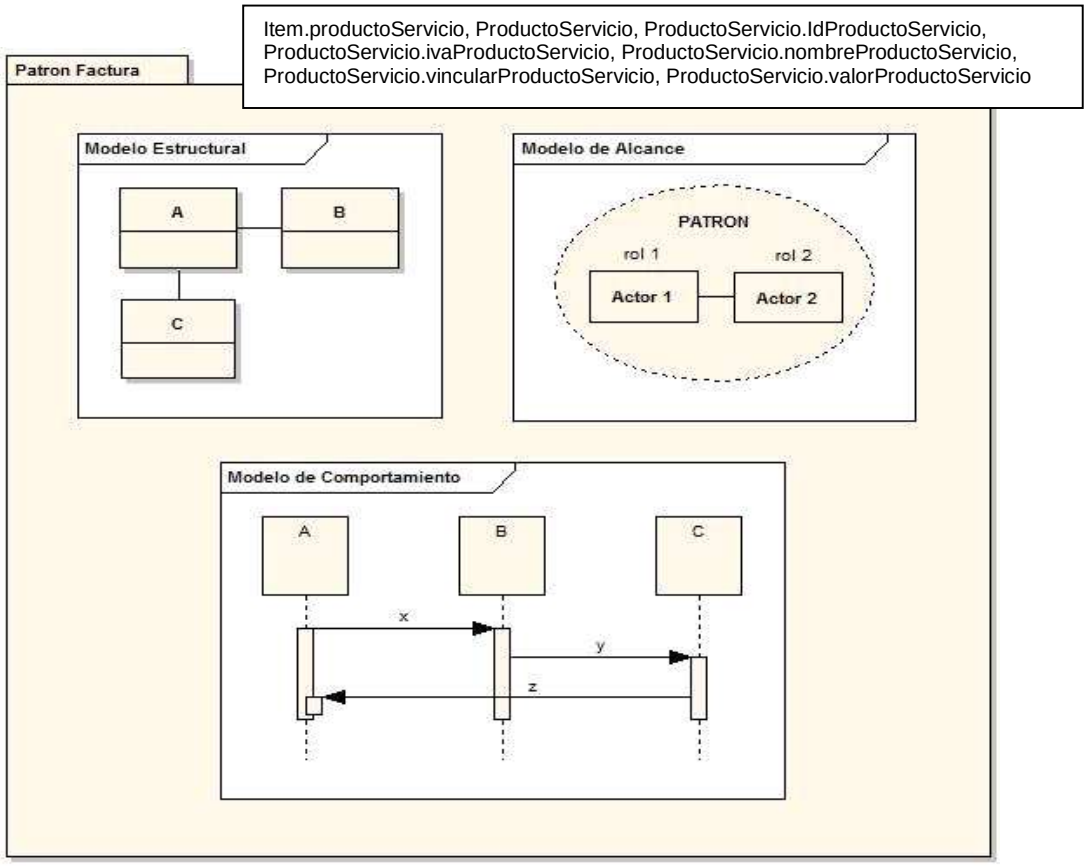
loop pagos: representa el ciclo en el cual se registran los pagos y se actualiza el valor total de pagos en la factura. Este ciclo se ejecuta mientras que el valor total de pagos sea menor que el valor total de la factura.

- **Alternativas**

alt vincular/eliminar: representa el punto de flujo donde se selecciona una opción de operación sobre un ítem de la factura. Las opciones son: eliminar o vincular.

✓ Template de patrón Factura

Figura 33: Template de patrón Factura



- **Catalogar patrón**

Tabla 8: Plantilla de descripción del patrón Factura

Nombre	Factura
Dominios	POS, Comercio
Descripción	El patrón de análisis Factura describe una propuesta de solución para la recolección, procesamiento y almacenamiento de la información pertinente a una operación de venta.
Requisito de dominio	Gestionar factura
Roles participantes	Crear, gestionar y pagar
Usos conocidos	▪ Supermercados y autoservicios - facturación de productos ▪ Clubes - facturación de productos y servicios ▪ Tiendas y negocios de venta minorista - facturación de productos ▪ Gastronomía (restaurantes, bares y negocios de comidas rápidas) - facturación de productos
Patrones relacionados	▪ Contract ▪ Portfolio

6.5. PROCESO DE CONSTRUCCIÓN DEL CATÁLOGO: PATRÓN TRANSACCIÓN

Para la creación del patrón Transaccion se identifican y especifican los requisitos de dominio Validar Forma de Pago y Control de devoluciones, luego se modela el patrón por medio de la creación de la colaboración, diagrama de clases y diagrama de secuencia; finalmente se lo cataloga.

6.5.1. Identificar y especificar requisitos del dominio

Tabla 9: Requisito de dominio Validar forma de pago

FRQ-002	Validar forma de pago
Versión	1.0
Autores	Iván Cisneros, Gloria Erazo, Germán Peña
Fuentes	Supervisor de caja
Descripción	El sistema deberá validar la forma de pago y realizar las operaciones correspondientes, por ejemplo: • Pago en efectivo. • Pago con moneda extranjera. • Pago con bonos. • Pago con tarjeta crédito. • Pago con tarjeta débito. • Pago mixto.
Dominios asociados	POS Comercio
Patrón solución	Transacción
Estado	Pendiente de verificación
Estabilidad	Alta
Comentarios	Ninguno

Tabla 10: Requisito de dominio Control de devoluciones

FRQ-003	Control de devoluciones
Versión	1.0
Autores	Iván Cisneros, Gloria Erazo, Germán Peña
Fuentes	Supervisor de caja
Descripción	El sistema deberá realizar el control de posibles devoluciones: 1. Devoluciones de 1 o más productos: Si la devolución del (los) producto(s) se realiza después de ser identificado y enlistado en la factura temporal, se procederá a borrarlo(s) de la lista y a actualizar el total de la factura. Si la devolución del (los) producto(s) se realiza después del pago del total de la factura entonces se borra el producto de la lista, se actualiza el total de la factura y se reversa la transacción de pago por el valor correspondiente al (los) producto(s) devuelto(s) de acuerdo a las políticas de la organización. Si la devolución del (los) producto(s) se realiza después de ejecutar la venta, el sistema envía la información a los sistemas involucrados en la operación inicial para reversar los registros generados correspondientes al (los) producto(s) devuelto(s). 2. Cancelación de la venta: Si la cancelación de la venta se realiza antes de efectuar el pago, el sistema deberá eliminar la factura temporal. Si la cancelación de la venta se realiza después de efectuar el pago o ejecutar la venta, el sistema envía la información a los sistemas involucrados en la operación inicial para reversar los registros generados correspondientes a la venta y se reversa la transacción de pago por el valor correspondiente a la venta de acuerdo a las políticas de la organización.
Dominios asociados	POS Comercio
Estado	Pendiente de verificación
Estabilidad	Media
Comentarios	Ninguno

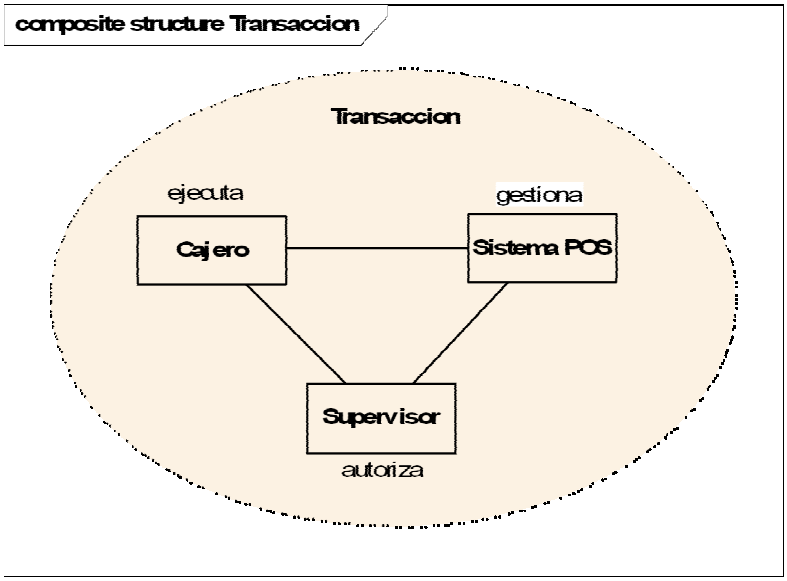
6.5.2. Definir el patrón

- **Modelar patrón**
  - ✓ **Modelar alcance**

Tabla 11: Asignación de responsabilidades patrón Transaccion

RESPONSABILIDAD	CLASE RESPONSABLE	ACTOR RESPONSABLE
Generar entrada	Transacción	Cajero/Supervisor
Acumular entrada	Entrada	Sistema POS
Validar forma de transacción	Transacción	Cajero
Actualizar nivel	Depósito	Sistema POS

Figura 34: Colaboración patrón Transaccion



- **Actores**

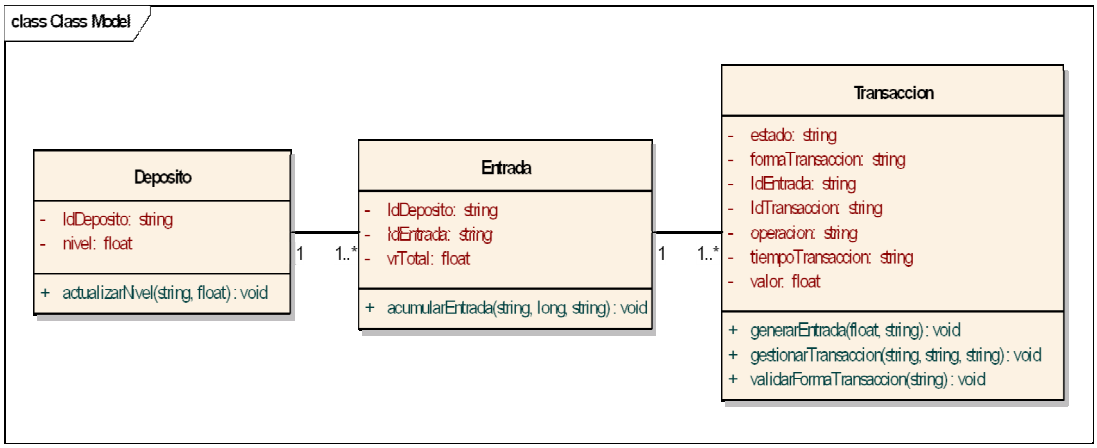
  - **Cajero:** representa el funcionario que opera la caja donde se realiza la transacción.
  - **Sistema POS:** representa el software que gestiona los procesos que implican una transacción.
  - **Supervisor:** representa el funcionario encargado de controlar y permitir los procesos de alto nivel requerido en un punto de venta.

- **Roles**

  - **Autorizar:** representa la función de conceder autoridad, facultad o derecho para realizar una acción que requiera de un alto compromiso para la empresa durante una transacción.
  - **Ejecutar:** representa la función de llevar a cabo una transacción.
  - **Gestionar:** representa la función de recibir, procesar, validar, almacenar la información referente a la operación de transacción.

✓ **Modelar estructura**

Figura 35: Diagrama de clases del patrón Transaccion



- **Clase Transaccion:** describe conceptualmente la acción que genera la entrada que será acumulada para afectar un depósito.

  Atributos

  - **estado:** representa un estado que determina que acciones se deben llevar a cabo durante la transacción
  - **formaTransaccion:** representa la manera en la que se realiza el proceso que genera la entrada, por ejemplo: si la transacción es un pago éste puede ser en efectivo, tarjeta debito o crédito, bonos, etc.
  - **IdEntrada:** representa el código de identificación de cada entrada generada por una transacción.
  - **IdTransaccion:** representa el código de identificación de la transacción.
  - **operación:** representa la forma en que la entrada afectara al depósito. Si será positiva o negativamente.
  - **tiempoTransaccion:** representa el periodo de tiempo durante el cual es realizada la transacción.
  - **valor:** representa la cantidad de elementos que se generan en una transacción. El tipo de elementos dependerá del tipo de transacción, por ejemplo: si se habla de una transacción económica dicha cantidad será de dinero, si se refiere a un deposito de una bodega la cantidad será de ítems.

  Métodos

  - **generarEntrada:** representa la operación que permite crear la entrada con su respectivo valor y operación a realizar.
  - **gestionarTransaccion:** representa la operación que permite llevar un registro de las transacciones realizadas con un identificador, el tiempo de transacción y el estado de la misma.
  - **validarFormaTransaccion:** representa la operación que permite al sistema reconocer el tipo de transacción a realizar.

- **Clase Entrada:** describe conceptualmente el punto donde se acumularan todas las posibles entradas que afecten a un depósito.

  Atributos

  - **IdDeposito:** representa el código de identificación del depósito que será afectado por las entradas.
  - **IdEntrada:** representa el código de identificación de cada entrada generada por una transacción.

- **vrTotal:** representa la cantidad de elementos que afectaran al depósito.

Métodos

- **acumularEntrada:** representa la operación que permite aumentar o disminuir el valor total de la entrada que afectará finalmente al depósito.

- **Clase Deposito:** describe conceptualmente el lugar donde se almacenan las entradas.

Atributos

- **IdDeposito:** representa el código de identificación del depósito que será afectado por las entradas.
- **nivel:** representa la cantidad de elementos existentes en el depósito.

Métodos

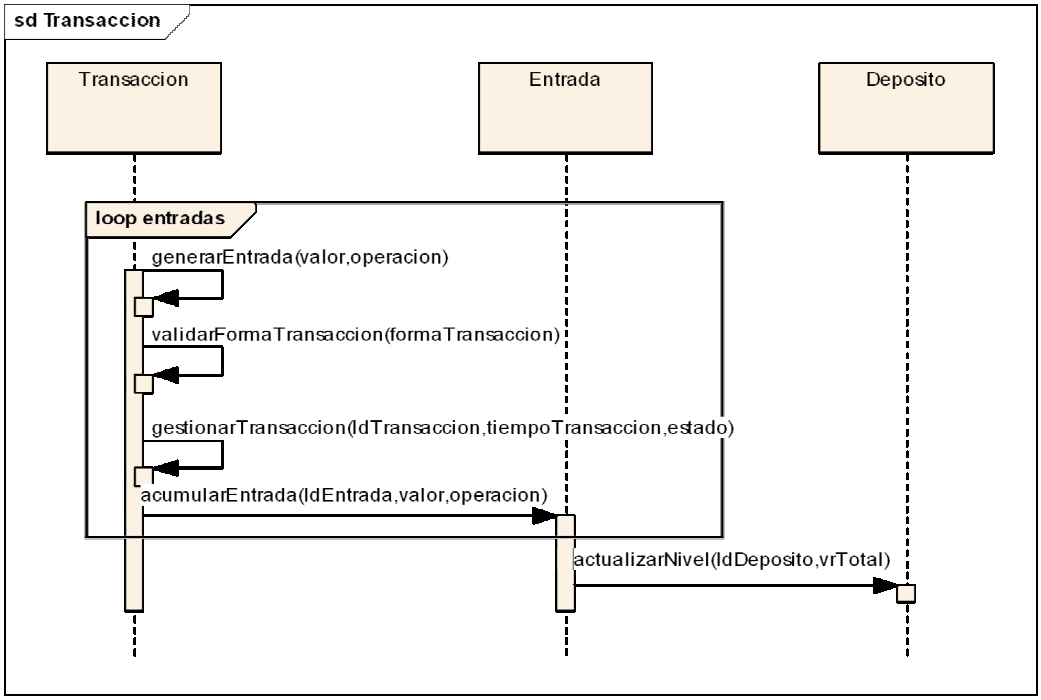
- **actualizarNivel:** representa la operación que permite aumentar o disminuir el nivel del depósito.

- **Cardinalidad**

- Una transacción puede generar una entrada, una entrada puede ser generada por una o más transacciones.
- Una entrada puede afectar a un solo depósito, un depósito puede ser afectado por una o más entradas.

- **Modelar comportamiento**

Figura 36: Diagrama de secuencia del patrón Transaccion



- **Objetos**

  - **Transaccion:** representa el proceso concreto que genera la entrada que será acumulada para afectar un depósito.
  - **Entrada:** representa el punto concreto donde se acumularan todas las posibles entradas que afecten a un depósito.
  - **Deposito:** representa el lugar concreto donde se almacenan las entradas.

- **Mensajes**

  - **generarEntrada:** representa el proceso de crear la entrada con su respectivo valor y operación a realizar. Este proceso se ejecuta una vez por cada transacción.
  - **validarFormaTransaccion:** representa el proceso de reconocer el tipo de transacción a realizar. Este proceso se ejecuta una vez por cada transacción.

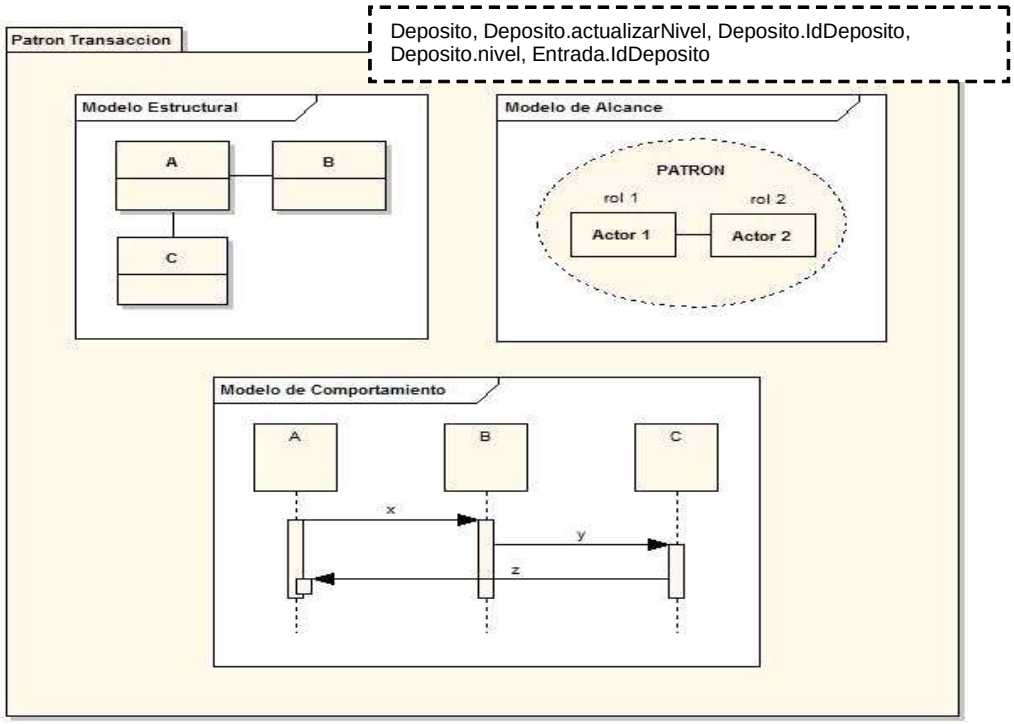
- **gestionarTransaccion:** representa el proceso de llevar un registro de las transacciones realizadas con un identificador, el tiempo de transacción y el estado de la misma. Este proceso se ejecuta una vez por cada transacción.
- **acumularEntrada:** representa el proceso de aumentar o disminuir el valor total de la entrada que afectará finalmente al depósito. Este proceso se ejecuta una vez por cada transacción.
- **actualizarNivel:** representa el proceso de aumentar o disminuir el nivel del depósito. Este proceso se ejecuta una vez ya estén acumuladas las entradas.

o **Ciclos**

- **loop entradas:** representa el ciclo donde se realizan las transacciones que generan una entrada, se validan y registran las transacciones y se acumulan las entradas que afectaran finalmente al depósito.

• **Template del patrón Transaccion**

Figura 37: Template del Patrón Transacción



- **Catalogar patrón**

Tabla 12: Plantilla de descripción del patrón Transaccion

Nombre	Transaccion
Dominios	POS, Comercio
Descripción	El patrón de análisis Transaccion describe una propuesta de solución para mantener un registro de cambios positivos o negativos en un depósito, realizados a través de unas entradas. Estas entradas son generadas a través de una o más transacciones.
Requisito de dominio	Validar Forma de Pago, Control de devoluciones
Roles participantes	Ejecutar, gestionar y autorizar
Usos conocidos	▪ Sistemas bancarios – manejo de cuentas de clientes ▪ Inventarios y depósitos – gestión de entradas y salidas ▪ Aerolíneas – manejo de cuentas de clientes
Patrones relacionados	▪ Account ▪ Summary account ▪ Memo account ▪ Posting rules ▪ Sources of an entry

6.6. PROCESO DE CONSTRUCCIÓN DEL CATÁLOGO: PATRÓN ACCION

Para la creación del patrón Accion se identifica y especifica el requisito de dominio Gestionar Acciones, luego se modela el patrón por medio de la creación de la colaboración, diagrama de clases y diagrama de secuencia; finalmente se lo cataloga.

6.6.1. Identificar y especificar requisitos del dominio

Tabla 13: Descripción de requisito de dominio Gestionar acciones

FRQ-004	Gestionar acciones
Versión	1.0
Autores	Iván Cisneros, Gloria Erazo, Germán Peña
Fuentes	Supervisor de caja
Descripción	El sistema POS deberá permitir programar e implementar acciones, las cuales podrán ser completadas, canceladas o suspendidas por un periodo de tiempo.
Dominios asociados	POS Comercio
Patrón solución	Accion
Estado	Pendiente de verificación
Estabilidad	Media
Comentarios	Ninguno

6.6.2. Definir patrón

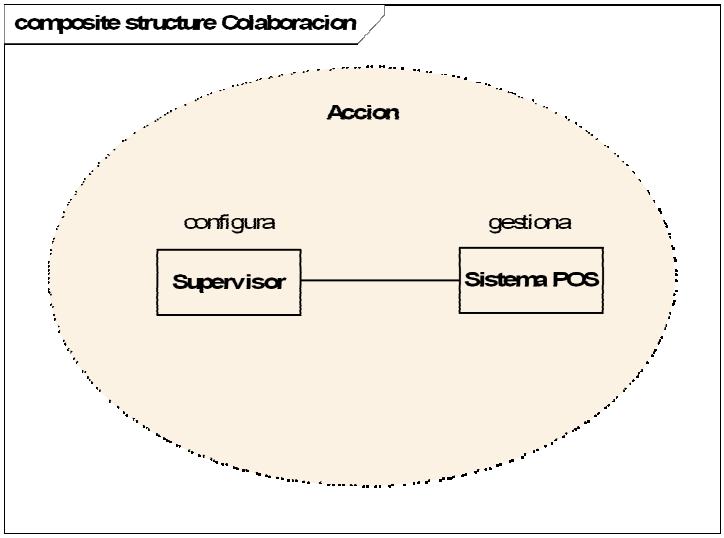
- **Modelar patrón**

  ✓ **Modelar alcance**

Tabla 14: Asignación de responsabilidades patrón Accion

RESPONSABILIDAD	CLASE RESPONSABLE	ACTOR RESPONSABLE
Programar acción	**AccionPropuesta**	**Sistema POS/ Supervisor**
Iniciar acción	**AccionImplementada**	**Sistema POS/ Supervisor**
Completar acción	**AccionCompletada**	**Sistema POS/ Supervisor**
Cancelar acción	**AccionAbandonada**	**Sistema POS/ Supervisor**
Posponer acción	**Suspension**	**Sistema POS/ Supervisor**
Retomar acción	**Suspension**	**Sistema POS/ Supervisor**

Figura 38: Colaboración patrón Accion



- **Actores**

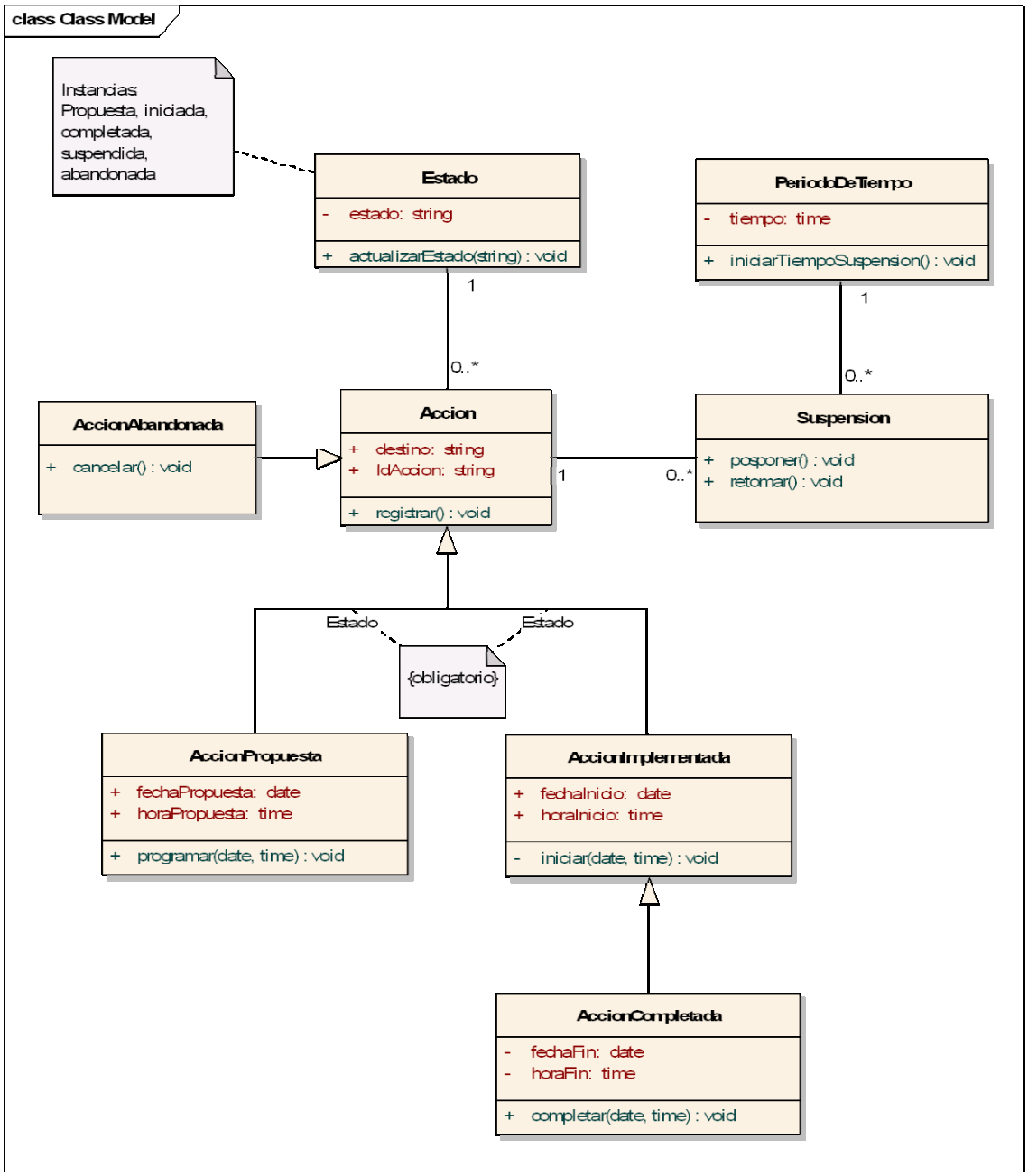
  - **Supervisor:** representa el funcionario encargado de la configuración de las acciones que se llevan a cabo en el dominio POS.
  - **Sistema POS:** representa el software que gestiona los procedimientos para llevar a cabo acciones dentro del dominio POS.

- **Roles**

  - **Configurar:** representa la función de programar e iniciar la implementación de acciones, las cuales podrán ser completadas, canceladas o suspendidas.
  - **Gestionar:** representa la función de recibir, procesar, administrar y almacenar la información referente a las actividades que se llevan a cabo en el dominio POS.

✓ **Modelar estructura**

Figura 39: Diagrama de clases del patrón Accion



- **Clase Accion:** describe conceptualmente las actividades que se llevan a cabo en el dominio.

  Atributos

  - **destino:** representa el lugar donde se va a llevar a cabo la acción.
  - **IdAccion:** representa el código de identificación de la acción.

  Métodos

  - **registrar:** representa la operación que permite almacenar la información de la acción.

- **Clase AccionPropuesta:** describe conceptualmente las actividades que se programan dentro del dominio.

  Atributos

  - **fechaPropuesta:** representa la fecha para la cual se programa la acción.
  - **horaPropuesta:** representa la hora para la cual se programa la acción.

  Métodos:

  - **programar:** representa la operación que permite programar la acción.

- **Clase AccionImplementada:** describe conceptualmente las actividades que se realizan dentro del dominio.

  Atributos

  - **fechaInicio:** representa la fecha en que se inicia la realización de la acción.
  - **horaInicio:** representa la hora en que se inicia la realización de la acción.

  Métodos

  - **iniciar:** representa la operación que permite comenzar la implementación de la acción.

- **Clase AccionCompletada:** describe conceptualmente las actividades que se han terminado de realizar.

  Atributos

  - **fechaFin:** representa la fecha en que se completó la acción.
  - **horaFin:** representa la hora en que se completó la acción.

  Métodos

  - **completar:** representa la operación que permite finalizar la acción. Completar

- **Clase AccionAbandonada:** describe conceptualmente las actividades que se cancelan completa y definitivamente.

  Métodos

  - **cancelar:** representa la operación que permite anular por completo y definitivamente la acción.

- **Clase Estado:** describe conceptualmente las etapas en la que se encuentran las acciones.

  Atributos

  - **estado:** representa la etapa en la que se encuentra la acción. Por ejemplo: propuesta, iniciada, completada, suspendida o abandonada.

  Métodos

  - **actualizarEstado:** representa la operación que permite cambiar del estado anterior al estado actual de la acción.

- **Clase Suspension:** describe conceptualmente la cesación temporal de las acciones.

  Métodos:

  - **posponer:** representa la operación que permite detener de manera temporal la acción.
  - **retomar:** representa la operación que permite continuar la acción que fue suspendida.

- **Clase PeriodoDeTiempo:** describe conceptualmente el intervalo de tiempo durante el cual se suspende una acción.

  Atributos

  - **tiempo:** representa la duración de la suspensión.

  Métodos

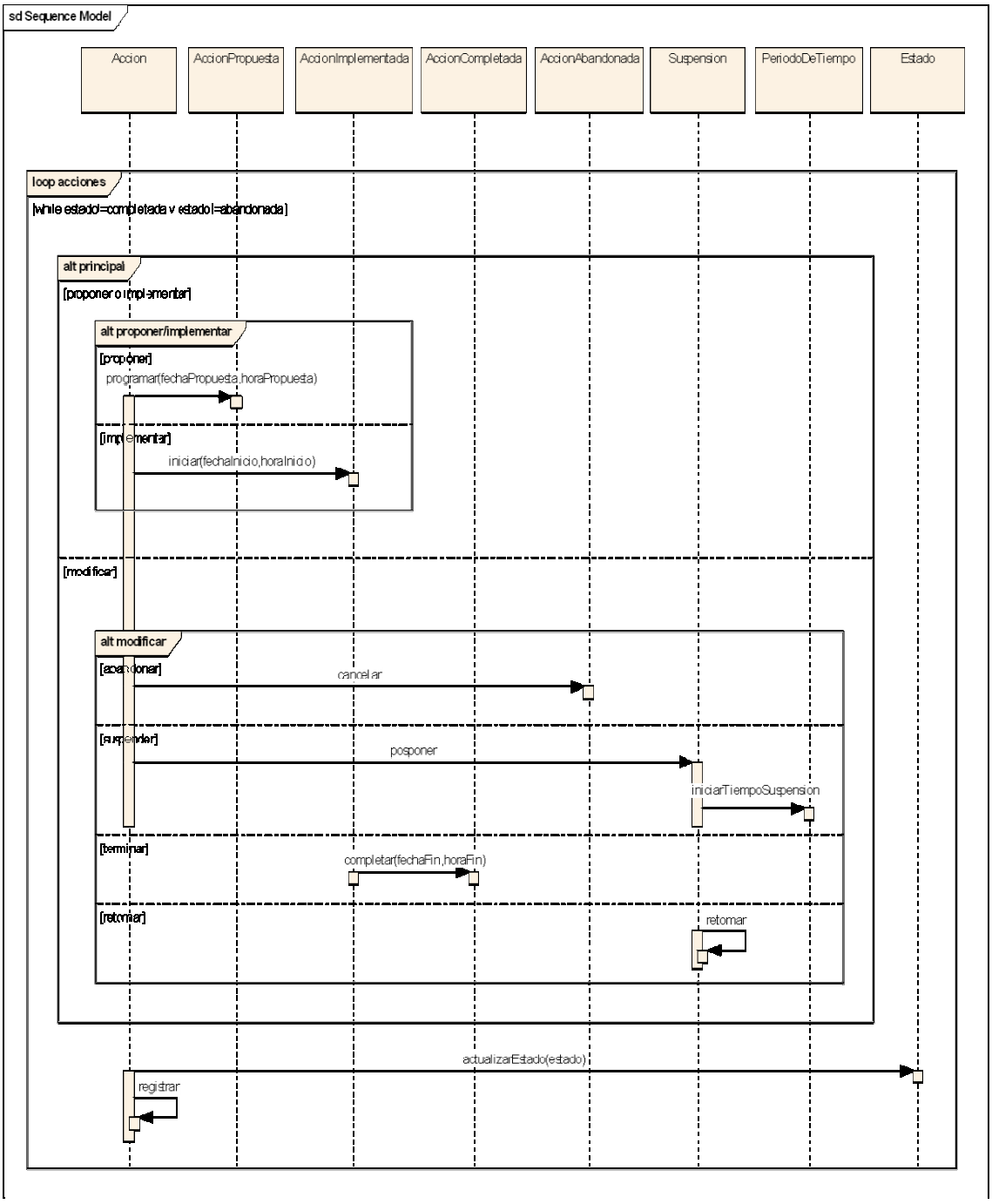
  - **iniciarTiempoSuspension:** representa la operación que permite comenzar el tiempo de suspensión de la acción.

- **Cardinalidad**

  - Una acción puede ser suspendida cero o más veces, cero o más suspensiones pertenecen a una acción.
  - Cero o más acciones deben tener un estado, un estado hace parte de cero o más acciones.
  - Cero o más suspensiones deben tener un periodo de tiempo, un periodo de tiempo pertenece a cero o más suspensiones.

- ✓ **Modelar comportamiento**

Figura 40: Diagrama de secuencia patrón Accion



- **Objetos**

  - **Accion:** representa la actividad concreta que se lleva a cabo en el dominio.
  - **AccionPropuesta:** representa la actividad concreta que se programa dentro del dominio.
  - **AccionImplementada:** representa la actividad concreta que se realiza dentro del dominio.
  - **AccionCompletada:** representa la actividad concreta que se ha terminado de realizar.
  - **AccionAbandonada:** representa la actividad concreta que se cancela completa y definitivamente.
  - **Suspension:** representa el cese temporal concreto de la acción.
  - **PeriodoDeTiempo:** representa el intervalo de tiempo concreto durante el cual se suspende la acción.
  - **Estado:** representa la etapa concreta en la que se encuentra la acción.

- **Mensajes**

  - **programar:** representa el proceso de programación de la acción.
  - **iniciar:** representa el proceso de implementación de la acción. Este proceso puede ejecutarse habiendo o no programado la acción.
  - **cancelar:** representa el proceso de anulación completa y definitiva de la acción. Este proceso puede ejecutarse si la acción ha sido programada o implementada anteriormente.
  - **posponer:** representa el proceso de detención temporal de la acción. Este proceso puede ejecutarse si la acción ha sido programada o implementada anteriormente.
  - **iniciarTiempoSuspension:** representa el proceso de inicialización del tiempo de suspensión de la acción. Este proceso se ejecuta una vez se suspenda la acción.
  - **completar:** representa el proceso de finalización de la acción. Este proceso puede ejecutarse si la acción se ha implementado anteriormente.
  - **retomar:** representa el proceso de continuación de la acción que fue suspendida.
  - **actualizarEstado:** representa el proceso de cambio de estado de la acción. Este proceso se ejecuta cada vez que la acción cambie de estado.
  - **registrar:** representa el proceso de almacenamiento de la información de la acción.

- **Ciclos**

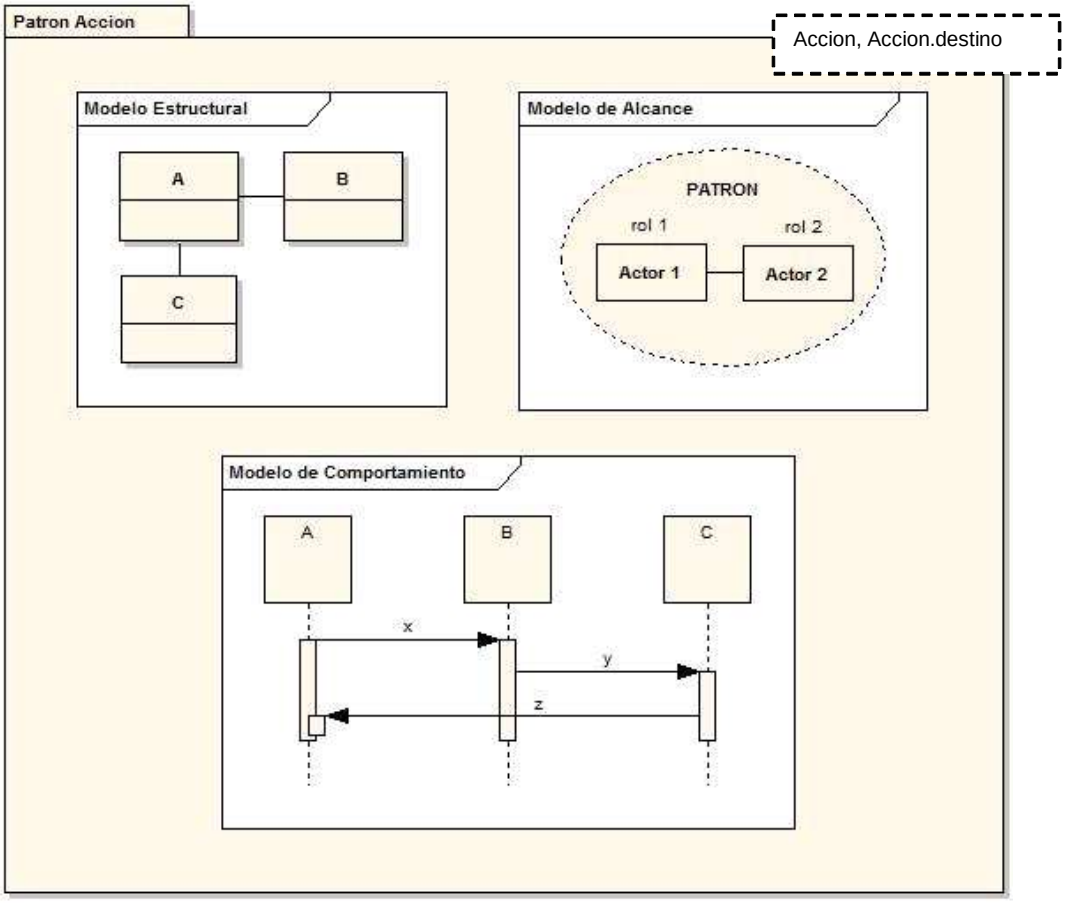
  - **loop acciones:** representa el ciclo en el cual se realizan todos los procedimientos de programación, implementación, finalización, cancelación y suspensión de la acción. Este ciclo se ejecuta mientras que el estado de la acción sea diferente de completado o abandonado.

- **Alternativas**

  - **alt principal:** representa el punto de flujo donde se selecciona una opción de operación sobre una acción. Las opciones son: proponer o implementar, ó modificar la acción que se propuso o implementó con anterioridad.
  - **alt proponer/implementar:** representa el punto de flujo donde se selecciona una opción de operación sobre una acción. Las opciones son: proponer o implementar.
  - **alt modificar:** representa el punto de flujo donde se selecciona una opción de operación sobre una acción ya propuesta o implementada. Las opciones son: abandonar, suspender, terminar o retomar.

- **Template del patrón Accion**

Figura 41: Template del patrón Accion



- **Catalogar patrón**

Tabla 15: Plantilla de descripción del patrón Accion

Nombre	Accion
Dominios	POS, Comercio
Descripción	El patrón de análisis Accion describe una propuesta de solución para el registro de acciones llevadas a cabo dentro del dominio. Los posibles estados de una acción se dividen en dos subtipos principales: acción propuesta y acción implementada, que representan la intención y lo que realmente sucede. El final de una acción está igualmente dividido en acción completada y acción abandonada. Una acción abandonada representa una cancelación definitiva de la acción, y posponer temporalmente la acción representa una suspensión.
Requisito de dominio	Gestionar acciones
Roles participantes	Configurar y gestionar
Usos conocidos	▪ Supermercados – programación de envío de información a otros sistemas ▪ Aerolíneas – programación de vuelos ▪ Restaurantes – reservaciones
Patrones relacionados	▪ Plan ▪ Protocol ▪ Resource allocation

6.7. RESULTADOS DE VALIDACIÓN EN ETAPA 1: CONSTRUCCIÓN DEL CATÁLOGO

En la siguiente tabla, se muestra el resumen de validación por cada uno de los elementos de la construcción del catálogo, siguiendo la estrategia de validación planteada anteriormente.

Tabla 16: Resultados de validación en etapa 1

Tipo	Elemento	Objetivo de elemento	Observaciones
Proceso - Gestión del catálogo de patrones de análisis	Identificar y especificar requisitos de dominio	Definir los requisitos que especifican el dominio y que son insumo para la definición del patrón.	Para la especificación de los requisitos de dominio no existe una plantilla definida, por esta razón se propone una para el desarrollo del objetivo. Ésta se describe detalladamente en el plan de mejoramiento, capitulo 7.
	Modelar patrón	Generar los modelos que definen al patrón	Al seguir los pasos de este proceso, es posible generar desde tres vistas diferentes los modelos que definen el patrón. Como resultado, se obtienen unas representaciones claras y adecuadas del patrón de análisis.
	Catalogar patrón	Integrar el nuevo patrón al catálogo a partir de la instanciación de los elementos de descripción del patrón	Siguiendo el proceso de ESKEMA, se observa que no existe un punto en donde se realiza la descripción del patrón; se propone que ésta se realice durante la etapa de catalogación debido a que es necesaria para la integración del nuevo patrón al catálogo.

Representación	Modelo estructural	Representar cómo se compone y relaciona la estructura del patrón de análisis.	Este elemento cumple con su objetivo ya que permite representar, de una manera clara y adecuada, los elementos que componen un patrón de análisis y su relación estructural.
		Representar la estructura conceptual del patrón de análisis.	Este objetivo es cumplido ya que al crear un diagrama de clases, es posible representar la estructura del patrón de análisis a través de la identificación de sus clases, atributos y relaciones.
	Modelo de comportamiento	Definir la parte dinámica del patrón de análisis.	Su objetivo es cumplido ya que permite describir, de una manera apropiada y comprensible, los aspectos del patrón de análisis que cambian con el tiempo.
		Describir la interacción de los objetos del patrón de análisis a través del tiempo.	Este objetivo es cumplido ya que al representar el comportamiento del patrón por medio de un diagrama de secuencia, es posible mostrar la interacción de sus componentes a través del tiempo, haciendo énfasis en el momento y orden correcto en que se realizan.

Representación	Modelo de alcance	Definir cómo el patrón de análisis interactúa con su contexto y en especial con los usuarios de acuerdo con los roles que asumen.	Este elemento cumple con su objetivo ya que permite determinar la relación del patrón de análisis con su contexto, más específicamente, cómo los usuarios se relacionan entre sí y con el patrón, a través de los roles que desempeñan.
		Representar de manera adecuada, cómo los roles asociados al patrón de análisis se relacionan entre sí.	Este objetivo es cumplido ya que al representar el alcance del patrón de análisis por medio de una colaboración, es posible definir la forma en cómo los roles y actores participantes se relacionan entre sí. Para facilitar la creación de la colaboración, se observó la necesidad de utilizar una tabla de asignación de responsabilidades, mostrada en el capítulo 7.
	Template	Servir de contenedor de los diagramas que representan los modelos que definen el patrón de análisis.	Este elemento cumple con su objetivo ya que permite encapsular el modelo estructural, de comportamiento y de alcance de cada patrón.

6.8. PROCESO DE SIMULACIÓN DE USO DEL CATÁLOGO: PATRÓN ACCION

Para la simulación de uso del catálogo, se toma un requisito específico del supermercado hipotético LINAR y se analizan los patrones para identificar cuál de ellos presenta una propuesta de solución para este requisito.

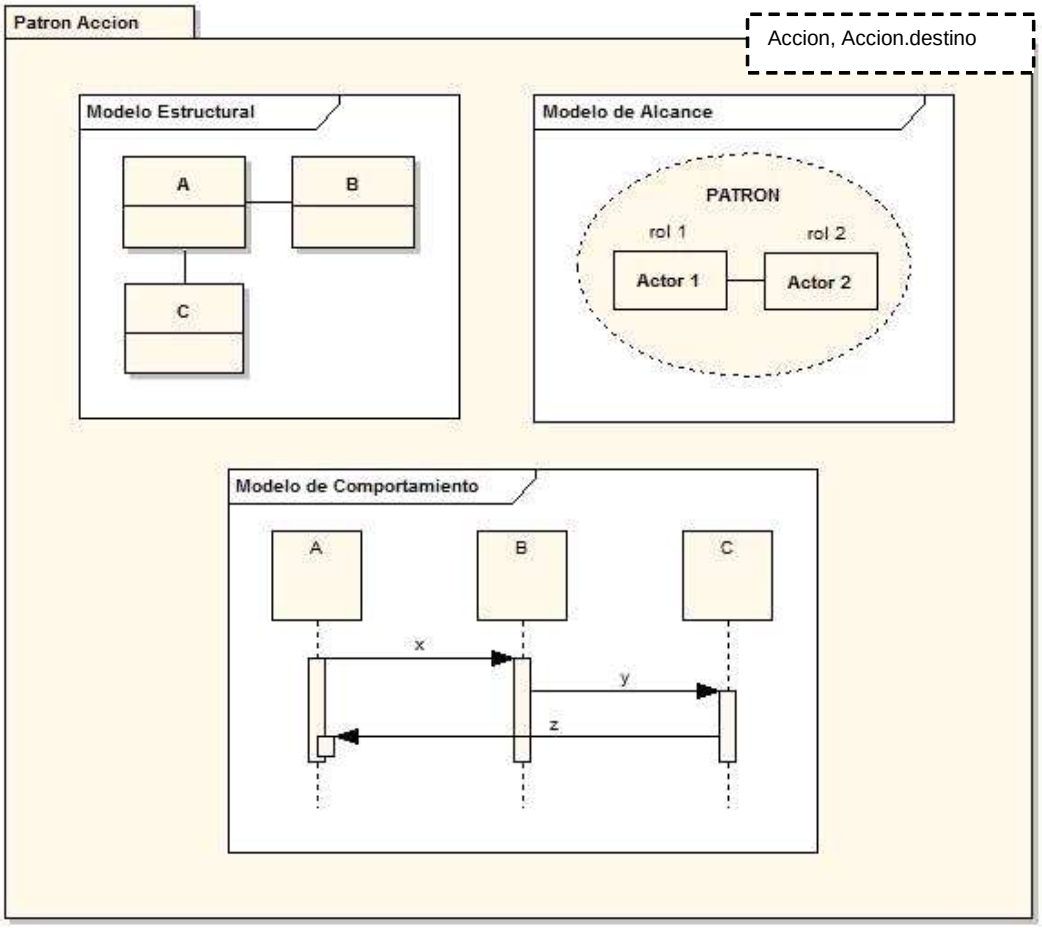
6.8.1. Requisito de aplicación

Tabla 17: Requisito especifico Gestionar Seguridad de Información

NFR-001	Gestionar Seguridad de Información
Versión	1.0
Autores	Iván Cisneros, Gloria Erazo, Germán Peña
Fuentes	Supervisor de Caja del supermercado LINAR
Descripción	El sistema creará un backup interno en el transcurso de dos horas, esta copia de seguridad contendrá todas las transacciones realizadas en el POS en ese espacio de tiempo para cualquier falla física o lógica. El sistema creará un backup externo en medio magnético el cual se conservará en un lugar distinto a la empresa, éste contendrá las transacciones del POS de una semana.
Dominios asociados	POS Comercio
Importancia	Vital
Urgencia	Inmediatamente
Estado	Pendiente de verificación
Estabilidad	Media
Comentarios	Ninguno

Una vez realizado el análisis de los patrones, se determina que el catálogo provee una solución genérica a este requisito a través del patrón Accion, mostrado en la figura 42, que podrá ser recuperado e instanciado para el problema particular del supermercado LINAR.

Figura 42: Template del patrón Accion

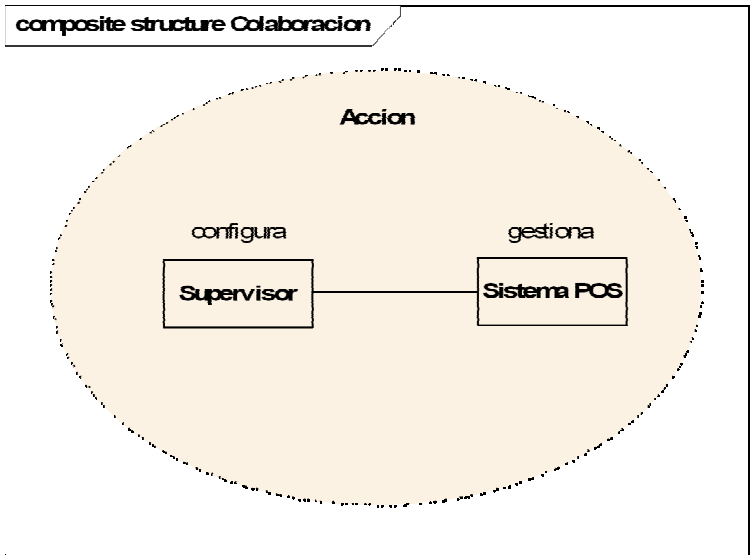


6.8.2. Instanciar el patrón. En este punto de la investigación se da importancia a la parte aspecutal del catálogo debido a que por medio de las variables del signature del template, el patrón es instanciado para un problema específico, en este caso, para cumplir el requisito “Gestionar Seguridad de Información” en el supermercado LINAR.

La instanciación se desarrolla en dos tiempos. Durante el tiempo uno, el patrón se instancia asignando valores a los parámetros del signature del Template. El tiempo uno de instanciación termina cuando los valores asignados a los parámetros del template se integran a los modelos. En este punto, se elimina el elemento signature del template y este se convierte en un paquete que contiene los modelos que ahora son la solución del problema especifico. Los modelos resultantes del tiempo uno pueden ser trabajados para terminar de configurar la solución especifica. Esta última tarea hace referencia al tiempo dos de instanciación.

- **Tiempo uno de instanciación**
  - **Modelo de alcance**

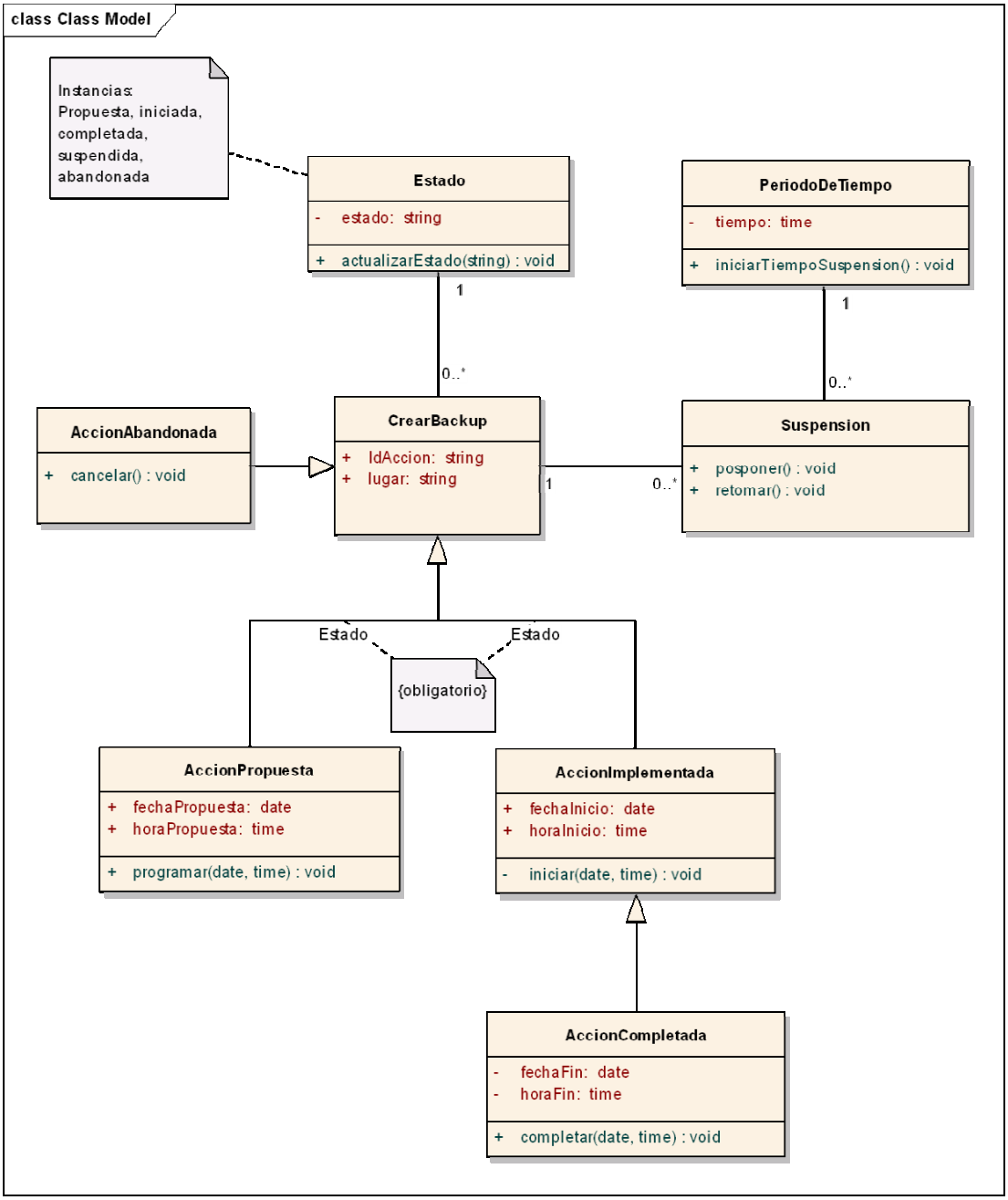
Figura 43: Colaboración patrón Accion en tiempo uno de instanciación



Los cambios realizados al patrón durante el tiempo de instanciación uno, no afectan el modelo de alcance debido a que no se adicionan nuevos actores, ni nuevos roles.

- **Modelo estructural**

Figura 44: Diagrama de clases del patrón Accion en tiempo uno de instanciación

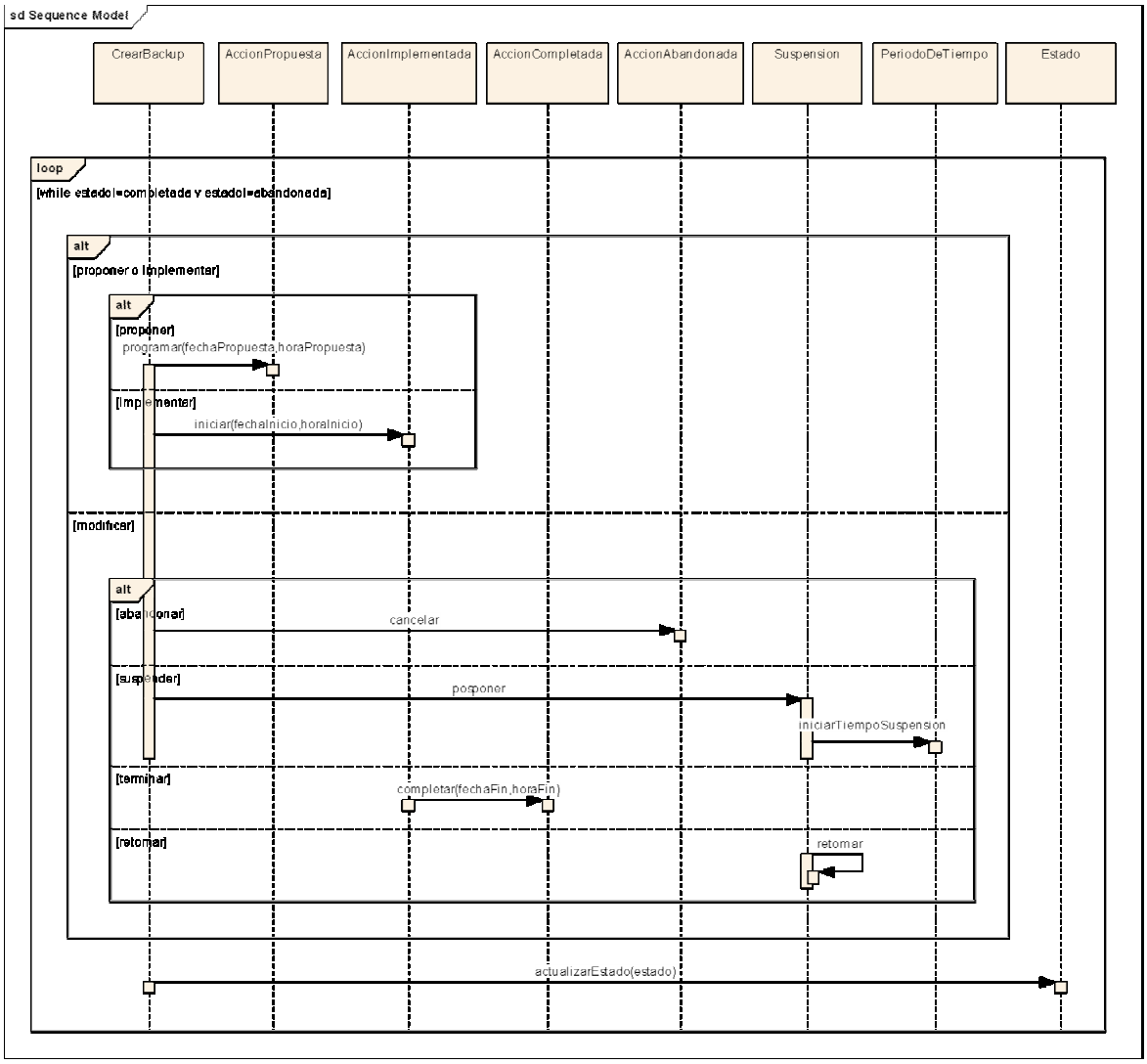


De acuerdo a los parámetros del signature del template, en este modelo se realizaron los siguientes cambios:

- Clase Accion por Clase CrearBackup
- Accion.destino por CrearBackup.lugar

- **Modelo de secuencia**

Figura 45: Diagrama de secuencia del patrón Accion en tiempo uno de instanciación



Debido a los cambios realizados en el modelo estructural del patrón, en este diagrama se modifica el objeto Accion por el objeto CrearBackup.

- **Tiempo dos de instanciación.** El patrón Accion no necesita un tiempo dos de instanciación para este caso, debido a que con los cambios realizados en el tiempo uno ya satisface las exigencias del requisito de aplicación.

6.9. PROCESO DE SIMULACIÓN DE RETROALIMENTACIÓN DEL CATÁLOGO: PATRÓN FACTURA

Los pasos que se siguen para la creación de un patrón de análisis son: definir los requisitos de dominio, modelar el patrón y finalmente catalogarlo. De esta manera, el patrón de análisis queda a disposición del usuario.

Para el caso de la retroalimentación, se desarrolla el proceso inverso: el usuario al afrontar un problema de aplicación encuentra que el catálogo de patrones de análisis no tiene un patrón que lo solucione, el desarrollador se ve en la necesidad de realizar la solución específica. Posteriormente, el requisito de aplicación, los modelos de alcance, el modelo estructural y el modelo de comportamiento que hacen parte de la solución específica, deben generalizarse hacia un requisito de dominio y los modelos que definen una propuesta de solución genérica respectivamente.

Para la simulación de la retroalimentación del catálogo se toma el requisito específico “Gestionar Factura” del supermercado hipotético LINAR, el cual especifica:

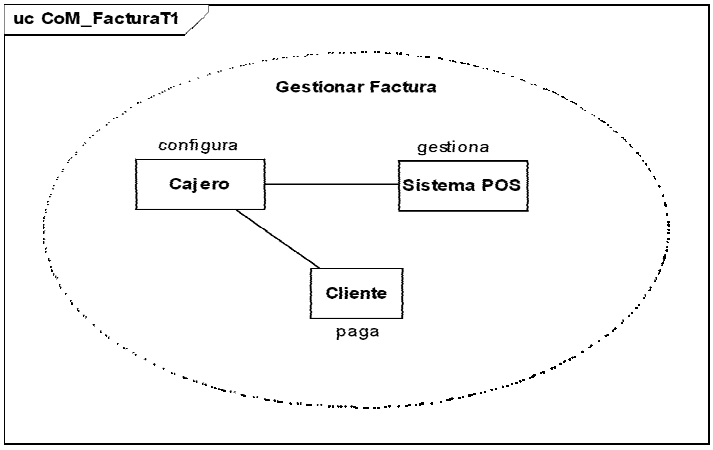
Tabla 18: Requisito especifico Gestionar factura

FRQ-011	Gestionar factura
Versión	1.1
Autores	Gloria Erazo, Iván Cisneros, Germán Peña
Fuentes	Supervisor de caja Supermercado LINAR
Objetivos asociados	Registrar las ventas realizadas en el Supermercado
Requisitos asociados	• FRQ-001 Identificación y registro de productos • FRQ-002 Identificación de formas válidas de pago • FRQ-004 Ejecutar la venta
Descripción	Una vez efectuado el pago, el sistema deberá hacer persistencia e impresión de la factura, la cual contiene la siguiente información: • Cabecera: Compuesta por el código de la factura, el NIT de la empresa LINAR, identificación y nombre del cajero, número de caja, fecha y hora de atención. • Productos: Nombre del producto, cantidad, valor unitario, valor total que corresponde al valor unitario por la cantidad de unidades compradas. • Subtotal: Sumatoria de los valores totales listados en la parte de Productos • IVA: Se discrimina el valor cobrado por concepto IVA • Total: Sumatoria del Subtotal más valor IVA. • Pago: Se indica la forma y valor de pago. • Cliente: Si el cliente ha hecho registro de sus datos personales en el punto de atención al cliente y pertenece al programa de puntos se mostrará el número de identificación y el nombre del cliente y la cantidad de puntos por la compra y puntos acumulados.
Dominios asociados	POS Comercio
Importancia	Vital
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Media
Comentarios	Ninguno

Para el ejercicio se plantea que este requisito no es solventado por ningún patrón del catálogo de patrones aspectuales de análisis, por esta razón, el desarrollador que utiliza el catálogo se ve en la necesidad de crear los tres modelos que plantea MORENA para generalizarlos y evaluarlos, con el fin de determinar si generan un nuevo patrón que retroalimenta el catálogo.

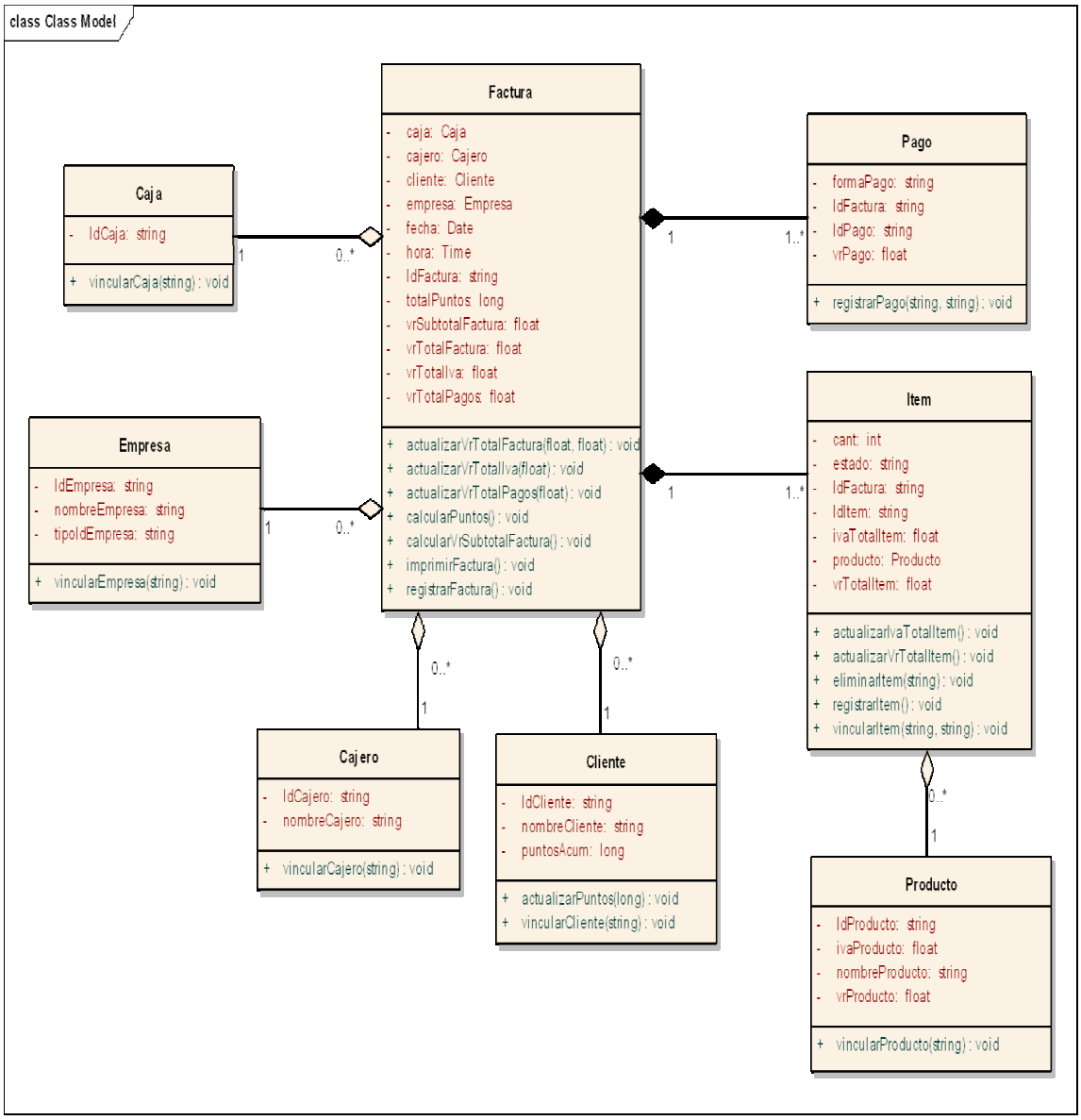
- **Modelo de alcance**

Figura 46: Colaboración realizado para el requisito de la solución específica



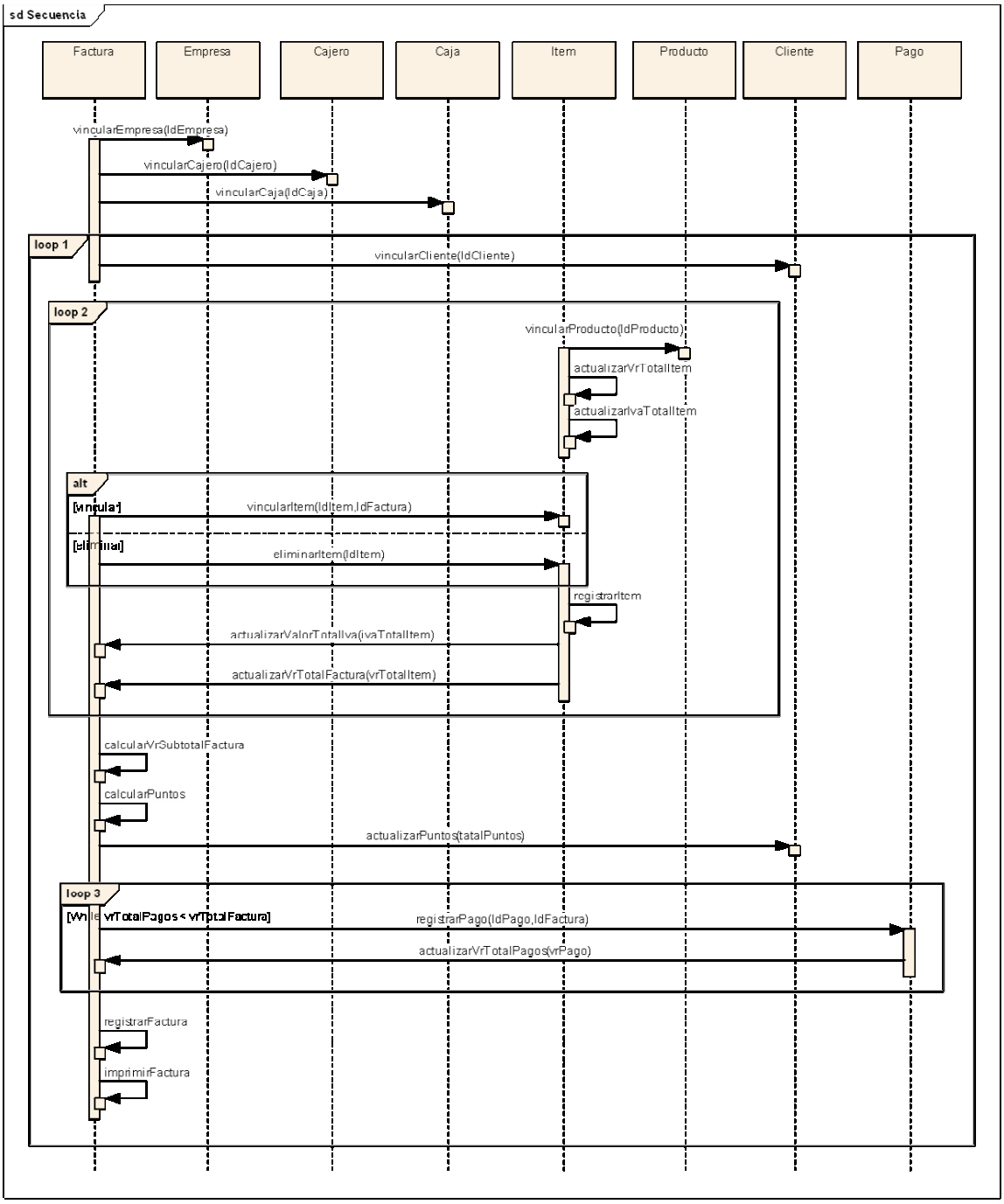
- **Modelo estructural**

Figura 47: Diagrama de clase realizado para el requisito de la solución específica



- **Modelo de comportamiento**

Figura 48: Diagrama de secuencia realizado para el requisito de la solución específica



- **Generalización del requisito del problema específico a requisito de dominio.** Para esto se utiliza la plantilla de requisitos de dominio, por lo cual se eliminan los campos: Objetivos asociados, Requisitos asociados, Importancia y Urgencia, en cambio, se aumenta Patrón solución. En el campo Descripción, se detalla la el requisito lo más genérico posible para que pueda ser aplicado en cualquier tipo de POS, como se muestra en la tabla 19.

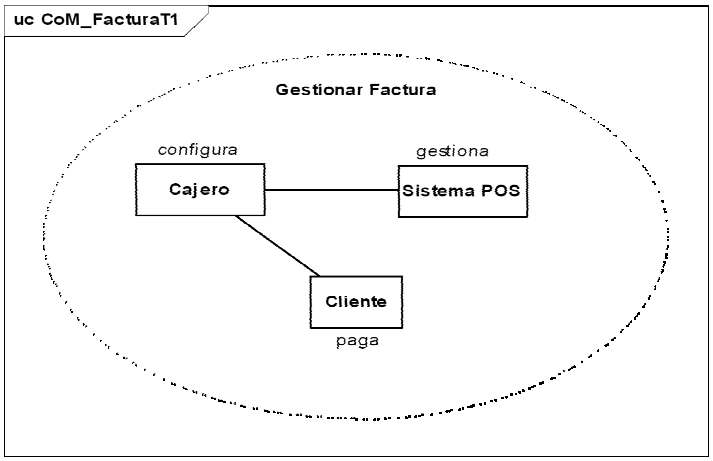
Tabla 19: Descripción del requisito de dominio Gestionar factura

FRQ-011	Gestionar factura
Versión	1.1
Autores	Iván Cisneros, Gloria Erazo, Germán Peña
Fuentes	Supervisor de caja
Descripción	El sistema deberá crear y hacer persistencia de la factura, la cual deberá contener la información que se considere pertinente. Por ejemplo: • Código de la factura, nombre e identificación de la empresa, código y nombre del cajero, número de caja, fecha y hora de atención. • Ítem: nombre del producto o servicio, código, IVA, cantidad, valor unitario, valor total que corresponde al valor unitario por la cantidad de unidades. • Subtotal: diferencia entre el valor total de la factura y el valor total del IVA. • IVA: impuesto sobre el valor agregado de cada producto o servicio. • Total: sumatoria de los valores totales de los ítems • Pago: se indican las formas de pago y sus respectivos valores. • Cliente: se mostrará el número de identificación y el nombre del cliente**.** Al finalizar el proceso el sistema deberá emitir la factura siguiendo los parámetros de ley.
Dominios asociados	POS Comercio
Patrón solución	Factura
Estado	Pendiente de verificación
Estabilidad	Media
Comentarios	Ninguno

- **Generalización de los modelos.** Se modifican los modelos para que sean genéricos y den solución al requisito del dominio.

  ✓ **Modelo de alcance.** No se realiza ningún cambio, la colaboración en este caso es genérica para el patrón.

Figura 49: Colaboración patrón Factura

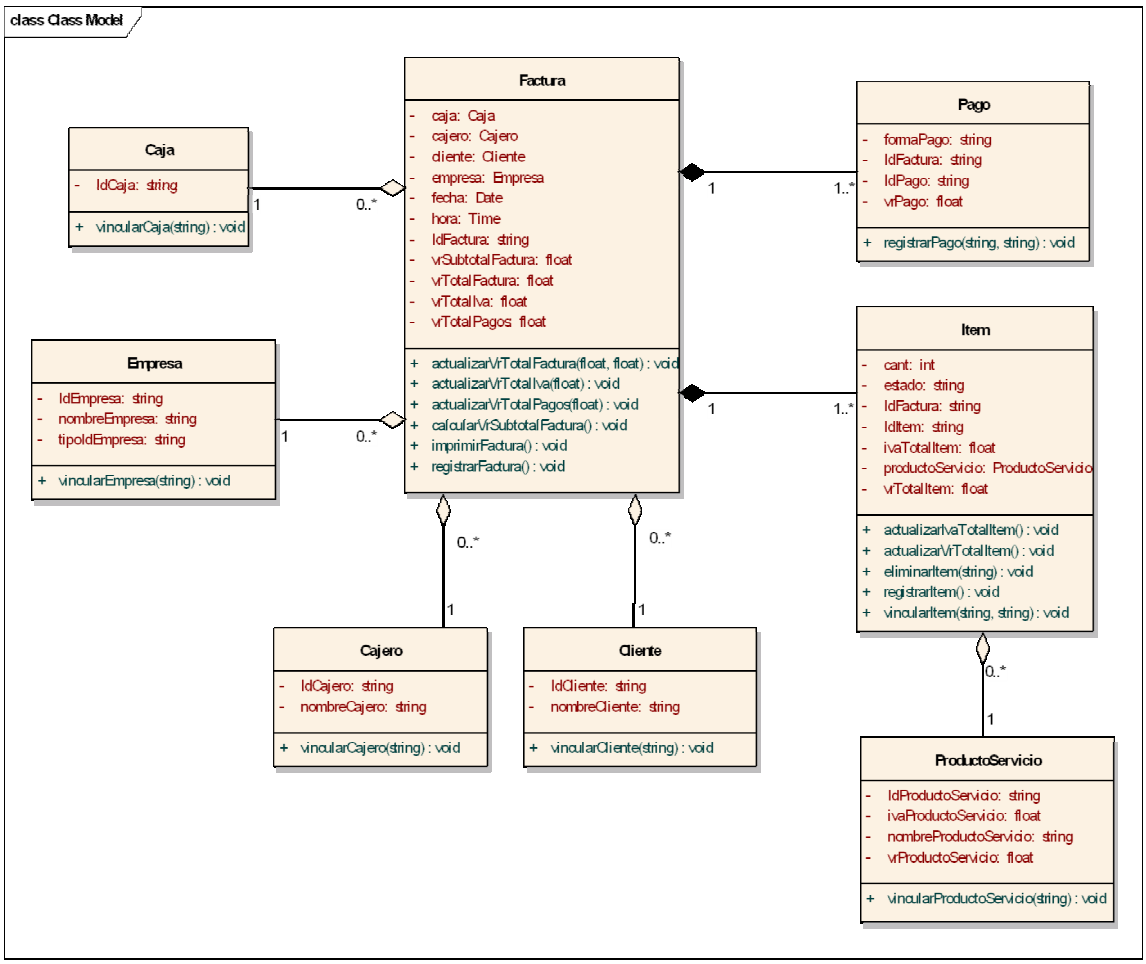


  ✓ **Modelo estructural.** Los cambios que se le realizan al modelo estructural del problema específico son los siguientes:

    - Clase Producto por Clase ProductoServicio
    - Producto.IdProducto por ProductoServicio.IdProductoServicio
    - Producto.ivaProducto por ProductoServicio.ivaProductoServicio
    - Producto.nombreProducto por ProductoServicio.nombreProductoServicio
    - Producto.vrProducto por ProductoServicio.vrProductoServicio
    - Producto.vincularProducto por ProductoServicio.vincularProductoServicio
    - Se elimina el atributo totalPuntos en clase Factura
    - Se elimina el atributo puntosAcum en clase Cliente
    - Se elimina el método calcularPuntos en clase Factura
    - Se elimina el método actualizarPuntos en clase cliente

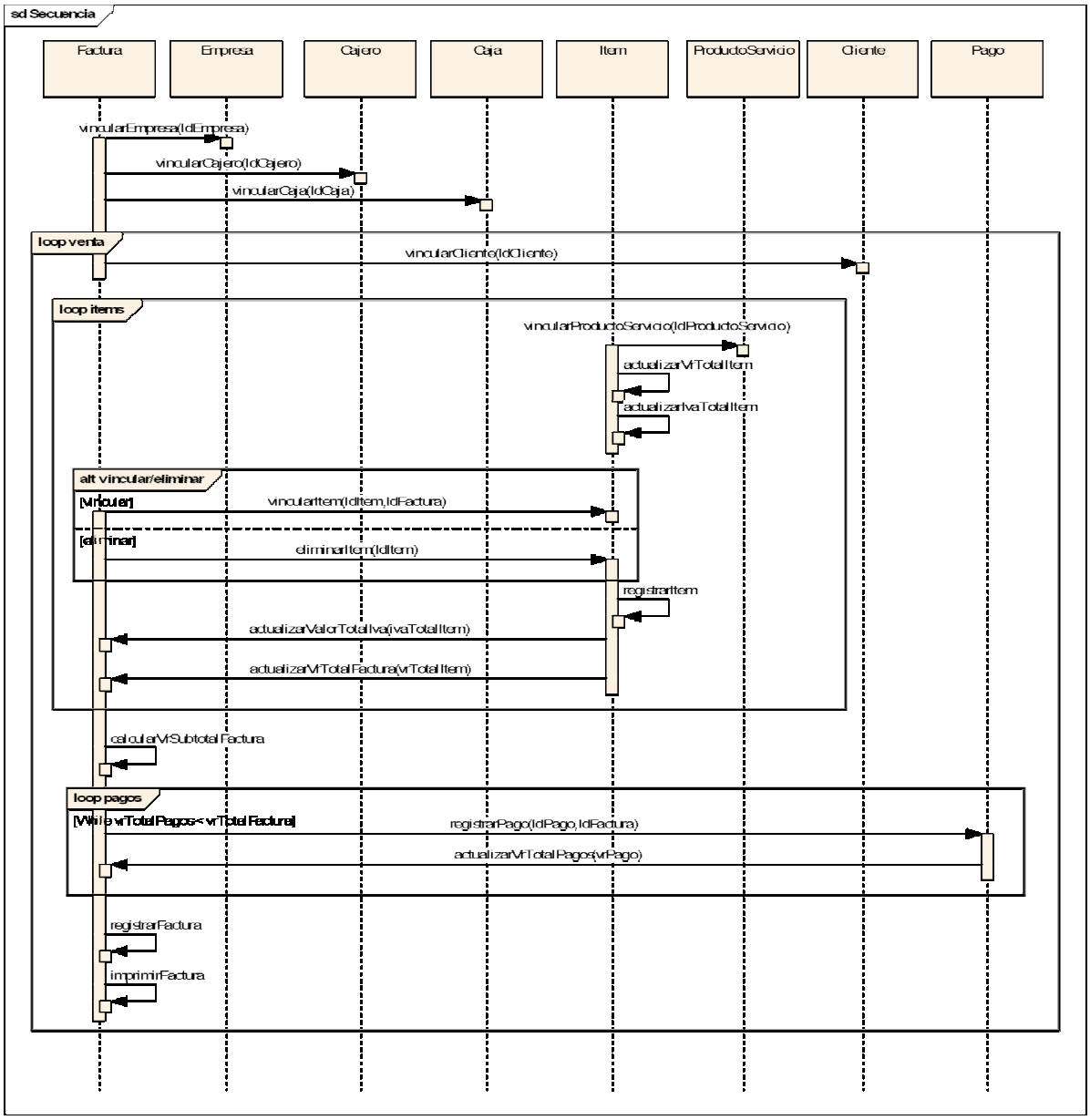
  Estos cambios son realizados porque una empresa u organización, puede ofrecer tanto productos como servicios, y no todos los tipos de POS manejan un programa de puntos por cliente. El nuevo modelo se muestra a continuación:

Figura 50: Diagrama de clases del patrón Factura



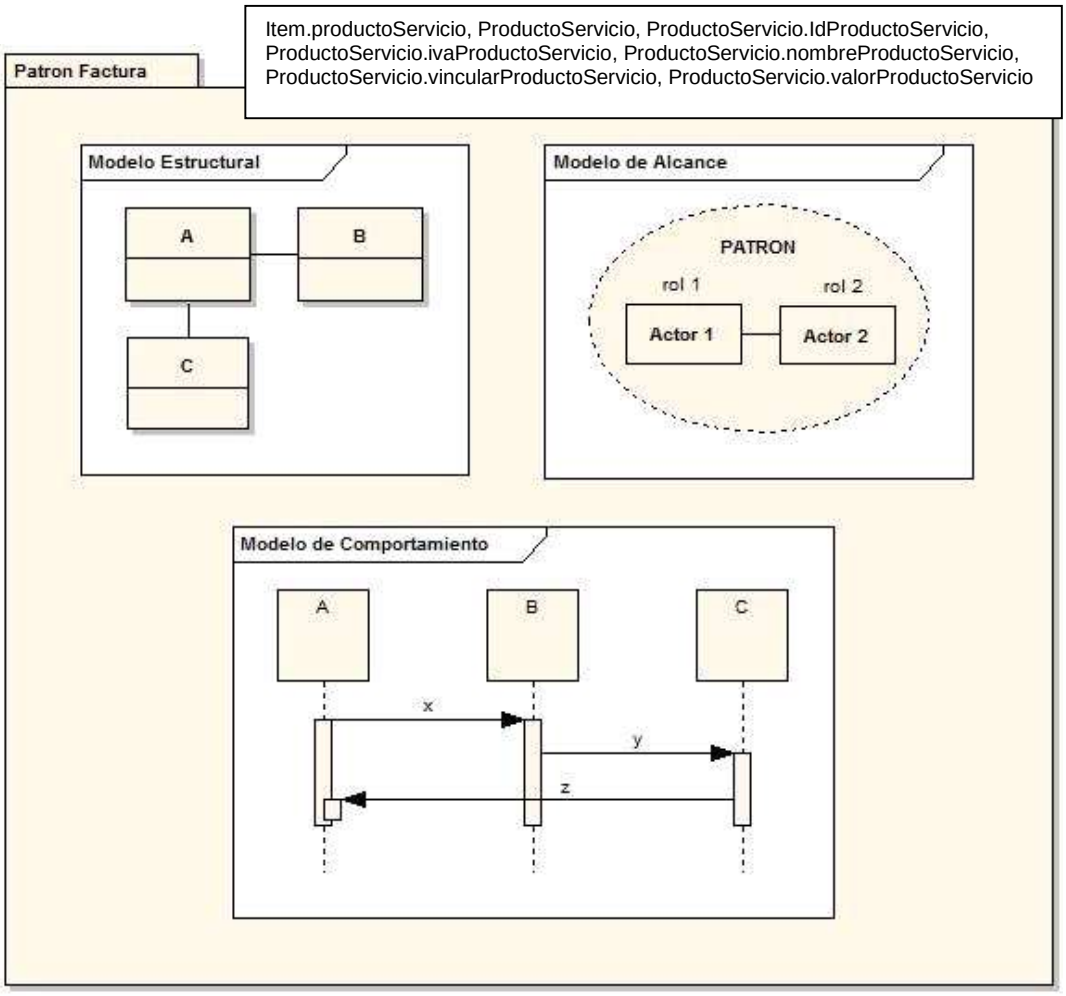
- ✓ **Modelo de comportamiento.** Los cambios realizados al modelo estructural afectan directamente a este modelo. El nuevo modelo se muestra a continuación:

Figura 51: Diagrama de secuencia del patrón Factura



- **Template del patrón Factura.** Para la identificación del signature, se analizaron los cambios que se realizaron a los modelos. El template de la solución genérica es el siguiente:

Figura 52: Template del patrón Factura



6.10. RESULTADOS DE VALIDACIÓN EN LA ETAPA 2: SIMULACIÓN DE USO Y RETROALIMENTACIÓN DEL CATÁLOGO.

En la siguiente tabla se muestra el resumen de validación por cada uno de los elementos de la simulación de uso y retroalimentación del catálogo siguiendo la estrategia de validación planteada anteriormente.

Tabla 20: Resultados de validación en etapa 2

Tipo	Elemento	Objetivo de elemento	Observaciones
Representación	Template	El Signature del template permite instanciar el patrón.	El signature es el componente del template que permite instanciar el patrón, convirtiéndolo en la solución de un problema específico. Es el elemento que incorpora el concepto aspectual.
Catálogo	Estructura	La estructura permite ejecutar los mecanismos de gestión de los patrones de análisis.	Con base en el proceso de simulación, se puede deducir que la estructura del catálogo si permite implementar los mecanismos de búsqueda, recuperación e instanciación de los patrones de análisis ya que provee los índices correspondientes para efectuarlos.

7. PLAN DE MEJORAMIENTO DE MORENA

7.1. PLANTILLA DE REQUISITOS

Se consideró realizar una búsqueda de plantillas ya que ESKEMA, aunque menciona y plantea la necesidad de descripción de requisitos como primer paso del proceso de ingeniería de dominio, no provee una. En esta búsqueda, se pudo observar que la gran mayoría de plantillas existentes para éste fin, se basan en el mismo modelo, que es el propuesto por Amador Durán Toro.

Amador Durán Toro[59], define distintos tipos de plantillas utilizados para la representación de requisitos durante la actividad de elicitación de la fase de ingeniería de requisitos del desarrollo de software.

La plantilla de requisitos que puede verse en la tabla 21, describe requisitos de la forma tradicional, y ayuda a responder a la pregunta "¿qué debe hacer el sistema con la información almacenada para alcanzar los objetivos de su negocio?".

Tabla 21: Plantilla para requisitos

FRQ–<*id*>	<*nombre descriptivo*>
Versión	<*nº de la versión actual*> (<*fecha de la versión actual*>)
Autores	• <*autor de la versión actual*> (<*organización del autor*>) ...
Fuentes	• <*fuente de la versión actual*> (<*organización de la fuente*>) ...
Objetivos asociados	• OBJ–*x* <*nombre del objetivo*> ...
Requisitos asociados	• Rx–y <*nombre del requisito*> ...
Descripción	El sistema deberá <*capacidad del sistema*>
Importancia	<*importancia del requisito*>
Urgencia	<*urgencia del requisito*>
Estado	<*estado del requisito*>
Estabilidad	<*estabilidad del requisito*>
Comentarios	<*comentarios adicionales sobre el requisito*>

Durán, Amador. Metodología para la Elicitación de Requisitos de Sistemas Software. Universidad de Sevilla, 2000

El significado de los campos específicos de esta plantilla es el siguiente:

[59] DURÁN, Amador. Metodología para la Elicitación de Requisitos de Sistemas Software. Universidad de Sevilla, 2000

- **Identificador y nombre descriptivo**: cada requisito se debe identificar por un código único y un nombre descriptivo. Con objeto de conseguir una rápida identificación, los identificadores de los requisitos comienzan con *FRQ.*

- **Versión**: para poder gestionar distintas versiones, este campo contiene el número y la fecha de la versión actual del requisito.

- **Autores, fuentes**: estos campos contienen el nombre y la organización de los autores (normalmente desarrolladores) y de las fuentes (clientes o usuarios), de la versión actual del requisito, de forma que la rastreabilidad pueda llegar hasta las personas que propusieron la necesidad del requisito.

- **Objetivos asociados**: este campo debe contener una lista con los objetivos a los que está asociado el requisito, es decir de los objetivos de los que depende. Esto permite conocer qué requisitos harán que el sistema a desarrollar alcance los objetivos propuestos y justifican de esta forma la existencia o propósito del requisito.
- **Requisitos asociados**: en este campo se indican otros requisitos que estén asociados por algún motivo con el requisito que se está describiendo, es decir de los requisitos de los que depende.

- **Descripción**: este campo contiene la capacidad o funcionalidad que debe presentar el sistema a desarrollar.

- **Importancia**: este campo indica la importancia del cumplimiento del requisito para los clientes y usuarios. Se puede asignar un valor numérico o alguna expresión enumerada como vital, importante o quedaría bien. En el caso de que no se haya establecido aún la importancia, se puede indicar que está por determinar (PD), equivalente al TBD (To Be Determined) empleado en las especificaciones escritas en inglés.

- **Urgencia**: este campo indica la urgencia del cumplimiento del requisito para los clientes y usuarios en el supuesto caso de un desarrollo incremental. Como en el caso anterior, se puede asignar un valor numérico o una expresión enumerada como inmediatamente, hay presión o puede esperar, o PD en el caso de que aún no se haya determinado.

- **Estado**: este campo indica el estado del requisito desde el punto de vista de su desarrollo. El requisito puede estar en construcción si se está elaborando, pendiente de negociación si tiene algún conflicto asociado pendiente de solución, pendiente de verificación si no tiene ningún conflicto pendiente y está a la espera de verificación o, pendiente de validación si ya ha sido verificado y está a la espera de validación o por último, puede estar validado si ya ha sido validado por clientes y usuarios.

- **Estabilidad**: este campo indica la estabilidad del requisito, es decir una estimación de la probabilidad de que pueda sufrir cambios en el futuro. Esta estabilidad puede indicarse mediante un valor numérico o mediante una expresión enumerada como alta, media o baja o PD en el caso de que aún no se haya determinado.

- **Comentarios**: cualquier otra información sobre el requisito que no encaje en los campos anteriores puede recogerse en este apartado.

7.1.1. Plantilla para los requisitos del dominio POS. Basados en lo anterior, se tomó la determinación de especificar los requisitos del dominio POS aplicando el modelo de plantillas de Amador Durán Toro con ciertas modificaciones:

- Se adiciona el campo Dominios asociados, en el cual se identifican los distintos dominios en los que se encuentra el requisito.

- Se adiciona el campo Patrón solución, en el cual se identifica el patrón que propone una solución al requisito.

- Se suprimen los campos: objetivos asociados, requisitos asociados, importancia y urgencia; ya que estos son más utilizados para problemas específicos.

La plantilla propuesta en esta investigación se muestra a continuación:

Tabla 22: Plantilla de requisito de dominio

FRQ - <id>	<nombre descriptivo>

Versión	<n^{o} de la versión actual> (<fecha de la versión actual>) …
Autores	<autor de la versión actual> (<organización del autor>)
Fuentes	<fuente de la versión actual> (<organización de la fuente>)
Descripción	<funcionalidad del requisito>
Dominios asociados	<diferentes dominios en los que se encuentra el requisito>
Patrón solución	<patrón que propone la solución del requisito>
Estado	<estado del requisito>
Estabilidad	<estabilidad del requisito>
Comentarios	<comentarios adicionales sobre el requisito>

7.2. TABLAS DE ASIGNACIÓN DE RESPONSABILIDADES

La creación de la colaboración que representa el modelo de alcance de un patrón de análisis, es un proceso difícil cuando no se tienen identificados los actores asociados, sus relaciones y los roles que desempeñan. Por este motivo, se sugiere la realización de una tabla llamada asignación de responsabilidades para facilitar dicha identificación.

Para el desarrollo de esta tabla, se llevan a cabo los siguientes pasos:

- Establecer las acciones que debe realizar el patrón de análisis (Responsabilidades).

- Determinar las clases encargadas de ejecutar cada acción (Clases responsables).

- Identificar los actores participantes en la ejecución de las acciones (Actores responsables).

El diseño de la tabla se muestra a continuación:

Tabla 23: Asignación de responsabilidades

RESPONSABILIDAD	CLASE RESPONSABLE	ACTOR RESPONSABLE

7.3. DESCRIPCIÓN DEL PATRÓN ASPECTUAL DE ANÁLISIS

Siguiendo el proceso de ESKEMA, se observa que no existe un punto en donde se realiza la descripción del patrón; se propone que ésta sea incorporada en la etapa de catalogación que incluye la ingeniería de dominio para facilitar la búsqueda, recuperación e instanciación del patrón.

En la descripción del patrón que propone MORENA, se ve la necesidad de adicionar los campos: roles participantes y requisitos de dominio ya que son los criterios de indexación definidos en el directorio.

La plantilla de descripción del patrón se muestra en la siguiente tabla:

Tabla 24: Plantilla de descripción del patrón

Nombre	<nombre descriptivo>
Dominios	<diferentes dominios en los cuales se aplica>
Descripción	<funcionalidad del patrón>
Requisitos de dominio	<requisitos que resuelve>
Roles participantes	<roles asociados al patrón>
Usos conocidos	<aplicaciones realizadas>
Patrones relacionados	<patrones con los cuales se relaciona>

7.4. DIAGRAMA DE CASOS DE USO

En MORENA, el modelo de alcance permite definir como el patrón de análisis interactúa con su contexto y en especial con los usuarios de acuerdo con los roles que asumen. MORENA propone la colaboración como mecanismo de representación del modelo de alcance.

En coherencia con el objetivo del modelo de alcance en MORENA, se sugiere emplear el diagrama de casos de uso como mecanismo de representación. El

diagrama de casos de uso es el más utilizado por su facilidad de lectura y escritura para describir las acciones desarrolladas entre los actores, el sistema y su entorno, que generan resultados observables.

CONCLUSIONES

1. La validación de MORENA permitió demostrar la eficiencia del modelo integralmente. Mediante el proceso de validación, se identificaron algunas deficiencias para las cuales se determinaron los procesos correctivos. Este proceso, permite retroalimentar la definición del modelo en tiempo de construcción y garantizar su eficiencia para el tiempo de uso.

2. Los elementos de validación del modelo son los aspectos en los cuales el proceso de validación se debe concentrar, estableciendo la estrategia y los ejercicios que permitan determinar el cumplimiento de sus objetivos para fijar el curso a seguir en el mejoramiento del modelo en construcción.

3. La estrategia de validación de modelos es el proceso que rige la validación del modelo, determinando para cada uno de los elementos de validación el ejercicio que se debe ejecutar y poder establecer los mecanismos de retroalimentación que optimizan el modelo en construcción.

4. Para validar MORENA, fue necesario identificar un espacio de acción que permitiera validarlo de manera integral y cada una de sus partes de manera particular. Este escenario fue el dominio POS, el cual ayudó a definir las deficiencias y eficacias del modelo en un entorno práctico

5. El uso de templates de la especificación de Superestructura de UML 2.0, permitió la representación de los patrones aspectuales de análisis a través del empaquetamiento de los diagramas que representan los modelos que lo definen. Por medio del signature del template, fue posible instanciar el patrón aspectual de análisis para convertirlo en la solución de un problema específico.

6. El objetivo final de todo proceso de validación es determinar aquellos aspectos que son susceptibles de mejoramiento del modelo, estos aspectos se consignan en el plan de mejoramiento que integra las sugerencias que posibilitan la optimización del modelo en construcción. El planteamiento de un plan de mejoramiento, solo es válido si para su definición se ha seguido un proceso consciente de ingeniería del software para la validación del modelo.

TRABAJOS FUTUROS

Como complemento a la presente investigación, se requiere demostrar la eficiencia de MORENA y del catálogo de patrones aspectuales de análisis en un caso de estudio especifico. Para ello se requiere la identificación del caso de estudio específico, la definición de requisitos de aplicación y la posterior instanciación de los patrones de análisis para las soluciones de problemas específicos del caso de estudio. Esta experiencia debe ser descrita y documentada de tal manera que sirva de referente para estudios sobre reutilización de software, catálogos, patrones de análisis y modelos de representación.

Para hacer práctico el uso de catálogos de patrones de análisis, se propone el desarrollo de una herramienta que pueda interactuar con diferentes herramientas CASE, haciendo portables los activos de software ofrecidos por el catálogo.

A partir del desarrollo de esta investigación, se propone el desarrollo de proyectos que busquen robustecer el catálogo de patrones de análisis desarrollado, implementando nuevos patrones para nuevos dominios específicos.

RECOMENDACIONES

1. Desarrollar nuevos patrones aspectuales de análisis que presenten una propuesta de solución a requisitos asociados a dominios diferentes al POS, con el fin de robustecer el catálogo.

2. Implementar TRIPODE en un caso de estudio específico para poder evaluar su utilidad al momento de realizar un proyecto de desarrollo de software real.

3. Reutilizar software desarrollado con anterioridad, con el fin de aprovechar los activos existentes y lograr un incremento en la calidad de los productos software.

4. Incursionar en el área de la orientación a aspectos como complemento al paradigma objetual, para así, mejorar y optimizar el proceso de desarrollo de software.

REFERENCIAS BIBLIOGRÁFICAS

ALEXANDER, Christopher. ISHIKAWA, Sara. SILVERSTEIN, Murray. JACOBSON, Max. FIKSDAHL-KING, Ingrid. ANGEL, Shlomo. A Pattern Language. New York: Oxford University Press, 1977

American Marketing Asociation. MarketingPower.com, sección Dictionary of Marketing Terms. Internet (www.marketingpower.com <http://www.marketingpower.com/>)

ANAYA, Raquel. "Presentación de resultados proyecto MMEDUSA: Marco Metodológico para Desarrollo de Aplicaciones Utilizando la Aproximación de Aspectos", 2008.

BARÓN, A. ESKEMA – Esquema para la Gestión de Catálogos de Patrones de Análisis utilizando el enfoque aspectual. Maestría en Ingeniería Informática, Universidad Eafit, Medellín Colombia. 2010.

BOOCH, G. RUMBAUGH, J. JACOBSON, I. El Lenguaje Unificado de modelado, Pearson Rentice Hall, 2006.

BUSCHMANN, Frank, MEUNIER, Regine, ROHNERT, Hans, SOMMERLAD, Peter, STAL, Michael. (1996). Pattern-oriented software architecture: A system of patterns. Chichester, UK: John Wiley & Sons.

CHITCHYAN, R. RASHID, A. SAWYER, P. GARCIA, A. PINTO, M. BAKKER,J. TEKINERDOGAN, B. CLARKE, S. JACKSON, A. Survey of Aspect-Oriented Analysis and Design Approaches. AOSD-EUROPE. 2005

Computación Estadística – Curso 2008-2009. Internet: (wwwdi.ujaen.es/asignaturas/computacionestadistica/pdfs/tema6.pdf <http://wwwdi.ujaen.es/asignaturas/computacionestadistica/pdfs/tema6.pdf>)

CONSTANTINIDES, Constantinos, ELRAD, Tzilla, y FAYAD, Mohamed. Extending the object model to provide explicit support for crosscutting concerns. s.l: John Wiley & Sons, 2002.

COPLIEN, J.O. "Agenerative Development-process Pattern Language". In Pattern Language of program design. J.O. Coplien and D.C. Schmidt, ED. Reading, MA:Addison-Weley, 1995.

DIJKSTRA, Edsger W., A Discipline of Programming, Prentice-Hall,1976.

DURÁN, Amador. Metodología para la Elicitación de Requisitos de Sistemas Software. Universidad de Sevilla, 2000

FERNANDEZ, Eduardo B., “Building Systems Using Analysis Patterns”, Florida Atlantic University, 2005.

FOWLER, Martin. Analysis Patterns, Reusable objects models, Addison Wesley, 1997.

GAMMA, E., R. HELM, R., Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Objects-Oriented Software Reading, MA: Addison Weley, 1995.

GEYER-SCHULZ, Andreas; HAHSLER, Michael (2002). Software reuse with analysis patterns.

GIRALDO, Juan Esteban. El comercio en la economía, 2002. Internet: (www.gestiopolis.com/recursos/documentos/fulldocs/eco/comeco.htm <http://www.gestiopolis.com/recursos/documentos/fulldocs/eco/comeco.htm>)

HAY, David. Data Model Patterns: Conventions of Thought. New York: Dorset House, 1996

Laboratorios Docentes de la E.T.S.I. Informatica. Internet: (jair.lab.fi.uva.es/~pablfue/leng_simulacion/materiales/v_v_0405.pdf <http://jair.lab.fi.uva.es/~pablfue/leng_simulacion/materiales/v_v_0405.pdf>)

LONDOÑO, Luis Fernando. Definición de las Necesidades para la Construcción de un Sistema de Puntos De Venta, Universidad EAFIT, 2008.

MARTÍNEZ, I.J. (2005): La comunicación en el punto de venta: estrategias de comunicación en el comercio real y on-line. ESIC, Madrid.

MESZAROS, Gerard; DOBLE, Jim (1998). A Pattern Language for Pattern Writing, Addison-Wesley

NAYLOR, Thomas H. FINGER J. M. MCKENNEY, James L., SCHRANK William E. HOLT, Charles C. Verification of Computer Simulation Models, *Management Science*, Vol. 14, No. 2, Application Series (Oct., 1967)

Object Management Group, Unified Modeling Language: Superstructure, Versión 2.1.1, 2007

ORTÍN, Maria José; GARCIA MOLINA, José. “Modelado basado en roles con UML”.Departamento de Informática, Lenguajes y Sistemas. Facultad de Informática - Universidad de Murcia.

OSSHER, H. TARR, P.: "Multi-Dimensional Separation of Concerns and The Hyperspace Approach." In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. Kluwer, 2000.

RIEHLE, Dirk. ZÜLLIGHOVEN, Heinz. (1996). Understanding and using patterns in software development. Theory and Practice of Object Systems.

Software Engineering Institute. Internet: (www.sei.cmu.edu/domain-engineering/domain_anal.html <http://www.sei.cmu.edu/domain-engineering/domain_anal.html>)

The free Dictionary. Internet: (es.thefreedictionary.com/cat%C3%A1logo <http://es.thefreedictionary.com/cat%C3%A1logo>)