Centro de Investigación en Matemáticas, A.C.

Un proceso híbrido para el diseño de líneas de producto de software

REPORTE TÉCNICO

Que para obtener el grado de

Maestro

en

Ingeniería de Software

P r e s e n t a

Jesús Ernesto Díaz Díaz

Director de Tesis

Dra. Perla Inés Velasco Elizondo

Zacatecas, Zac., Julio de 2010

Contenido

Índice de figuras

Capítulo 1. Introducción

En este capítulo de presentará una descripción general del presente documento, a fin de obtener una idea clara de su alcance y organización, respecto a la problemática que será objeto de estudio.

1.1 Antecedentes

Hoy en día existe una gran variedad de paradigmas de desarrollo de software tales como orientados a objetos, basados en componentes, o líneas de productos, entre otros. Para cada uno de estos paradigmas se han propuesto distintas maneras de resolver el proceso de desarrollo de software. Esto es, existen distintas maneras en las que se puede ir desde la captura de requerimientos hasta la instalación de los sistemas construidos bajo estos paradigmas. En este documento nos concentraremos en los paradigmas **software basado en componentes** y el **software de líneas de producto,** así como algunos de sus procesos de desarrollo propuestos. Esto con el propósito de definir un nuevo proceso para el diseño de líneas de producto que involucre el uso de componentes de software.

El software basado en componentes se centra en la idea de reutilizar distintos elementos pre-existentes de software (llamados componentes), a fin de formar un sistema final. En términos generales, un componente se define como una "unidad independiente de software, con puntos y modos de interacción bien definidos, la cual puede estar combinada con otras unidades dentro de un sistema de tamaño mayor"[1].

El paradigma de líneas de producto de software está basado en la generación y reutilización de *core assets* permitiendo con esto la creación de familias de productos. Dentro de los distintos core assets que se pueden generar y/o reutilizar encontramos a los del software.

1.2 Problemática

Para cada uno de los paradigmas citados anteriormente, encontramos distintos métodos que describen el proceso de desarrollo de software. Cada uno de estos métodos difiere en aspectos como alcance (etapas del proceso de desarrollo que considera), sintaxis, artefactos generados, nivel de detalle en las descripciones de las actividades consideradas por el proceso, etc.

Como mencionamos anteriormente, en este documento nos concentraremos en métodos que describen el proceso de desarrollo de software basado en componentes y en métodos que describen el proceso de desarrollo de líneas de productos de software. Específicamente UML Components[3] y Product Line UML-Based Software Engineering (PLUS)[2].

UML Components es un método que describe un proceso de desarrollo de software basado en componentes. El método usa como sintaxis UML (Unified Modeling Language) y divide el proceso en fases llamadas *workflows*. Para cada uno de estos workflows, el método indica de manera

puntual los artefactos que se deben de tener como entrada. Así mismo, el método describe con detalle que artefactos y como se generarán y evolucionarán cada uno de ellos durante cada workflow. El uso de sintaxis UML así como el alto nivel de detalle que presentan las descripciones de cada una de las actividades de cada workflow hacen que este método sea fácil de adoptar en la práctica por diseñadores de software novatos.

PLUS es un método que describe un proceso de desarrollo de líneas de productos de software. En este método se utiliza como sintaxis UML y se tiene dividido el proceso en fases. Si bien el método indica los artefactos a generar en cada una de estas fases, no indica detalladamente la relación y evolución de estos en cada una de las fases. Esto ocasiona que el método se vuelva complejo para llevarlo a la práctica cuando es utilizado por usuario con poca experiencia.

Tomando en cuenta las fortalezas y carencias detectadas en cada uno de los métodos, es como surge la idea de generar un proceso híbrido para guiar el diseño de líneas de producto que involucre el uso de componentes de software preservando las fortalezas y resolviendo las carencias de los procesos descritos en cada uno de estos métodos.

1.3 Objetivos

- Desarrollar un proceso híbrido para guiar el diseño de líneas de producto de software basado en UML Components y en PLUS.
- Desarrollar un proceso que rescate las fortalezas de cada uno de los métodos en los que se encuentra basado.
- Disminuir en el proceso híbrido las carencias propias de cada método.

1.4 Alcance

En este documento podremos encontrar información básica de los paradigmas de desarrollo de software tales como desarrollo basado en componentes y líneas de producto. Se describirán dos métodos propuestos para el proceso de desarrollo de cada uno de estos paradigmas.

Se presentará la definición de un proceso híbrido que combine los métodos UML Components y PLUS.

El proceso definido solo considera las fases de requerimientos y especificación; las fases de implementación, ensamblaje y pruebas no se tienen previsto definirlas.

1.5 Organización

El presente documento se encuentra dividido en siete capítulos.

- Capítulo 1. Introducción
- Capítulo 2. Software basado en componentes y su proceso de desarrollo
- Capítulo 3. Líneas de producto de software y su proceso de desarrollo
- Capítulo 4. Un proceso híbrido para el desarrollo de líneas de producto de software
- Capítulo 5. Caso de estudio
- Capítulo 6. Evaluación

- Capítulo 7. Conclusiones y trabajo futuro

En el capítulo 2, encontraremos los conceptos básicos del desarrollo basado en componentes, así como el método de UML Components.

En el capítulo 3, se abordarán los conceptos básicos del desarrollo de líneas de producto de software y el método PLUS.

En el capítulo 4, se presentará la propuesta del proceso híbrido que combina los procesos en los métodos UML Components y el método PLUS.

En el capítulo 5, se mostrará el caso de estudio con el objeto de corroborar las ventajas y desventajas del proceso propuesto.

En el capítulo 6, es realizada la evaluación del proceso propuesto.

En el capítulo 7, finalmente presentaremos las conclusiones del presente documento, así como la posible línea de trabajo a futuro.

Capítulo 2. Software basado en componentes y su proceso de desarrollo

En este capítulo se abordarán los conceptos del desarrollo de software basado en componentes, al igual que las características principales de este paradigma. También describiremos el método conocido como UML Components.

2.1 Desarrollo basado en componentes

Uno de los principales objetivos de la ingeniería de software es disminuir los tiempos y costos involucrados en el proceso de desarrollo de software. Durante varios años, gran número de paradigmas a este respecto han sido propuestos. "Desarrollo de software basado en componentes" recibe por nombre uno de ellos. Este paradigma está basado en la idea de tener una serie de elementos de software pre-existentes y reutilizarlos para la construcción de varios sistemas.

Técnicamente hablando, dentro del paradigma de software basado en componentes los elementos de software reciben el nombre de "componentes". Según Clemens Szyperski durante el WCOP'96 un componente de software lo define como[1]:

"Un componente software es una unidad de composición en formato ejecutable con interfaces especificadas contractualmente y dependencias explícitas del contexto. Un componente de software se puede instalar independientemente y está sujeto a composición por terceros."

2.2 Métodos de desarrollo

Para llevar a cabo el proceso de desarrollo de software basado en componentes en la práctica, se han generado una serie de métodos. En este estudio abordaremos el método conocido como UML Components.

2.3 UML Components

UML Components es un método para el diseño de software basado en componentes. Desarrollado por John Chessman y John Daniels, se finca en la sintaxis y la semántica de UML. Caracterizado por ser un método que indica claramente las fases del flujo del proceso de desarrollo, define los artefactos que se generan dentro de cada una de éstas permitiendo ir evolucionando los artefactos. En UML Components dichas fases reciben el nombre de workflows. En las siguientes secciones describimos cada uno de ellos.

2.3.1 Workflows

En términos generales los *workflows* representan la secuencia de actividades que dan como resultado uno o más artefactos.

La Figura 1 nos muestra los workflows definidos en UML Components.

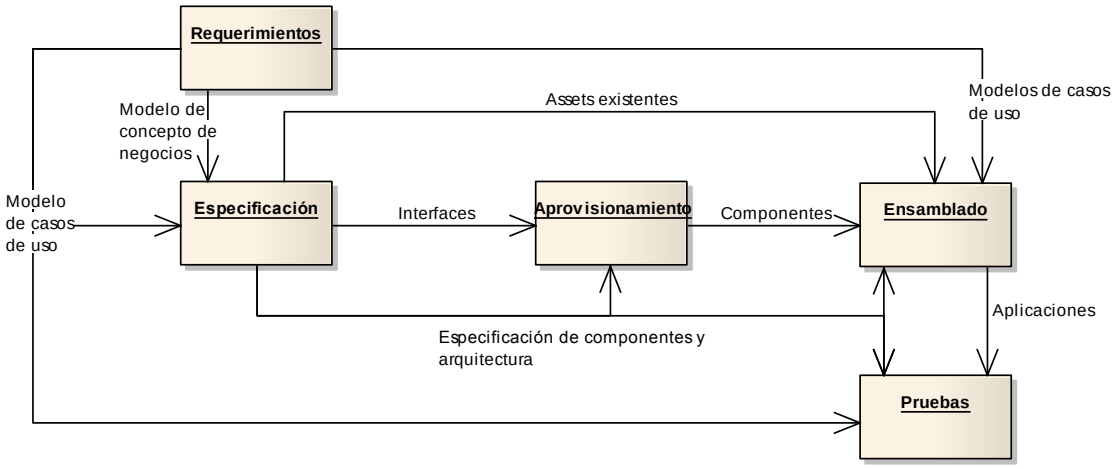


Figura 1. Workflows definidos en UML Components.

Las flechas representan el flujo de información entre un workflow y el otro. Como puede observarse en la figura UML Components considera cinco workflows que son:

1. Requerimientos
2. Especificación
3. Aprovisionamiento
4. Ensamblado
5. Pruebas

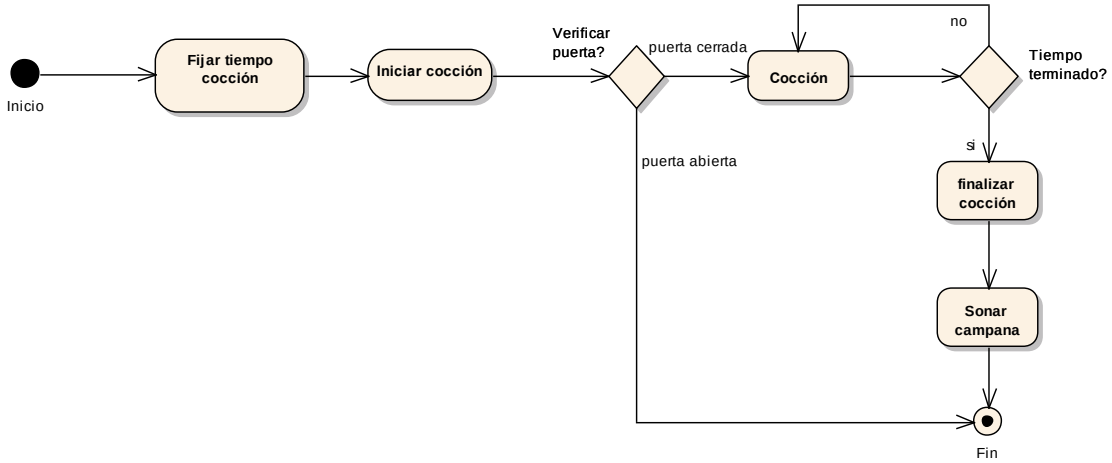
2.3.1.1 Requerimientos

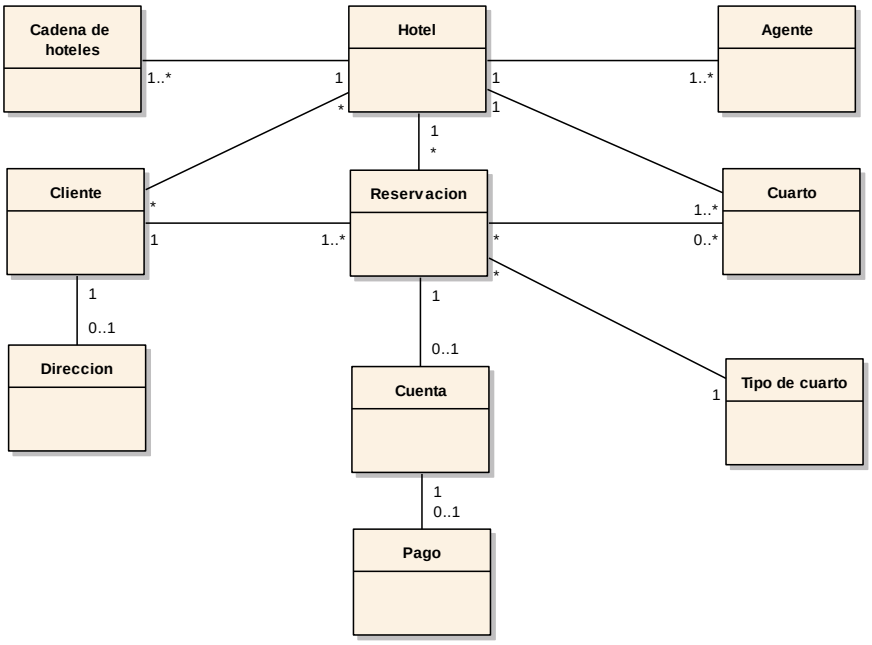
En este workflow se tiene por objeto hacer el análisis de los requerimientos del sistema a desarrollar. Los artefactos a generar son:

- Modelo de procesos de negocio
- Modelo conceptual de negocio
- Modelo de casos de uso

El **modelo de procesos de negocio** nos muestra el flujo normal que se sigue en nuestro negocio.
El **modelo conceptual de negocio** es un mapa de los conceptos del negocio que busca generar un vocabulario común entre las personas involucradas con el proyecto.

El **modelo de casos de uso** es la forma de especificar los requerimientos funcionales del sistema. En él se describen las interacciones entre éste último y el usuario, y se define su frontera. Ejemplos de cada uno de estos modelos se muestran, en la Figura 2.





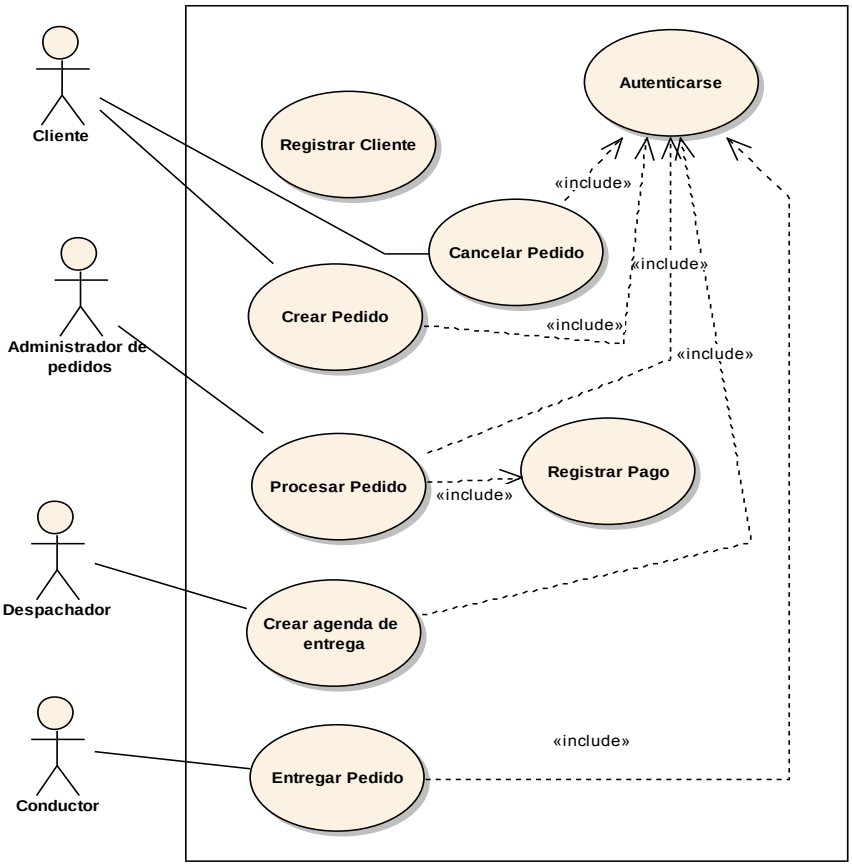


Figura 2. Ejemplos de modelo de procesos de negocio, modelo de conceptos de negocio y modelo de casos de uso.

2.3.1.2 Especificación

Este workflow tiene por objetivo generar las especificaciones de los componentes así como la arquitectura. Se encuentra dividido en tres subfases: identificación de componentes, interacción de componentes y especificación de componentes. En la Figura 3, podemos observar ilustradas las subfases del workflow de especificación.

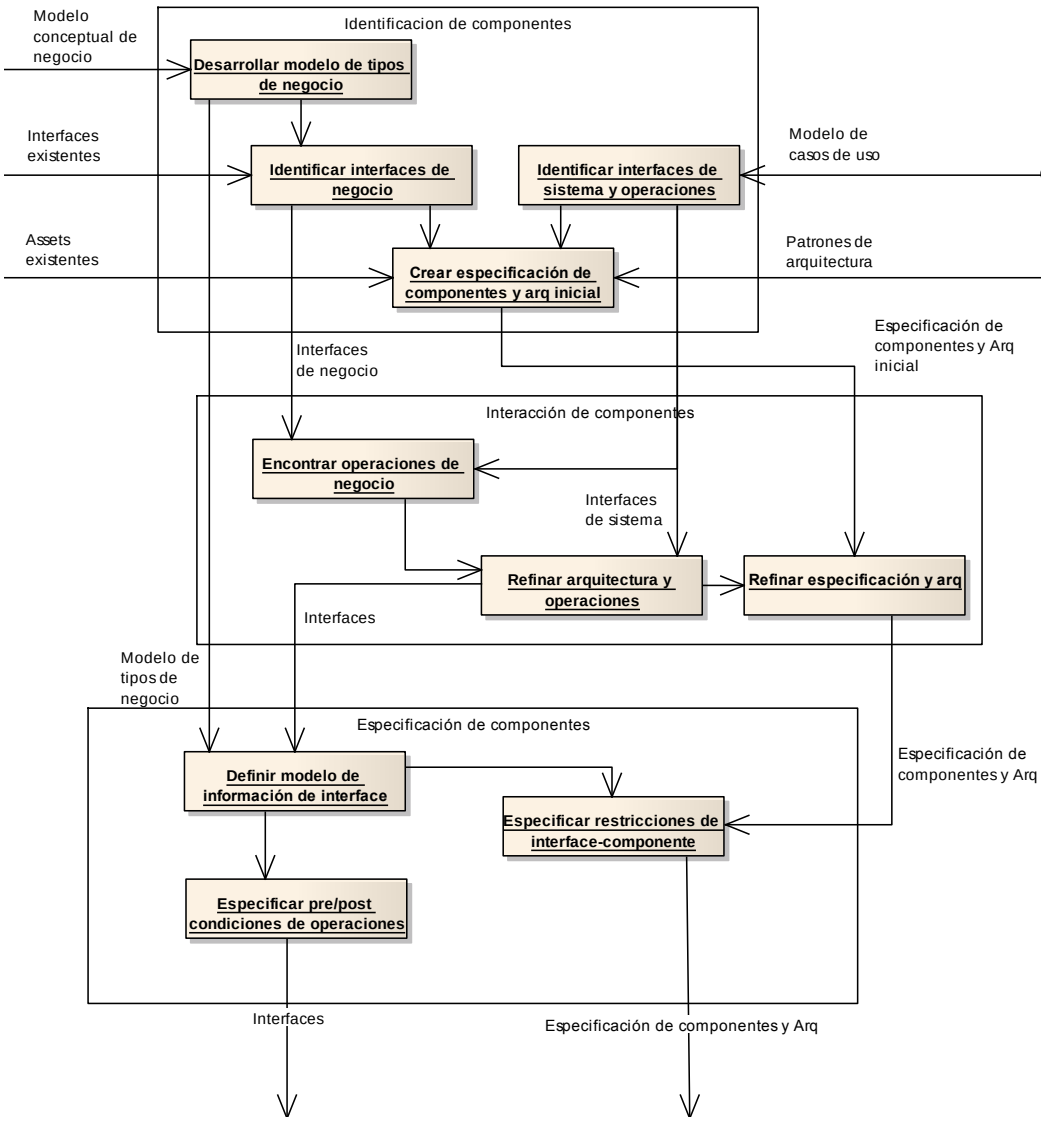


Figura 3. Las tres subfases del workflow de especificación.

Identificación de componentes

En este apartado se toman como base los modelos de conceptos de negocio y de casos de uso generados previamente en requerimientos. Se toman en cuenta los assets existentes a fin de identificar aquellos que se pueden reutilizar en el sistema a desarrollar. Su objetivo es identificar un conjunto inicial de interfaces de negocio para los componentes de negocio, y de interfaces de sistema para los componentes de sistema, de forma que uniéndolas pueda generarse una arquitectura inicial.

Se crea el "modelo de tipos de negocio", un artefacto que tiene la forma de un diagrama de clases en UML y tiene como propósito formalizar —mediante el uso del lenguaje OCL (Object Constraint Language) [4]— el modelo de conceptos de negocio para definir el conocimiento del sistema en el

mundo exterior. Este modelo sentará las bases para identificar las interfaces; esto puede observarse en la Figura 4. Aquí podemos encontrar las restricciones, las cuales se encuentran escritas entre llaves ({}) y en OCL.

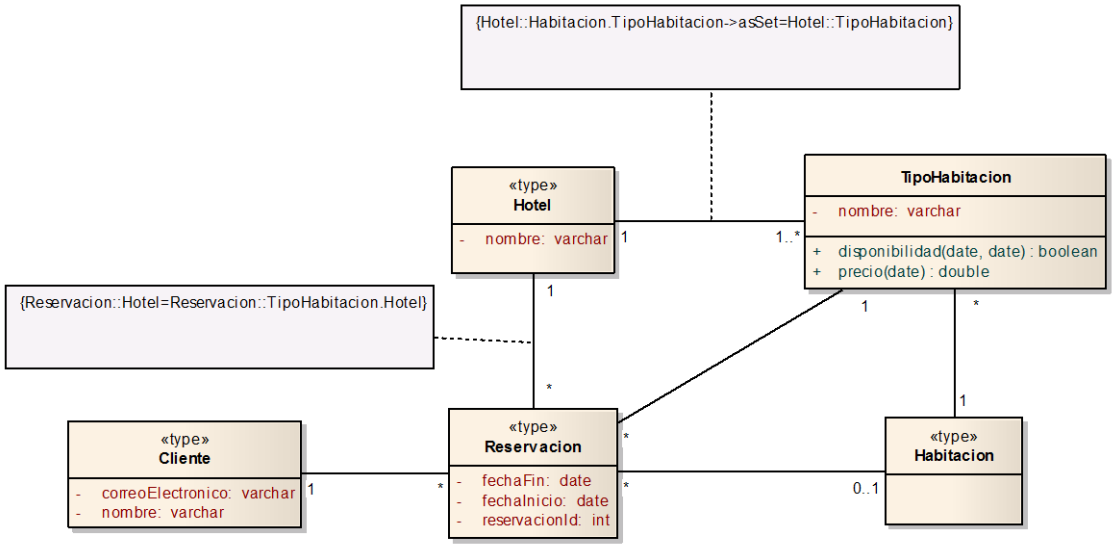


Figura 4. Modelo de tipos de negocio.

Además de identificar las interfaces de sistema, en la subfase de identificación también se genera un primer acercamiento de las operaciones que necesitan ser soportadas por el sistema.

Se genera el diagrama de responsabilidad de interfaces, el cual está basado en el modelo de tipos de negocio y muestra cómo las interfaces interactúan con la información. En la figura 5 podemos observar el diagrama de responsabilidad de interfaces.

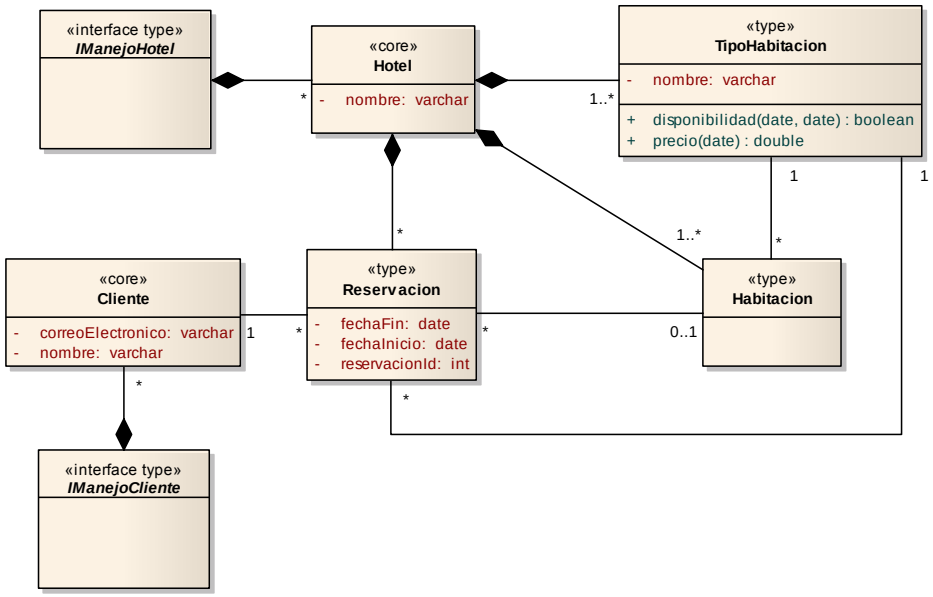


Figura 5. Diagrama de responsabilidad de interfaces

Interacción de componentes

En la subfase de interacción de componentes se examina como cada una de las operaciones se logrará generar en base a la arquitectura de componentes. Se utiliza modelos de interacción para descubrir las operaciones dentro de las interfaces de negocio. A mayor número de interacciones se consideren, se encontrarán operaciones comunes.

En esta subfase se generan artefactos que nos muestran la interacción de las interfaces. Se utilizan diagramas de colaboración de UML para representar las interacciones. En la figura 6 observamos un diagrama de colaboración.

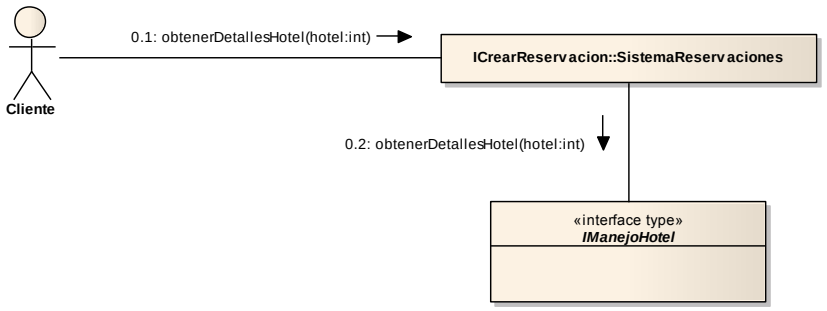


Figura 6. Diagrama de colaboración

Especificación del componente

Durante esta subfase se genera la especificación a detalle de las operaciones y restricciones. Para una interface significa definir los posibles estados de un componente en un modelo de información de interface y especificar las pre y post condiciones de las operaciones, así como capturar las reglas de negocio como restricciones.

El artefacto de la especificación de interface, presentado en la figura 7, nos muestra la interface con sus operaciones, así como con cuales clases se relaciona.

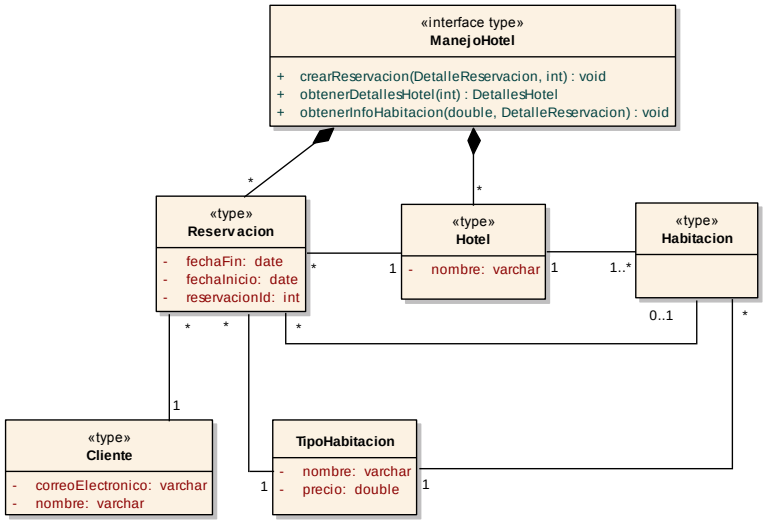


Figura 7 Diagrama de especificación de interface

2.3.1.3 Aprovisionamiento

El propósito del aprovisionamiento es determinar cuáles serán los nuevos componentes, ya sea a través de la implementación de las especificaciones, o encontrando un componente que coincida con ellas.

2.3.1.4 Ensamblado

Es el proceso en que todos los componentes se concentran en un sistema, definiendo una interfaz de usuario basada en el modelo de casos de uso, de manera que se forme una aplicación.

2.3.1.5 Pruebas

Durante la fase de pruebas es importante llevar a cabo las pruebas de unidad, integración, sistema y aceptación.

Capítulo 3. Líneas de producto de software y su proceso de desarrollo

En este capítulo se presentarán los conceptos básicos del paradigma de líneas de producto de software. Se explicará el método PLUS, que es para soportar, el diseño de líneas de producto de software.

3.1 Desarrollo basado en líneas de producto de software

Como ya se mencionó anteriormente una línea de producto de software consiste en un conjunto de sistemas que poseen una parte común y una parte variable. Las líneas de producto de software buscan el beneficio de volver reutilizable la parte común del conjunto de sistemas.

Las líneas de producto no son nuevas, existiendo distintos ejemplos de ellas a través de la historia. Las pirámides de Egipto son consideradas como la primera línea de producto desarrollada por la humanidad. Actualmente se encuentran en distintas áreas de la ingeniería, por ejemplo en la industria aérea: dentro de la compañía Airbus podemos descubrir que los aviones A-318, A-319, A-320 y A-321 comparten características comunes tales como motores, equipo de navegación y equipo de comunicación.

Técnicamente hablando una línea de producto de software es “un conjunto de sistemas de software intensivos que comparten algo en común, administran una serie de características que satisfacen necesidades específicas de un segmento del mercado y son desarrolladas desde un conjunto base en una determinada manera”[2].

3.1.2 Principios de las líneas de producto de software

Modelar la “comunalidad” y la “variabilidad” es una de las más importantes actividades en el desarrollo de una línea de producto de software.

En términos generales comunalidad se define como “la funcionalidad que es común para todos los miembros de una línea de producto de software”[2], y se refiere variabilidad a “la funcionalidad provista por algunos —pero no por todos— los miembros de una línea de producto de software”[2].

3.2 Métodos de desarrollo

Dentro del paradigma de líneas de producto de software encontramos distintos métodos, entre los que se ubica el *Product Line UML-Based Software Engineering* más conocido por sus siglas como “PLUS”, y KobrA[1], entre otros.

En este documento presentaremos y utilizaremos el método PLUS.

3.3 Product Line UML-Based Software Engineering (PLUS)

El método PLUS, desarrollado por Hassan Gomma, está propuesto para el diseño de líneas de producto de software. Basado en la sintaxis y la semántica de UML, busca ser capaz de extender otros métodos de diseño, de forma que se puedan modelar y diseñar líneas de producto.

El proceso de desarrollo del método PLUS es iterativo y orientado a objetos, y resulta compatible con el Proceso de Desarrollo Unificado de Software (*Unified Software Developtment Process*) y el Modelo de Proceso Espiral (*Spiral Process Model*).

El Proceso de Ingeniería Evolucionario de Líneas de Producto de Sofware (Evolutionary *Software Product Line Engineering Process*, ESPLEP) es el proceso en el que el método PLUS está basado y que para mayor conocimiento, detallaremos a continuación.

3.3.1 Evolutionary Software Product Line Engineering Process

El modelo Evolutionary Software Product Line Engineering Process (ESPLEP) es un proceso que elimina la distinción entre el desarrollo de software y el mantenimiento; en lugar de esto los sistemas evolucionan a través de una serie de iteraciones, debiendo ser capaces de adaptarse a los nuevos cambios de requerimientos que se generan durante cada iteración, para lo que toman la perspectiva de las líneas de producto de software, permitiendo el desarrollo de familias.

El ESPLEP se divide en dos grandes procesos o ciclos de vida, que se muestran en la figura 8 y que son:

1. Ingeniería de líneas de producto de software: En este proceso, la parte común y variable de la línea de producto es analizada de forma general conforme a los requerimientos. Consiste en desarrollar los modelos de casos de uso, modelos de análisis de líneas de producto, arquitectura de línea de producto de software y componentes reutilizables.
2. Ingeniería de aplicaciones de software: Aquí, una aplicación que es parte de la línea de producto de software se desarrolla, pero en lugar de comenzar de cero, se utilizan los artefactos generados en el ciclo de vida de la línea de producto de software.

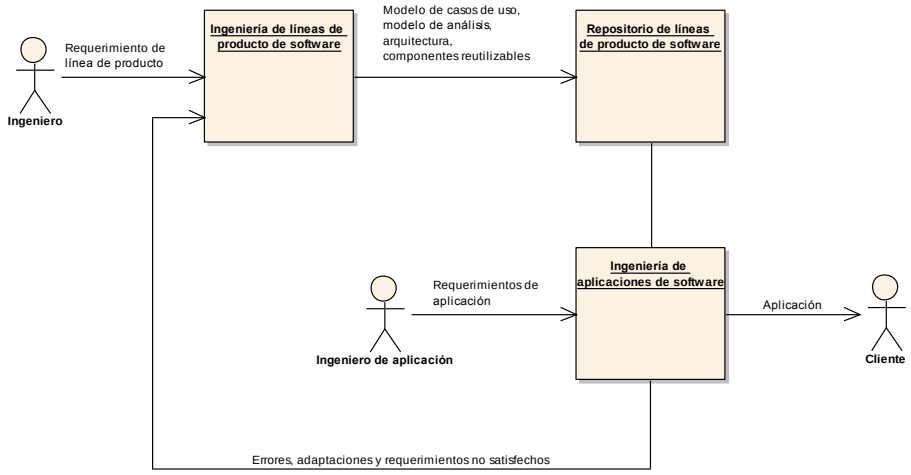


Figura 8. Evolutionary Software Product Line Engieenring Process

3.3.2 Fases de la ingeniería de líneas de producto de software

Como ya se ha mencionado, el modelo ESPLEP es un proceso de desarrollo iterativo basado en el concepto de casos de uso. Durante la **fase de requerimientos**, los requerimientos funcionales de una línea de producto de software son definidos en términos de actores y casos de uso. Durante la **fase de análisis**, cada caso de uso de la línea de producto de software describe los objetos que participan y sus interacciones. Durante la **fase de diseño**, se desarrolla la arquitectura basada en componentes. Podemos observar las fases en la Figura 9, donde todos los artefactos generados se almacenan en un repositorio de la línea de producto de software.

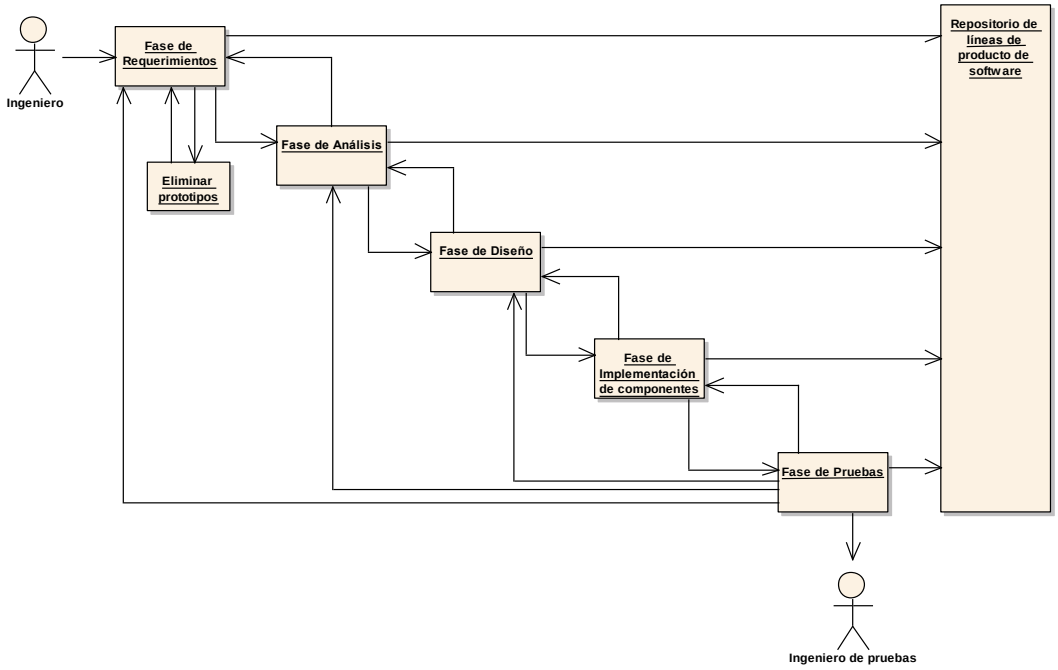


Figura 9. Ingeniería de líneas de producto de software con ESPLEP

En las siguientes secciones se describen cada una de las fases ilustradas en la Figura 9.

3.3.2.1 Fase de Requerimientos

En esta fase se realiza el análisis de los requerimientos de la línea de producto de software, teniendo por objetivo generar los artefactos que indiquen los requerimientos. Los artefactos a generar son “modelo de casos de uso” y “modelo de características”.

Modelo de casos de uso

El modelo de casos de uso define los requerimientos funcionales de la línea de producto en términos de actores y casos de uso.

Sin embargo el modelo de casos de uso de una línea de producto de software necesita ser enriquecido de forma que se pueda modelar la parte común y la parte variable. Esto se logra con

el desarrollo de los casos de uso de tipo “kernel”, “opcional” y “alternativos” (*kernel*, *optional*, *alternative*).

Los casos de uso que son requeridos por todos los miembros de la línea de producto de software son referidos como casos de uso kernel. Los casos de uso opcionales son requeridos por algunos pero no por todos los miembros de la línea de producto de software. Otros casos de uso son alternativos, es decir, distintas versiones del caso de uso son requeridas por distintos miembros de la línea de producto de software, y son de manera usual mutuamente excluyentes. En la Figura 10 podemos observar un ejemplo del modelo de estos casos de uso.

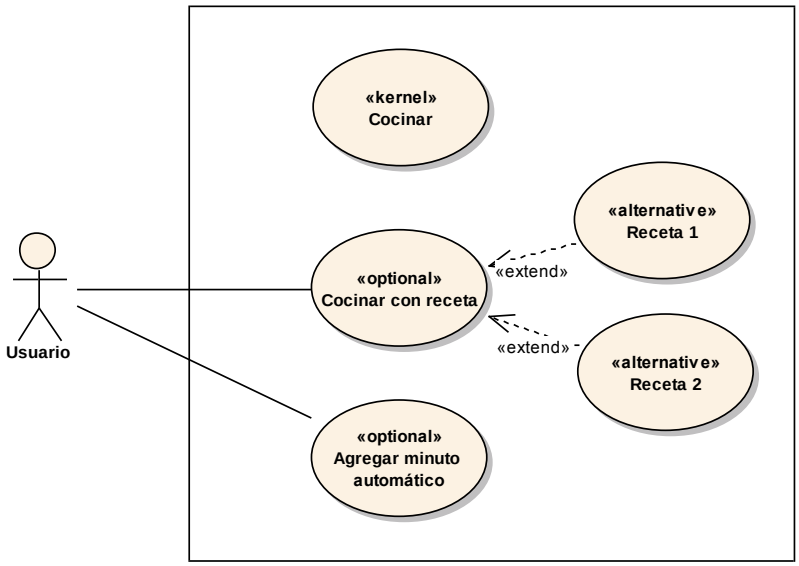


Figura 10. Ejemplo de modelo de casos de uso.

Los puntos de variación en el diagrama de casos de uso se pueden representar de dos formas:

1. Documentado dentro del mismo caso de uso
2. Como una relación de ampliación

Modelo de características

Una característica es un requerimiento provisto por uno o más miembros de la línea de producto, utilizada para diferenciar y por lo tanto determinar la funcionalidad común y variable. Las características a su vez, pueden ser funcionales o parametrizadas.

Las características comunes y variables, son categorizadas como comunes, opcionales o alternativas. Las **comunes** determinan el grado de comunalidad en la línea de producto; las **opcionales** y **alternativas** determinan el grado de variabilidad en la línea de producto.

Características comunes (*common features*): deben de ser provistas por cada uno de los miembros de la línea de producto. Normalmente estas características son denominadas como “necesarias”, “obligatorias” o “kernel”. Es denotada en UML con el estereotipo de *common feature*.

Características opcionales (*optional features*): necesitan ser provistas solo por algunos miembros de la línea de producto, denotadas por el estereotipo de *optional feature*.

Características alternativas *(alternative features*): dos o más características pueden ser alternativas entre ellas, donde sólo una de éstas puede ser provista por algún miembro de la línea de producto, siendo mutuamente excluyentes. Se denotan por el estereotipo de *alternative feature*.

Caracteristicas parametrizadas (parameterized features): es una característica que define un parámetro de una línea de producto cuyo valor necesita ser definido en el sistema.

Los modelos de características pueden ser formados con UML bajo distintos enfoques:

a. Como caso de uso: puede usarse cuando una característica es modelada como un caso de uso.
b. Como paquete de casos de uso: puede usarse cuando una característica es modelada como un grupo de casos de uso.
c. Como clase: utilizando el modelado UML estático para crear meta-clases, por ejemplo.

En la Figura 11 podemos observar un modelo de características bajo el enfoque de clase.

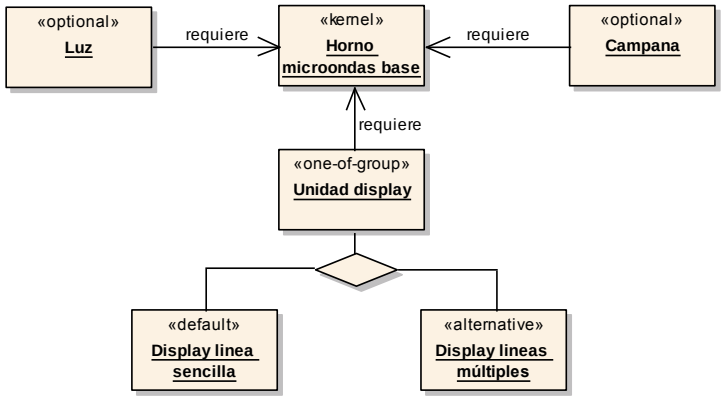


Figura 11. Ejemplo del modelo de características.

3.3.2.2 Fase de Análisis

En esta fase se genera el diseño de la línea de producto de software. El objetivo es identificar el dominio del problema de los objetos así como la comunicación entre ellos; la fase está constituida por dos grandes subfases que son:

1. Modelado Estático
2. Modelado Dinámico

Modelado Estático

El modelado estático tiene como objetivo definir las relaciones que existen entre las clases, mostradas junto con sus relaciones en los diagramas de clases. En esta subfase la comunalidad y variabilidad de los miembros de la línea de producto es dirigida mediante la categorización de las clases como kernel, opcional y variante.

Dentro del modelado estático encontramos los siguientes artefactos: modelo conceptual, modelo de contexto y el modelo de entidades.

Modelo conceptual

El modelo conceptual aborda las relaciones que existen entre las distintas clases. Una clase que es requerida por todos los miembros se enuncia como una clase kernel (*kernel class*); en tanto una clase opcional es la utilizada sólo por ciertos miembros de la familia. Algunas ocasiones existen distintas versiones de una clase que es empleada por diferentes miembros, llamada clase variante.

Los estereotipos utilizados entonces son kernel, optional, y variant. En la Figura 12 encontramos un ejemplo del modelo conceptual.

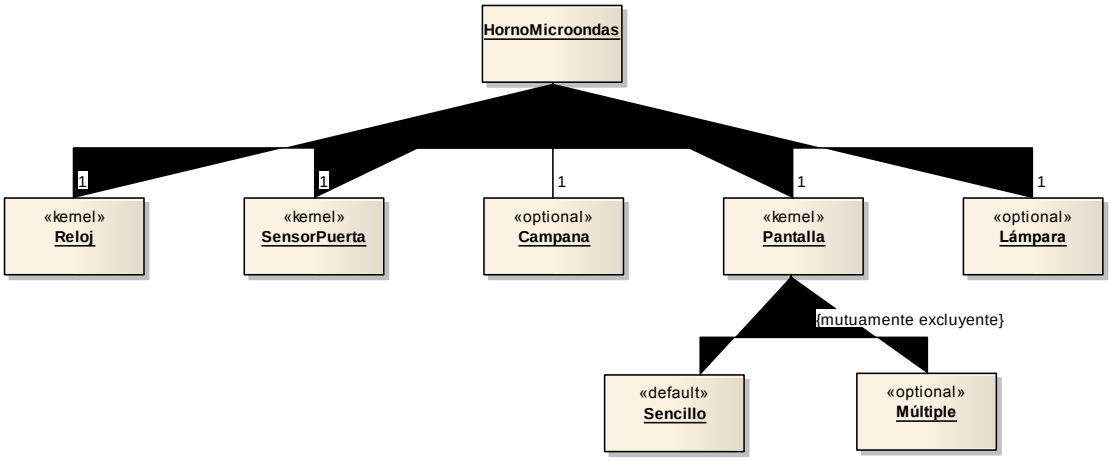


Figura 12. Modelo conceptual

Modelo de contexto

Lo que divide una línea de producto de software y la parte externa se conoce como “Contexto de la línea de producto de software”. Dentro de un análisis estructurado esto es llamado como “Contexto del sistema”.

Cuando el contexto de una línea de producto de software está siendo modelado, se define el límite de hardware/software de un miembro de la familia de productos; esto es conocido como sistema de líneas de producto.

El modelo de contexto de línea de producto utiliza estereotipos a fin de representar los límites de la línea de producto en un diagrama de clases. Las clases externas están categorizadas por estereotipos, clasificadas como *external user*, *external device*, *external system* y *external timer*. El

external device a su vez es clasificado como *external input device*, *external output device*, y *external input/output device*. En la figura 13 podemos observar un modelo de contexto.

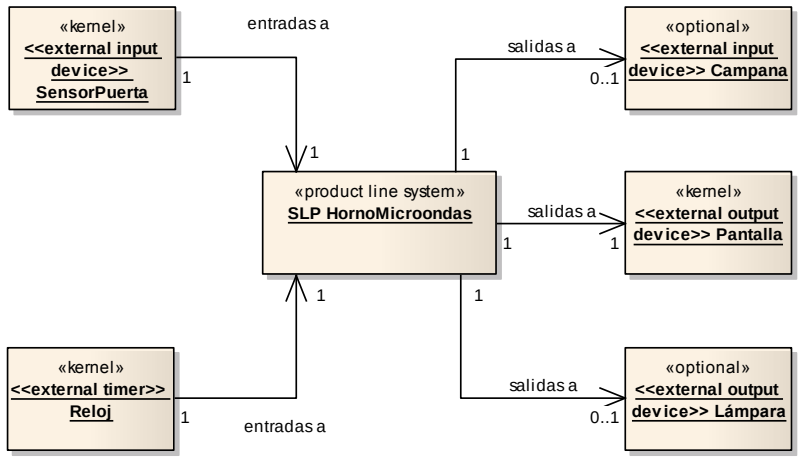


Figura 13. Modelo de contexto.

Modelo de entidades

El modelo de entidades, ilustrado en la figura 14, representa las clases conceptuales de los datos en las que es almacenada la información. Una vez que son integrados los modelos de entidades de los distintos miembros, las clases que son comunes para todos los miembros de la línea de producto se convierten en clases kernel del modelo de entidades. Las clases que se encuentran en una vista pero no en todas se convierten en clases opcionales. Las clases que tienen algunos atributos y funciones en común pero también algunas diferencias, son generalizadas.

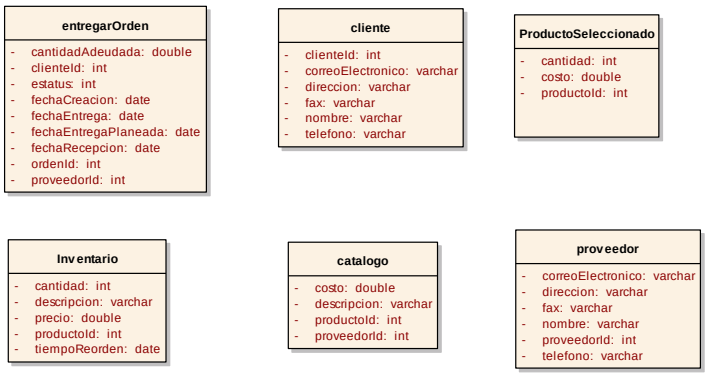


Figura 14. Modelo de entidades.

Modelo dinámico

El modelo dinámico muestra cómo los objetos interactúan durante la ejecución. Las distintas formas en que interactúan se muestran también, de manera que se satisfacen los requerimientos descritos.

Dentro del modelo dinámico se toman como base dos diagramas: el diagrama de comunicaciones y el diagrama de secuencia.

Diagrama de comunicaciones

El diagrama de comunicaciones, ilustrado en la figura 15, es desarrollado para cada caso de uso, por lo que solamente los objetos que en éste participan son representados; la secuencia en la que interactúan es descrita y representada mediante mensajes numerados.

La descripción del mensaje de secuencia se desarrolla como parte del modelo dinámico y describe cómo los objetos del modelo de análisis participan en cada caso de uso, indicando qué es lo que ocurre cuando un mensaje llega a su destino.

El diagrama de comunicación ayuda a determinar las operaciones de los objetos debido a que la llegada de un mensaje a un objeto, normalmente viene acompañado de la llamada a una operación.

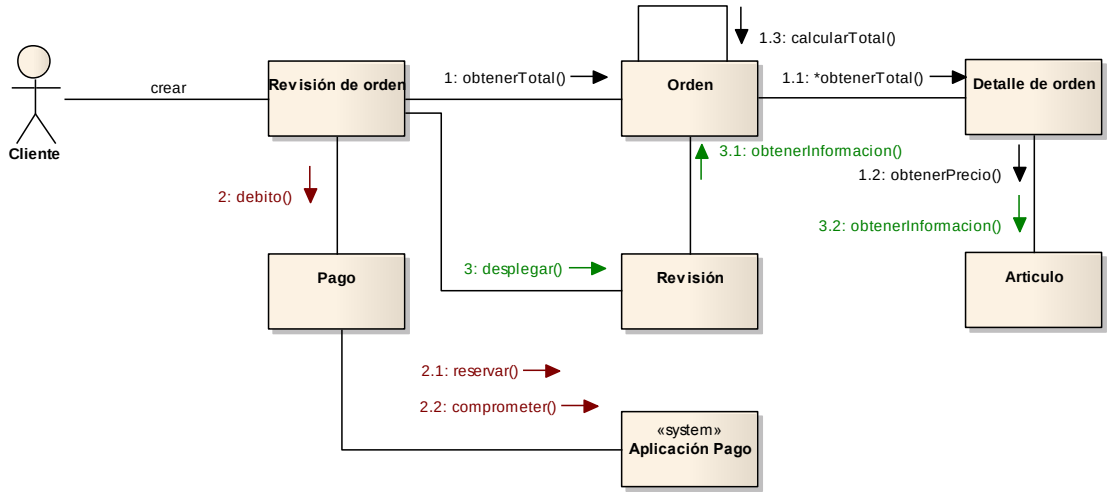


Figura 15. Diagrama de comunicaciones

Diagrama de secuencia

La interacción entre objetos puede ser mostrada en un diagrama de secuencia, pues evidencia cómo se relacionan los objetos durante un lapso en el cual los mensajes son enviados.

Vista del kernel

El análisis dinámico es generado a partir de los casos de uso kernel (que definen el kernel de la línea de producto de software), también conocido como *sistema kernel*, cuyo diagrama de comunicación es conocido como diagrama kernel de comunicación. Solamente los objetos de tipo kernel participan en él.

Para generar la vista del kernel se inicia con el caso de uso tomando en cuenta cada interacción entre el actor principal y el sistema. Posteriormente, se desarrolla el diagrama de comunicación para el escenario descrito en el flujo básico. Por cada interacción hay que seguir los siguientes pasos:

1. Determinar los objetos de interface
2. Determinar los objetos internos
3. Determinar los objetos de comunicación
4. Considerar las secuencias alternativas
5. Considerar los puntos de variación

En la figura 16 podemos observar el modelo de vista del kernel.

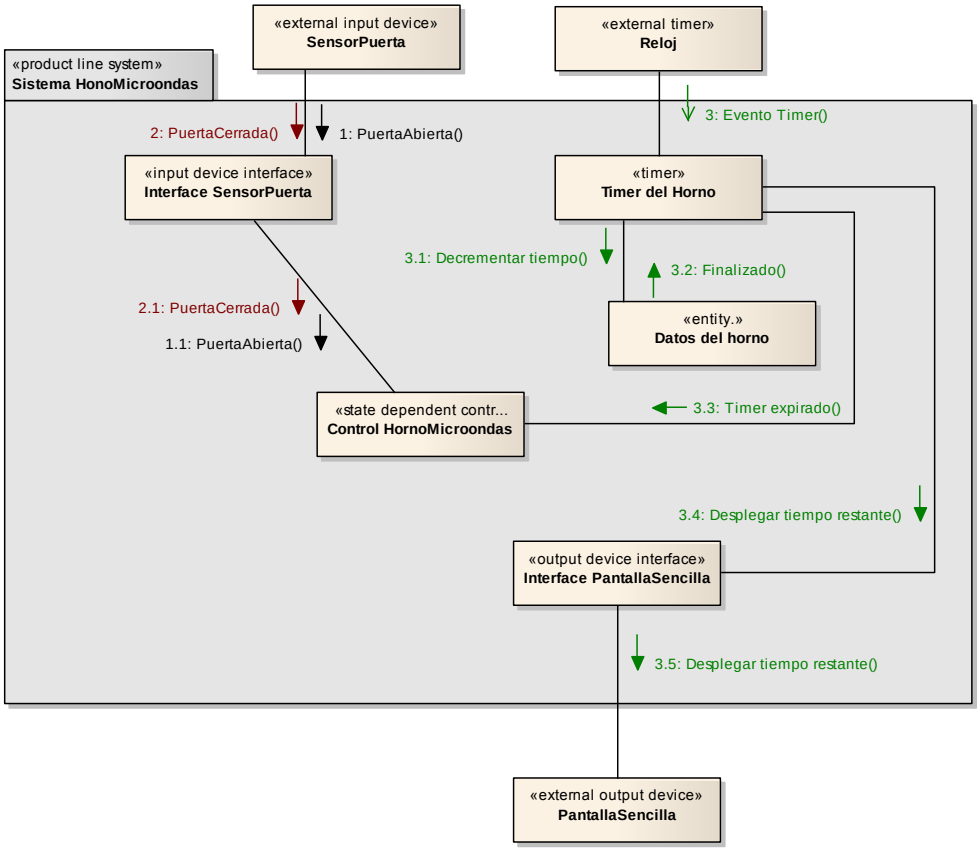


Figura 16. Vista del kernel.

Vista evolucionada

Para poder generar la vista evolucionada es necesario contar con el diagrama de comunicación generado en la vista del kernel. La línea de producto de software se despliega con la inserción de los casos de uso opcionales y alternativos.

Hay dos tipos de evolución que se deben de tener en cuenta. En el primer tipo, uno o más puntos de variación existen en un caso de uso. Bajo esta situación el diagrama de comunicación que debe generarse es una versión modificada del diagrama original, la cual es enunciada como un diagrama de comunicación variante.

En el segundo caso, existen casos de uso que son opcionales y alternativos, para los cuales, se tienen que generar nuevos diagramas de comunicación. En la figura 17 se encuentra ilustrado un ejemplo de la vista evolucionada.

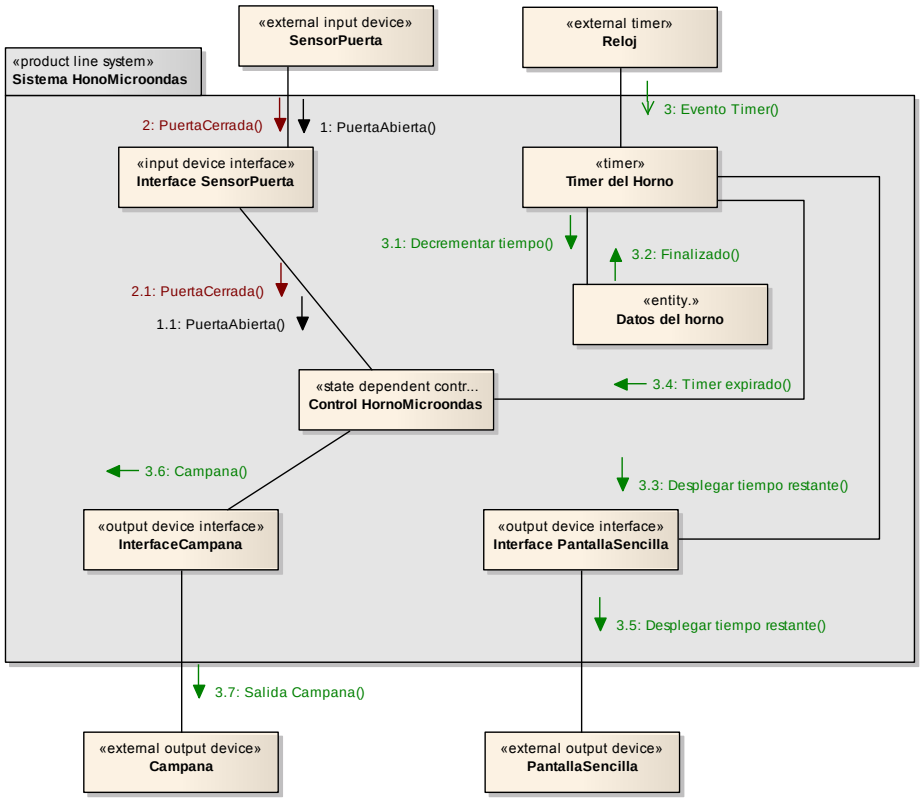


Figura 17. Artefacto generado durante la vista evolucionada.

3.3.2.3 Fase de Diseño

Durante esta fase se modela la arquitectura del sistema tanto para la estructura como para el comportamiento. Podemos apoyarnos en el uso y aplicación de los distintos patrones de software, que proveerán una mejor respuesta al problema encontrado debido a que son soluciones utilizadas previamente.

3.3.2.4 Fase de Implementación de componentes

En ella se lleva a cabo el desarrollo de los componentes identificados a fin de obtener el sistema.

3.3.2.5 Fase de Pruebas

Durante la fase de pruebas es importante llevar a cabo las pruebas de unidad, integración, sistema y aceptación.

Capítulo 4. Un proceso híbrido para el diseño de líneas de producto de software

En este capítulo se describe una propuesta de proceso de desarrollo para generar los artefactos correspondientes al diseño de líneas de producto de software.

4.1 Definición del proceso

Se analizaron los métodos UML Components y PLUS, y se pudo observar que el método propuesto para el proceso de desarrollo por UML Components requiere de un nivel menor de experiencia por parte de los diseñadores de software. Tomando esta ventaja se decide basarse en las fases de este.

Los artefactos proporcionados por UML Components en la fase de requerimientos, tales como modelo de proceso de negocios, modelo conceptual de negocios y casos de uso, permiten ilustrar los requerimientos de un sistema, a estos artefactos se decide extenderlos mediante el uso de estereotipos al igual que se hace en el método PLUS.

Los artefactos proporcionados por PLUS en la fase de análisis, permiten observar el comportamiento del sistema, su interacción con el exterior. Definen claramente la parte común de la línea de producto.

No se consideraron roles de trabajo, debido a que sólo se tiene contemplado el rol de ingeniero de software.

En la siguiente tabla se observa una comparación entre los métodos y se indica cuáles serán los artefactos a generar en nuestro proceso.

Workflow/Fase	Artefacto	UML Components	PLUS	Proceso híbrido
Requerimientos	Modelo de proceso de negocio	Este artefacto se encuentra presente. Permite diseñar un sistema que esté basado en componentes de software.	Este artefacto no existe para el método.	Este artefacto nos permite mapear el flujo del proceso del sistema. Se agregó el uso de estereotipos para permitir el diseño de líneas de productos de software.

Workflow/Fase	Artefacto	UML Components	PLUS	Proceso híbrido
Requerimientos	Modelo conceptual de negocio	Mapa mental del sistema, este artefacto se encuentra presente, permitiendo el diseño de software basado en componentes.	Mapa mental del sistema, utiliza estereotipos para permitir el diseño de líneas de productos de software. Conocido como modelo de características.	Mapa mental del sistema, se definen estereotipos para permitir el diseño de líneas de productos de software.
Requerimientos	Modelo de casos de uso	Diagrama de la funcionalidad del sistema, está presente permitiendo el diseño de software basado en componentes	Diagrama de la funcionalidad del sistema, utiliza estereotipos para permitir el diseño de líneas de productos de software.	Diagrama de la funcionalidad del sistema, se definieron estereotipos para permitir el diseño de líneas de productos de software.
Especificación/ Análisis	Modelo conceptual de negocios refinado	Mapa mental del sistema, acotando a las clases principales. Está presente permitiendo el diseño de software basado en componentes.	No presente.	Mapa mental del sistema, acotando a las clases principales. Se le agregó el uso de estereotipos para permitir el diseño de líneas de productos de software

Workflow/Fase	Artefacto	UML Components	PLUS	Proceso híbrido
Especificación/ Análisis	Modelo de contexto	Diagrama que define las fronteras del sistema. Está presente permitiendo el diseño de software basado en componentes. Conocido como diagrama de interfaces	Diagrama que define las fronteras del sistema. Utiliza estereotipos para permitir el diseño de líneas de productos de software.	Diagrama que define las fronteras del sistema, la interacción con sistemas externos.
Especificación/ Análisis	Modelo de entidades	No presente	Mapa de entidades de datos del sistema. Utiliza estereotipos para permitir el diseño de líneas de productos de software.	Mapa de entidades de datos del sistema. Se definieron estereotipos para permitir el diseño de líneas de productos de software.
Especificación/ Análisis	Modelo kernel	No presente	Diagrama de colaboración de la parte común del sistema.	Diagrama de colaboración de la parte común del sistema.
Especificación/ Análisis	Modelo kernel ampliado	No presente	Diagrama de colaboración de la parte variable del sistema.	Diagrama de colaboración de la parte común del sistema.

4.2 Proceso de desarrollo

El proceso de desarrollo de software sugerido está compuesto por dos fases. En este documento dichas fases recibirán el nombre de workflows, representando cada uno un conjunto de actividades para realizar uno o varios artefactos; ellos son:

1. Requerimientos
2. Especificación

En la Figura 18 podemos observar los workflows del proceso de desarrollo.

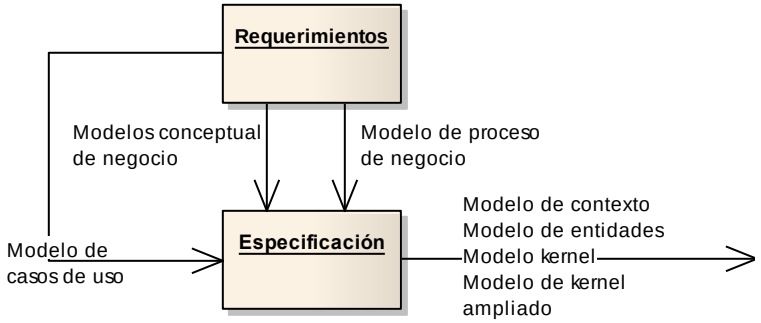


Figura 18. Proceso híbrido de diseño de líneas de producto de software.

Este proceso puede ejecutarse de forma iterativa, de manera que puedan generarse un conjunto de componentes, que posteriormente se unirán y conformarán una línea de producto de software.

En este reporte, estaremos enfocados a los dos primeros workflows, (i.e. requerimientos y especificación), de forma que podamos generar los artefactos necesarios para plasmar los requerimientos, la funcionalidad de la familia de software, la parte común, y las distintas variaciones que pueden llegar a existir.

4.2.1 Requerimientos

Ubicado dentro de los requerimientos de una línea de producto de software, su meta es analizar cada uno de las necesidades de ésta.

Durante el flujo de requerimientos se generan los siguientes artefactos: **modelo de proceso de negocio**, **modelo de conceptos de negocio** y el **modelo de casos de uso**.

A continuación se describen cada uno de ellos.

Modelo de proceso de negocio

En este artefacto obtenemos los flujos básicos y alternativos del sistema, debiendo el primero estar completamente descrito en el modelo. Cada flujo alternativo también deberá de incluirse dentro del modelo, de forma que cada una de las variaciones sea mostrada dentro del flujo.

Existen distintos tipos de flujo que se clasifican de la siguiente manera:

a. Flujo base: es utilizado por todos los miembros de la familia de software. Este flujo se denota con el estereotipo “base”.
b. Flujo opcional: empleado por algunos miembros de la familia de software. Para identificar este flujo se utiliza el estereotipo “opcional”.
c. Flujo alternativo: se presenta cuando dos o más características pueden ser alternativas entre ellas, pudiendo solo una ser provista por algún miembro de la línea de producto, manteniéndose como mutuamente excluyentes. El estereotipo a utilizar para este flujo es “alternativo”.

En la Figura 19 podemos observar un modelo de proceso de negocio.

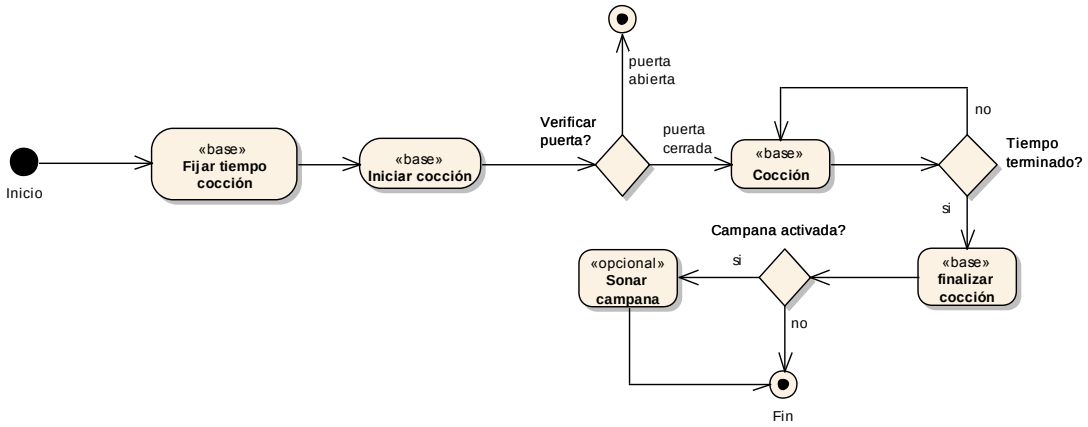


Figura 19. Modelo de proceso de negocio extendido.

Modelo conceptual de negocio

En este artefacto –ilustrado en la Figura 20– obtenemos un mapa mental de cómo se relaciona cada una de las características, de modo que podamos mostrar la parte variable y común del software. Existen distintos tipos de características que a continuación se muestran.

Características comunes: son aquellas que deben de ser provistas por cada uno de los miembros de la línea de producto. Normalmente son denominadas como necesarias, obligatorias o kernel. Una característica común es denotada con el estereotipo de “base”.

Características opcionales (*optional features*): son aquellas que necesitan ser provistas solo por algunos miembros de la línea de producto. Son denotadas por el estereotipo de “opcional”.

Características alternativas (*alternative features*): dos o más características pueden ser alternativas entre ellas, y sólo una provista por algún miembro de la línea de producto, manteniéndose mutuamente excluyentes. Se denotan por el estereotipo de “alternativo”.

Caracteristicas parametrizadas (*parameterized features*): es una característica que define un parámetro de una línea de producto cuyo valor necesita ser definido en el sistema. En la Figura 17 se encuentra ilustrado un modelo conceptual de negocio.

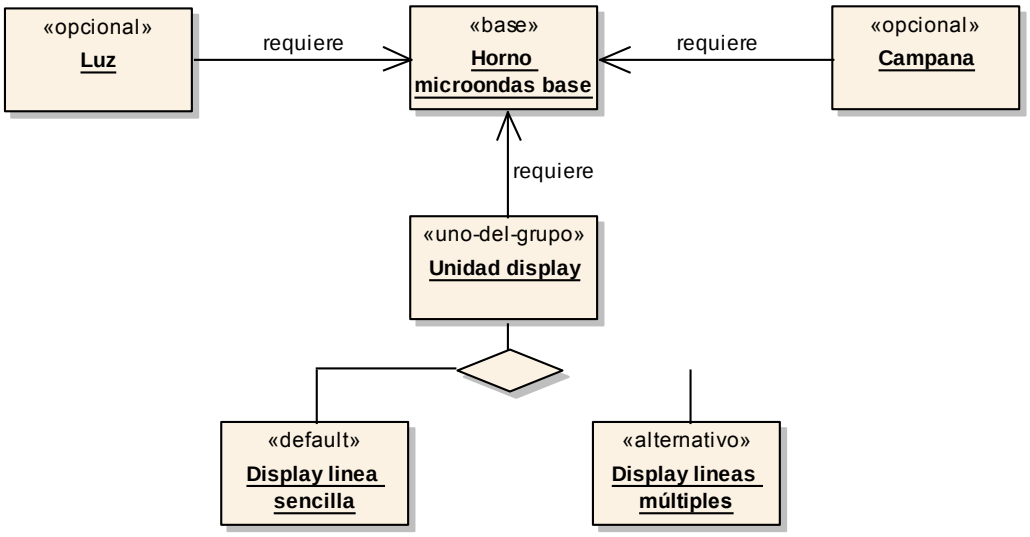


Figura 20. Modelo conceptual de negocio.

Modelo de casos de uso

El modelo de casos de uso, podemos observarlo en la Figura 21, define los requerimientos funcionales de la línea de producto de software en términos de actores y casos de uso, así como las relaciones existentes entre éstos. Con los estereotipos definidos se enriquece el modelo de casos de uso, permitiendo de modelar la comunalidad y la variabilidad dentro de cada caso de uso.

a. Casos de uso base: son requeridos por todos los miembros de la línea de producto de software. Se representan con el estereotipo “base”.
b. Casos de uso opcionales: son aquellos casos de uso que no requieren todos los miembros de la línea de producto de software. Se representan con el estereotipo “opcional”.
c. Casos de uso alternativos: caracterizados por ser distintas versiones requeridas a través de diferentes miembros de la línea de producto de software, los casos de uso alternativos son normal y mutuamente excluyentes. Se recomienda modelarlos como una relación de extensión, de forma que nos permita visualizar las distintas opciones que existen para esos casos de uso. Se representan con el estereotipo “alternativo”.

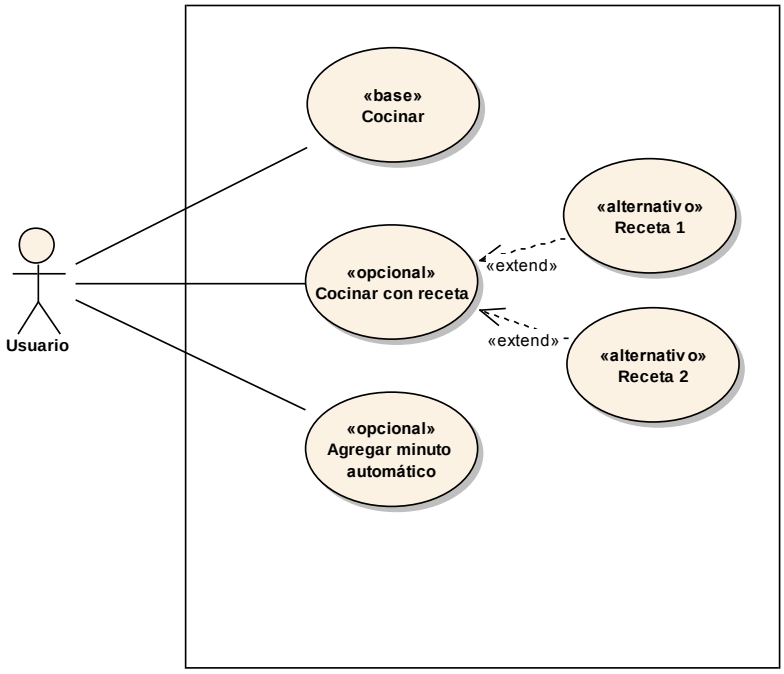


Figura 21. Modelo de casos de uso extendido.

4.2.2 Especificación

El workflow de especificación lo podemos situar dentro del análisis, ya que tiene por objetivo identificar el dominio del problema así como la comunicación entre los objetos.

Este workflow está dividido en 2 subfases que son:

1. Estático
2. Dinámico

4.2.2.1 Estático

Define las relaciones que existen entre las clases y los límites de la línea de producto de software. En esta etapa se generan los siguientes artefactos, el modelo conceptual de negocios refinado, el modelo de contexto, y el modelo de entidades. A continuación se describe cada uno de los artefactos.

Modelo de conceptual de negocios refinado

Se toma como base el modelo de conceptos de negocios generado en el workflow de requerimientos, eliminando y/o agregando clases de manera que nuestro enfoque sea el adecuado. Este modelo refinado nos permitirá visualizar las interfaces existentes entre los diferentes componentes y sistemas; es posible envolver dentro de nuestro modelo algunas restricciones básicas que acoten la relación existente.

En la Figura 22 podemos observar un modelo conceptual de negocios refinado.

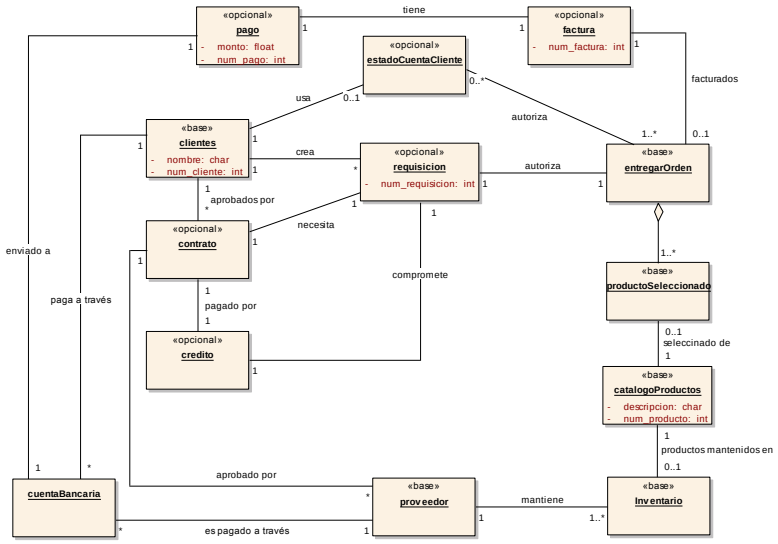


Figura 22. Modelo conceptual de negocios refinado.

Modelo de contexto

Este artefacto utiliza distintos estereotipos a fin de representar los límites de la línea de producto de software. Un miembro externo a la línea de producto de software se concibe como una clase externa con la cual el sistema debe de tener una interface.

Las clases externas se categorizan por estereotipo. Son clasificadas como “usuario externo”, “dispositivo externo”, “sistema externo”, y “contador externo”.

El modelo de contexto nos permitirá observar las distintas interfaces que la línea de producto de software deberá de establecer con las distintas clases externas. En la Figura 23 queda un ejemplo de las clases externas de una línea de producto de software.

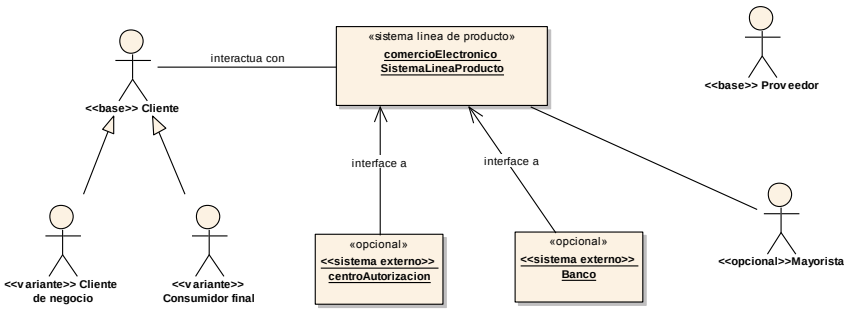


Figura 23. Modelo de contexto.

Modelo de entidades

El artefacto está basado en un diagrama de clases, donde debemos de definir las entidades del sistema, los atributos de cada entidad y sus relaciones. Las operaciones aquí no toman importancia debido a que son actividades propias del diseño de la línea de producto de software.

El modelo de entidades permite visualizar cómo formarán los datos que utilizará el sistema, y cada clase que interactúa en la línea de producto de software indica sus atributos, de forma que este artefacto puede ser mapeado directamente con la base de datos. En el modelo de entidades encontraremos la clase conceptual de los datos, almacenados por la línea de producto de software.

En la Figura 24 hay un modelo de entidades en el cual se observan las distintas clases y sus atributos de la línea de producto de software

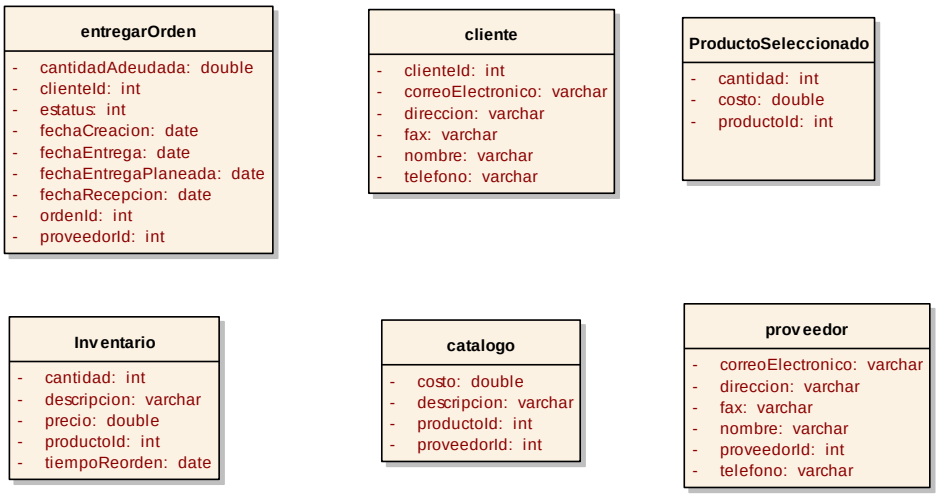


Figura 24. Modelo de entidades.

4.2.2.2 Dinámico

La subfase dinámico tiene por propósito mostrar cómo los objetos interactúan a fin de ejecutar el caso de uso. Durante esta subfase, la etapa de dinámico, se generan artefactos en los cuales es considerada la línea de tiempo. Estos artefactos nos permiten tener en cuenta la parte del control y la secuencia de un objeto u objetos. Es la etapa en que se desarrollan dos artefactos basados en el diagrama de comunicación y el diagrama de secuencia, los cuales son el modelo de kernel y el de kernel ampliado.

Modelo kernel

Para el desarrollo de este artefacto es necesario analizar el modelo de casos de uso generado en el workflow de requerimientos, analizando los que están marcados por el estereotipo kernel para así obtener el kernel de la línea de producto.

El modelo kernel es el miembro mínimo de la línea de producto de software pero en otros casos, algunos objetos que no están categorizados como tipo kernel son necesarios. Esto ocurre cuando existen casos de uso alternativos donde uno de ellos tiene que ser seleccionado.

En caso de existir un punto de variación, el modelo requerirá de otros objetos debido a que puede precisar de la inclusión de un objeto alternativo. Para construir el artefacto es necesario elaborar el diagrama de comunicación tomando como base el flujo básico del caso de uso.

En la Figura 25 podemos observar un ejemplo del modelo kernel.

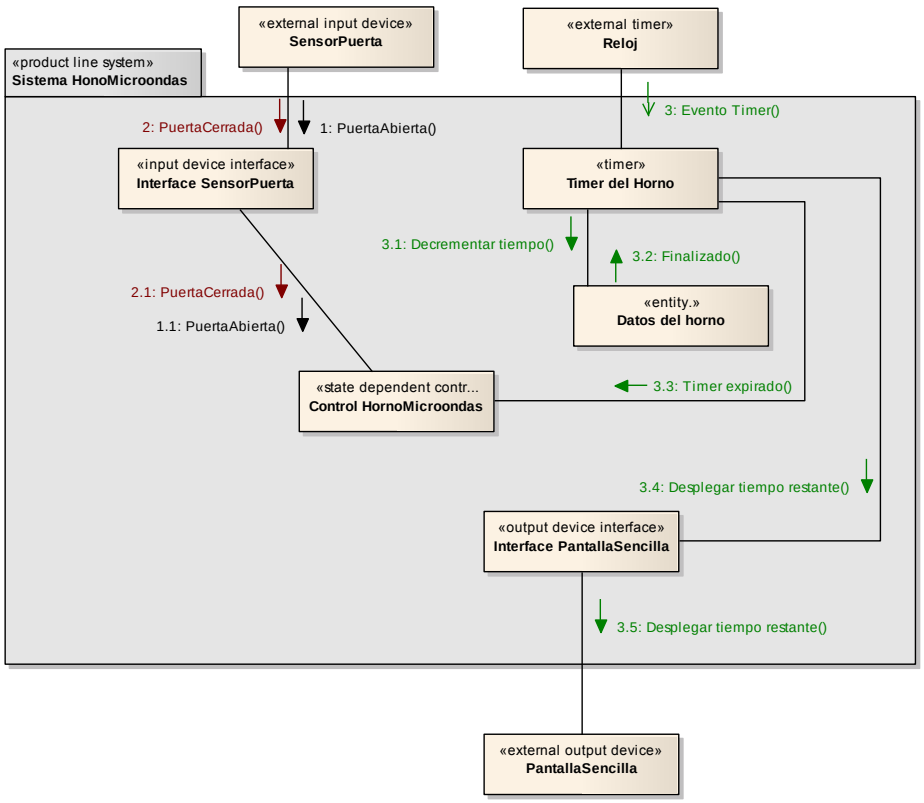


Figura 25. Modelo kernel

Modelo kernel ampliado

El modelo kernel evoluciona al modelo de kernel ampliado, se observa en la Figura 26, con la inclusión de los casos de uso opcionales y alternativos.

La evolución puede darse en dos sentidos: en el primer tema, uno o más puntos de variación existen dentro de un caso de uso. Bajo esta circunstancia el diagrama de comunicación que se desarrollará será una modificación del modelo kernel.

En el segundo, existen casos de uso opcionales y alternativos que son externos; para ellos se requiere generar un nuevo diagrama de comunicación y obtener nuestro artefacto. En la Figura 26, un ejemplo del modelo kernel ampliado.

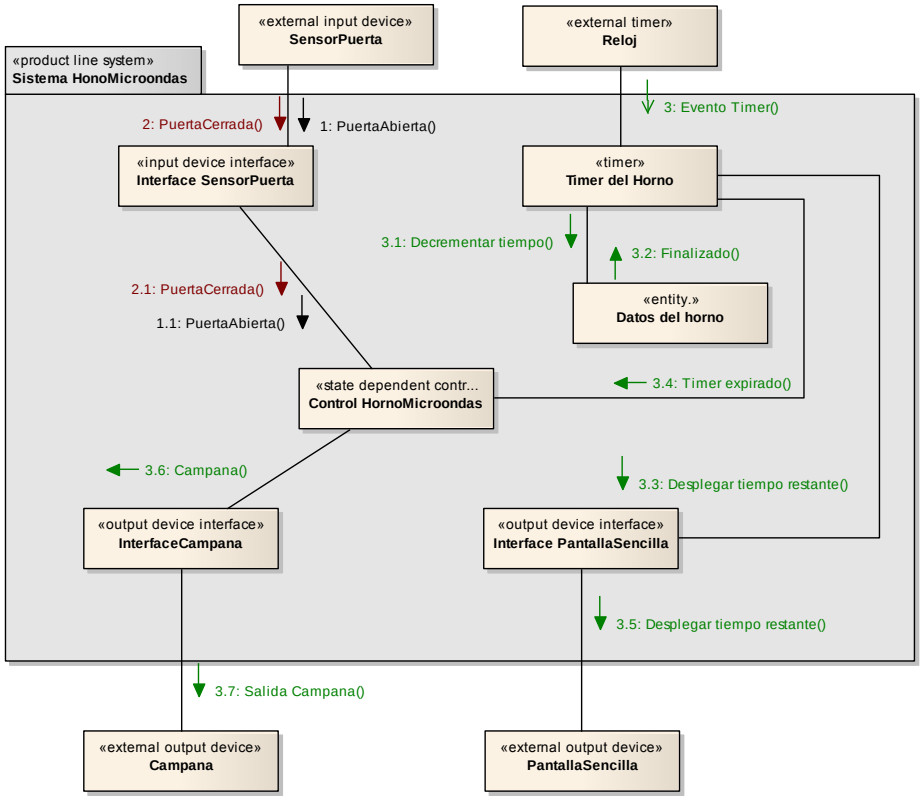


Figura 26. Modelo kernel ampliado.

Capítulo 5. Caso de estudio: Comercio electrónico con líneas de producto de software

En este capítulo encontraremos un caso de estudio enfocado a un sistema transaccional, con el objeto de mostrar que las líneas de producto de software no solamente son adaptables al dominio de las aplicaciones de control.

5.1 Descripción

En los sistemas de comercio electrónico business-to-business (B2B), existen clientes y proveedores. Cada cliente tiene un contrato con el proveedor para las compras, así como una o más cuentas bancarias a través de las que los pagos, pueden ser hechos a los proveedores. Cada proveedor puede proporcionar un catalogo de artículos, acepta ordenes de los clientes y mantiene el estado de cuenta con cada cliente por pago recibido.

Un cliente puede buscar a través de distintos catálogos en la Internet —provistos por los proveedores— y seleccionar los productos para la compra. La orden necesita ser revisada para verificar que exista un contrato disponible a fin de determinar si es válido con el proveedor, pues será utilizado para generar el cargo de la compra. Cada contrato tiene indicados el monto límite de las operaciones. Esto es necesario para determinar si existen los fondos suficientes y generar la orden. Asumiendo que el contrato es válido, una orden de surtido es creada y enviada al proveedor, quien confirma la orden y programa la fecha de entrega. Conforme el tiempo avanza, la orden de envío es monitoreada y en caso de existir un retraso se notifica tanto al proveedor como al cliente; una vez embarcada el cliente recibe una notificación. El cliente indica cuando la orden es recibida y la orden de entrega es actualizada. Después de recibir el envío, el pago de la factura es autorizado. La factura se verifica contra el contrato, fondos disponibles y estatus de orden de entrega, para ser posteriormente enviada a cuentas por pagar, que autoriza el pago de los fondos. El pago se realiza a través de un traspaso electrónico desde el banco del cliente al banco del proveedor. El sistema utiliza distintas bases de datos legadas, lo que hace necesarias tecnologías que permitan generar la conexión.

Opcionalmente, un proveedor puede crear una orden de compra requiriendo nueva materia prima a un mayorista. La orden de compra es enviada directamente al mayorista. Cuando es surtida al proveedor, la mercancía ingresa a un almacén de inventario del proveedor, y el pago es generado vía traspaso electrónico desde el banco del proveedor al banco del mayorista.

En el sistema de comercio electrónico B2C, el cliente solicita la compra de uno o más artículos de un proveedor. El cliente proporciona sus datos personales como dirección e información de tarjeta de crédito. La información es almacenada en una cuenta del cliente. El proveedor confirma la orden y registra la fecha de entrega. Cuando la orden es enviada, el cliente es notificado y el cargo es aplicado en la tarjeta de crédito.

5.2 Requerimientos

En esta sección mostraremos los artefactos que se deben generar para este workflow, encontrando el **modelo de proceso de negocios**, el **modelo de conceptos de negocio** y el **modelo de casos de uso**.

5.2.1 Modelo de proceso de negocios

Con el modelo de proceso de negocios podemos identificar el flujo que sigue el negocio. Se detectan los componentes que forman parte del flujo básico y aquellos que forman los flujos alternativos. Esto permite comenzar a identificar la comunalidad y la variabilidad de la línea de producto de software. En la figura 27, el modelo de proceso de negocio del caso de estudio.

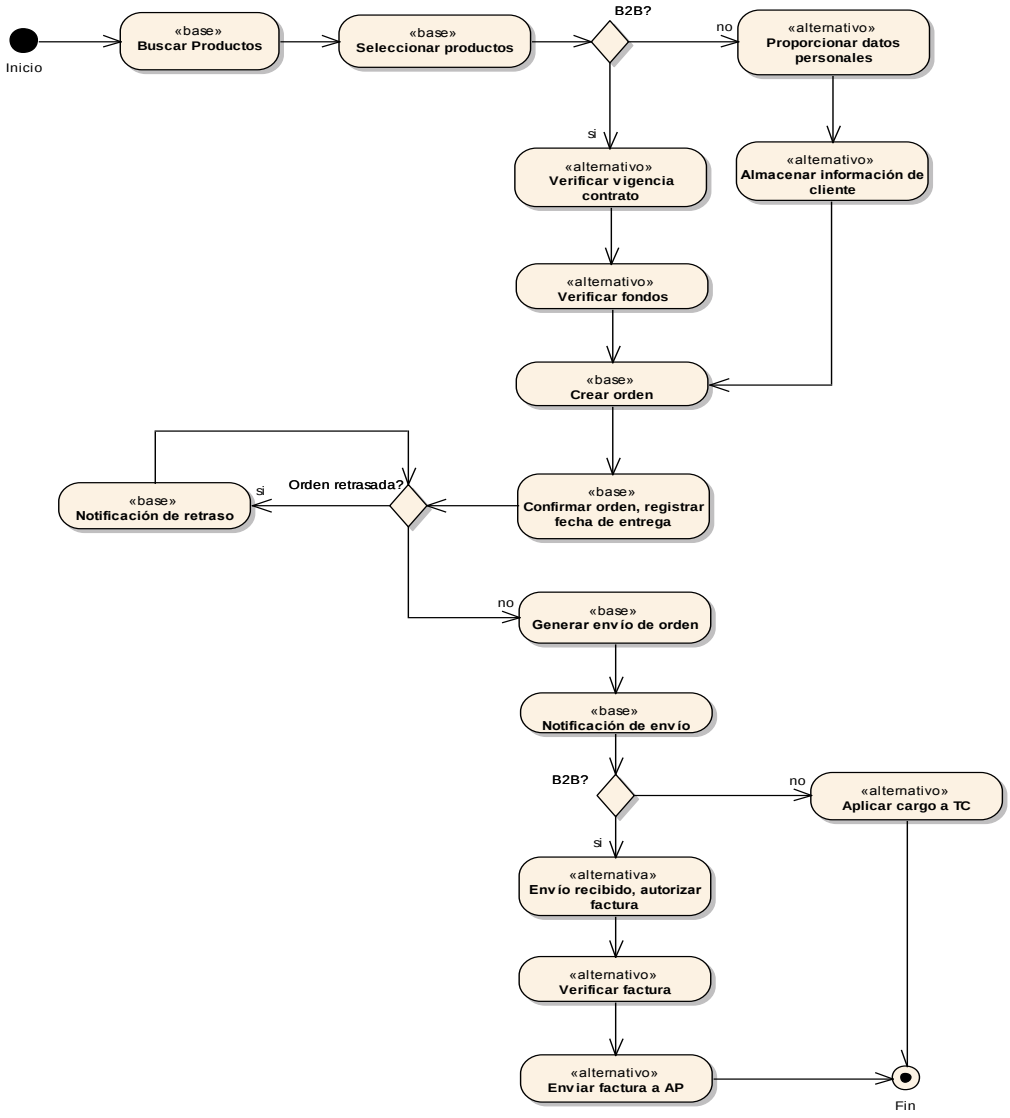


Figura 27. Modelo de proceso de negocios de e-commerce.

En nuestro caso de estudio existen dos distintos sistemas a considerar, el B2B y el B2C, con distintos procesos de negocio, y un mismo objetivo: proveer a un tercero un producto del catálogo de comidas. Se visualizan claramente algunos de los componentes que son comunes en ambos miembros de la línea de producto de software, y los que son variables.

Componentes como la búsqueda de productos y seleccionar productos forman parte del flujo base, mientras que componentes como autorizar factura, es parte de un flujo alternativo.

5.2.2 Modelo conceptual de negocio

En el modelo conceptual de negocio los requerimientos se revisarán desde la perspectiva de reutilización.

El modelo conceptual de negocios —mostrado en la figura 28—, muestra clases marcadas como "base" u "opcionales". Las clases como cliente o catalogoProductos se convierten en base debido a que existen en ambos miembros de la línea de producto de software, (i.e B2B y B2C). Las clases que sólo aparecen en una de las dos aplicaciones, tal como contrato o estadoCuentaCliente se convierten en opcionales. Por último aquellas clases que pueden o no estar en alguno de las dos aplicaciones se convierten en opcionales.

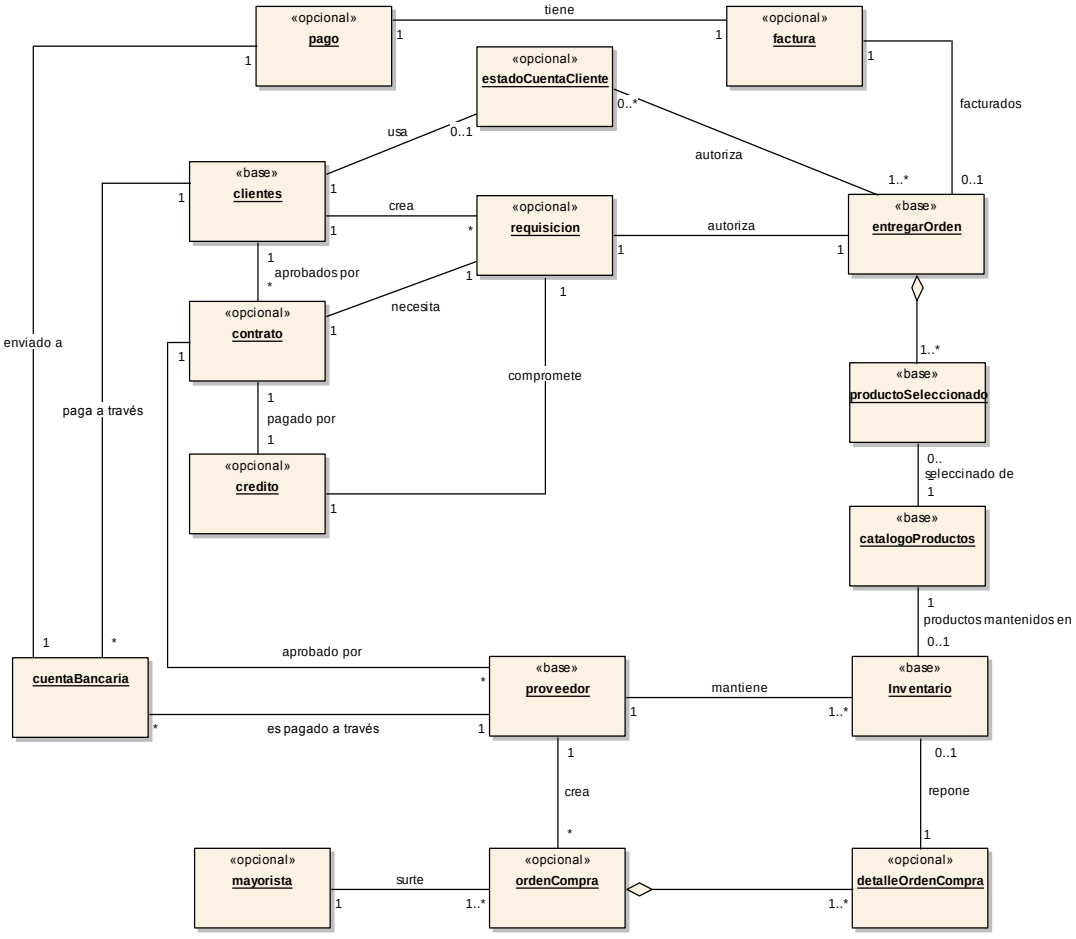


Figura 28. Modelo de conceptos de negocio.

5.2.3 Modelo de casos de uso

Debido a que existen dos sistemas en nuestro caso de estudio es necesario desarrollar el modelo de casos de uso para cada sistema, desde donde podremos observar la comunalidad de la línea de producto de software y posteriormente, la variabilidad de la misma.

En la Figura 29 se encuentra el modelo de casos de uso de B2B. Existen tres actores que son cliente, proveedor y banco. El cliente se relaciona con cuatro casos de uso: buscar producto, crear requisición, crear compra y confirmar entrega. El proveedor se relaciona con tres casos de uso: procesar entrega de orden, confirmar envío y enviar factura.

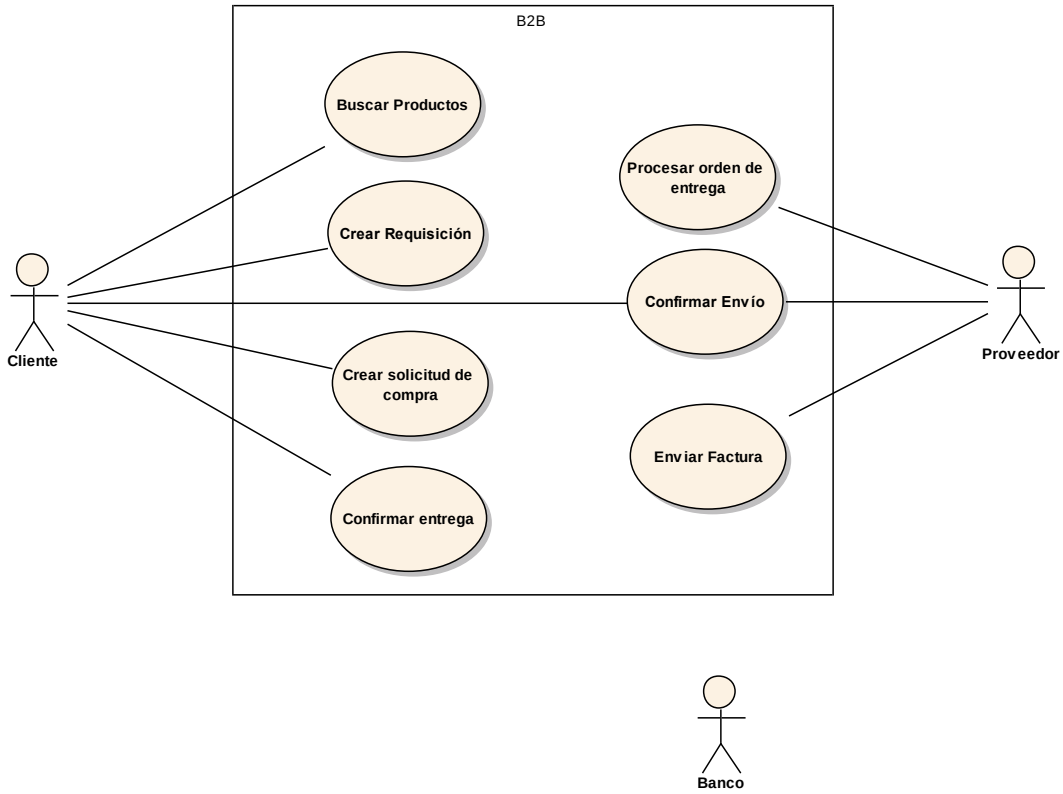


Figura 29. Modelo de casos de uso del sistema B2B.

El caso de uso buscar catálogos permite al cliente buscar en los catálogos de productos que existen en la Internet y seleccionar productos del catálogo. En el caso de uso crear requisición, el cliente genera la solicitud de requisición. El sistema tiene que encontrar el contrato del cliente con el proveedor para el cual tenga los fondos suficientes y efectuar la operación. Si el contrato es válido, el sistema autoriza la requisición y notifica al cliente. En crear solicitud de compra, el cliente le cuestiona al sistema para enviar la solicitud de compra al proveedor.

En procesar orden de entrega, el proveedor solicita una orden de entrega, determina si se cuenta con la materia prima necesaria para procesar la orden y la muestra. En el caso de uso confirmar envío, el proveedor prepara el envío y posteriormente lo confirma. En el caso de uso de confirmar entrega, cuando el envío llega con el cliente, el cliente confirma la entrega. El monto de la

operación es reservado para el pago. En el caso de uso enviar factura, el proveedor envía la factura a la organización del cliente. Una vez que se confirma la entrega la factura es aprobada por el departamento de cuentas por pagar y una solicitud de pago es enviada.

En la Figura 30 podemos observar el modelo de casos de uso para el sistema B2B.

Figura 30. Modelo de casos de uso del sistema B2C.

En este modelo de casos de uso encontrarnos tres actores: cliente, proveedor y centro de autorización. El cliente se relaciona con buscar productos, verificar cuenta cliente y crear solicitud de compra. El proveedor se relaciona con procesar entrega de orden, confirmar envío y facturar.

Los casos de uso buscar catálogo, crear solicitud de compra, procesar entrega de orden y confirmar envío son idénticos a los casos de uso del sistema B2B. El caso de uso verificar cuenta cliente, el cliente ingresa sus datos personales. El sistema crea una cuenta de cliente si el cliente no existe. La tarjeta de crédito del cliente es validada en cuanto a su autenticidad y crédito disponible para generar el pago de los productos solicitados. Si la validación de la tarjeta de crédito resulta satisfactoria la compra del cliente es aprobada. En el caso de uso de aplicar cargo a TC, el sistema obtiene la información de la tarjeta de crédito del cliente y envía al centro de autorización la aplicación del cargo en la tarjeta de crédito.

Aplicando ingeniería inversa, los casos de uso de los distintos modelos son integrados para producir un caso de uso de la línea de producto de software. Los casos de uso buscar catálogo y crear solicitud de compra son comunes para todos los sistemas lo que los convierte en casos de uso base. De forma similar los casos de uso procesar entrega de orden y confirmar envío son comunes para todos los miembros convirtiéndose en casos de uso base.

Por otra parte, los casos de uso crear requisición, confirmar envío, verificar cuenta cliente, enviar factura y aplicar cargo a TC se convierten en alternativos.

Por último los casos de uso de la funcionalidad opcional de las órdenes de compra (preparar orden de compra y entregar orden compra) son casos de uso que se convierten en opcionales.

En la Figura 31, el modelo de casos de uso de la línea de producto de software.

Figura 31. Modelo de caso de usos de la línea de producto de software.

5.3 Especificación

En esta sección mostraremos los artefactos que deben de generarse para este workflow, que se divide en las subfases de estático y dinámico, que a continuación se mostrarán.

5.3.1 Estático

En esta sección se describirán los artefactos que nos muestren las relaciones existentes entre las clases así como los límites del sistema. Los artefactos a generar durante esta subfase son el **modelo conceptual de negocios refinado**, **modelo de contexto** y **modelo de entidades**.

Modelo conceptual de negocios refinado

En este artefacto tomaremos como base el modelo conceptual de negocios creado durante el workflow de requerimientos. Durante el refinamiento eliminamos las clases de la funcionalidad opcional para generar órdenes de compra, y se agregaron a las clases principales los atributos más importantes.

En la Figura 32, el modelo de tipos de negocio.

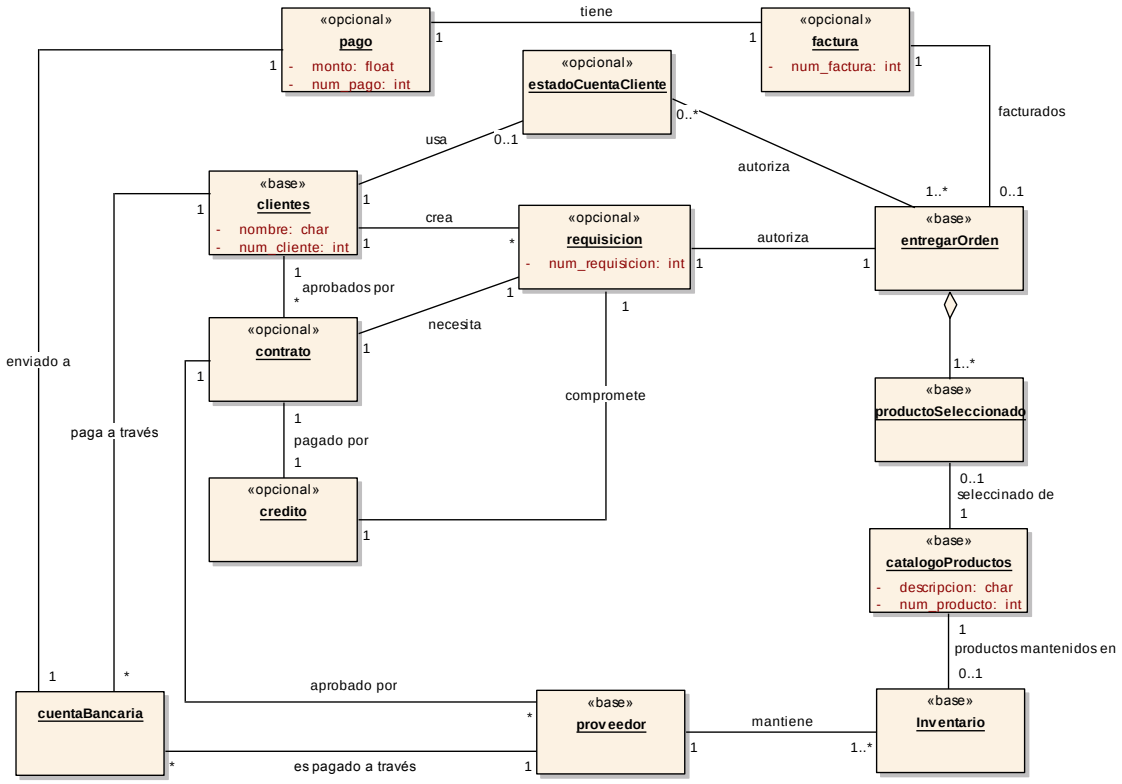


Figura 32. Modelo de tipos de negocio refinado.

Modelo de contexto

El modelo de contexto nos muestra tres clases externas de usuario, representadas como actores: las clases de tipo base *cliente* y *proveedor* y la clase opcional *mayorista.* El actor cliente está especializado en dos actores variantes, cliente de negocio y consumidor final. Existen dos clases opcionales externas de sistema: centro de autorización y banco.

En la Figura 33, descrito el modelo de contexto.

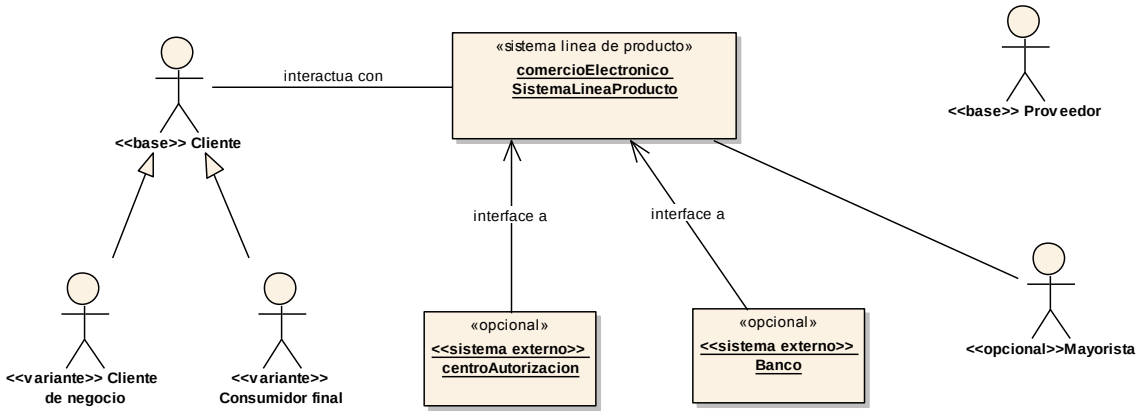


Figura 33. Modelo de contexto.

Modelo de entidades

En el modelo de entidades mostramos las clases conceptuales que formarán los datos en nuestra base de datos. Los atributos de las clases deben de ser definidos. En la Figura 34 se encuentra el modelo de entidades para las clases de tipo base y en la Figura 35, el modelo de entidades para las clases de tipo opcional.

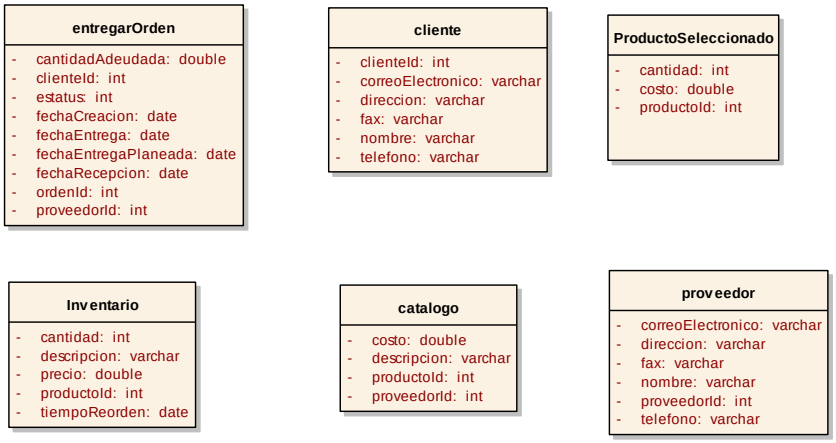


Figura 34. Modelo de entidades de las clases bases.

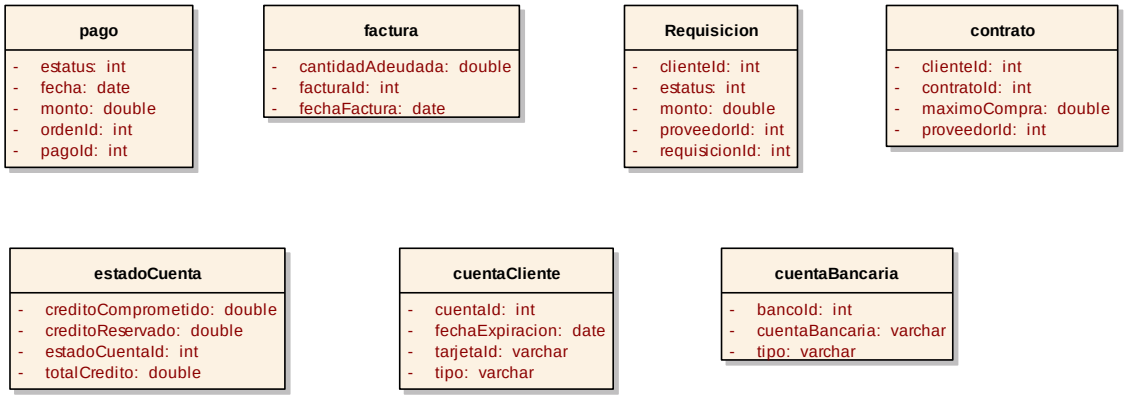


Figura 35. Modelo de entidades de las clases opcionales.

5.3.2 Dinámico

En esta subfase se debe generar por cada uno de los casos de uso el diagrama de comunicación a fin de generar el modelo kernel y el modelo kernel ampliado. A continuación se detallarán estos artefactos para el caso de uso buscar catálogo. En el modelo se podrán observar los objetos que interactúan y los mensajes que se pasan entre ellos.

Modelo kernel

El modelo kernel está compuesto de los diagramas de comunicación de cada uso del sistema que están estereotipados como base. Para el caso de estudio sólo mostraremos el caso de uso buscar productos.

Para el caso de uso buscar productos encontramos que la interface cliente interactúa con el agente cliente a fin de comunicarse con el servidor de catálogos. En la figura 36 podemos observar el diagrama de comunicación del caso de uso buscar productos.

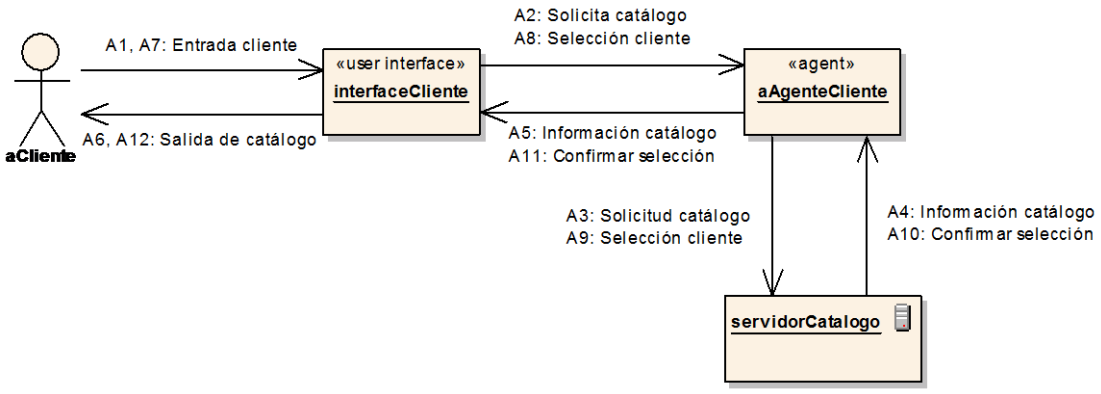


Figura 36. Diagrama de comunicación del caso de uso buscar productos.

Modelo kernel ampliado

El modelo kernel ampliado está compuesto por los diagramas de colaboración de los casos de uso que están estereotipados como opcionales y alternativos. Para el caso de estudio únicamente se mostrará el caso de uso confirmar entrega.

El caso de uso confirmar entrega se ejecuta cuando el envío es recibido por el cliente, y éste notifica la recepción de los bienes. La orden de envío entonces se actualiza con la fecha de recepción. El agente de recepción es también notificado. En la Figura 37 se encuentra ilustrado el diagrama de comunicación para el caso de uso confirmar entrega.

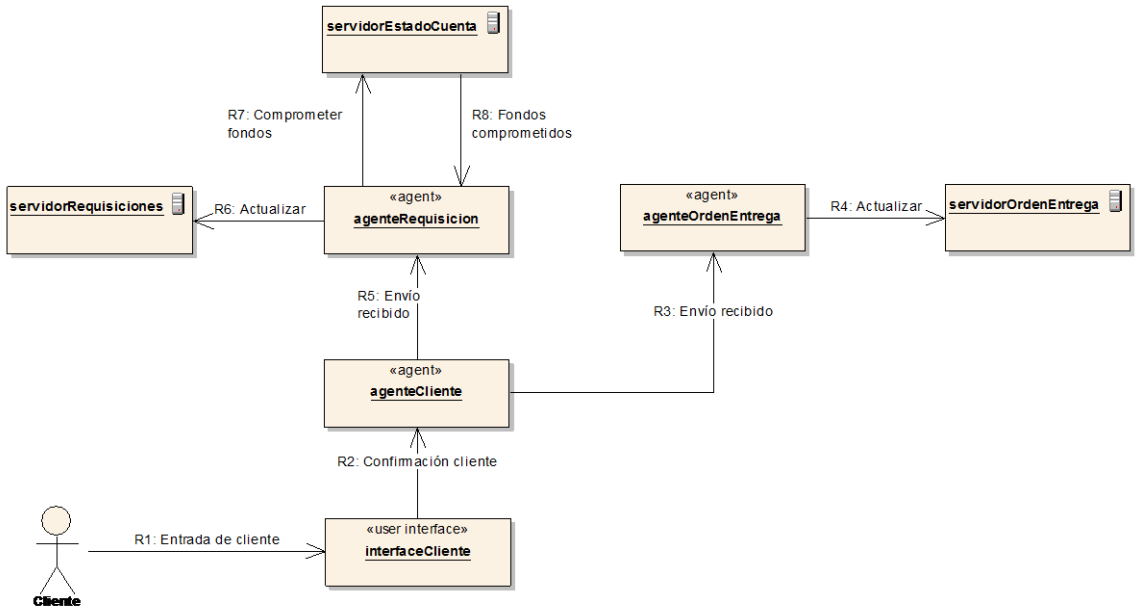


Figura 37. Diagrama de comunicación del caso de uso confirmar entrega.

Capítulo 6. Evaluación

En este capítulo se realizará una evaluación del proceso propuesto, siendo necesario indicar si durante la realización del caso de estudio se logró resolver la problemática que motivó su desarrollo.

6.1 Descripción

La evaluación consistirá en evaluar las ventajas y desventajas del proceso híbrido con respecto a los procesos descritos en los métodos UML Components y PLUS.

6.2 Ventajas

- Diseño prescriptivo de líneas de producto de software.

  Como pudimos observar en el capítulo 4 el proceso propuesto nos genera un detalle de pasos para la generación de artefactos de entrada y salida de cada fase, de igual manera nos presenta la evolución de estos durante el proceso.

- Basado en una semántica y sintaxis bien conocida como UML.

  UML es un lenguaje de modelado ampliamente conocido y de fácil adopción en la práctica. Nos permite generar diagramas que especifican los requerimientos de una manera simplificada. De esta forma, consideramos el hecho que nuestra propuesta incorpore el uso de UML como sintaxis principal es una ventaja representativa.

- Proceso enriquecido.

  Como pudimos observar en el capítulo 4 el proceso propuesto permite generar artefactos de una forma detallada, disminuyendo la complejidad propia de PLUS. Por otro lado, nos permite el manejo de líneas de producto de software, mejorando en esta parte lo que presenta UML Components.

- Manejo explícito de la parte común de la línea de producto de software.

  Con el uso de los diferentes estereotipos, nos permite dilucidar la parte común y variable de la línea de producto de software. Esto se observó en las secciones 4.2.1 y 4.2.2

- Manejo explícito de la parte variable de la línea de producto de software.

  Con el uso de los diferentes estereotipos, nos permite definir la parte variable de la línea de producto de software. Esto se observó en las secciones 4.2.1 y 4.2.2

- Diseño incremental.

  Como pudimos observar en el capítulo 4 en el proceso propuesto existen permite generar artefactos los cuales irán evolucionando. Estas evoluciones permiten generar artefactos complejos sin necesidad de iniciar con una versión de este de un alto grado de dificultad.

- Modelado del proceso de negocio.

  El modelo del proceso del negocio es un diagrama en el cual se visualiza el flujo(s) que se realizarán en el sistema, indica las actividades principales. En el podemos encontrar el camino básico, que en un futuro nos definirá los componentes que conformarán la parte base del sistema. Por otra parte nos permitirá conocer los flujos opcionales que formarán parte de los puntos de variación del sistema.

- Modelo de tipos de negocio.

  Es un diagrama de clases en el cual se pueden observar las clases más importantes del sistema. Este diagrama permite tener un mapa mental del sistema acotado a las clases de mayor relevancia. En él se indican las clases que forman parte principal de nuestro sistema, los principales atributos y las operaciones de mayor relevancia.

6.3 Desventajas

- Definición de interfaces.

  Una interface es lo que permite la comunicación entre dos o más sistemas o componentes. Es importante conocer los puntos de interacción existentes del sistema con el exterior, así como se debe de generar la implementación de la interface. En nuestro proceso no existe una definición de las interfaces de sistema. Quedando limitado a las interfaces con sistemas externos.

- Roles no definidos.

  Un rol de trabajo nos define las tareas a realizar. En el proceso no se tienen considerados roles de trabajo, debido a que no se tuvo la visión de trabajo en equipo.

- Poca aplicabilidad en equipos de trabajo.

  Al no existir roles, no es posible aplicar el proceso dentro de un equipo de trabajo. En los equipos de trabajo encontramos tareas específicas para cada uno de los integrantes dependiendo del rol que se desempeñe.

Capítulo 7. Conclusiones y trabajo futuro

En este capítulo se generarán las conclusiones del documento presentado, así como la posible línea de seguimiento para un trabajo futuro.

7.1 Conclusiones

El objetivo principal de desarrollar un proceso híbrido se logró concretar.

El proceso híbrido generó una mezcla de artefactos, que nos permitirán diseñar de una forma sencilla una línea de productos de software.

El proceso híbrido definido, genera una aportación valiosa debido a que nos permite diseñar una línea de producto de software de una forma prescriptiva, al indicar en cada uno de los workflows los artefactos que fungen como entrada y cuales serán generados en el.

La parte medular de las líneas de producto de software, que se basa en la reutilización de una parte común para todos los miembros queda cubierta; no obstante, la parte de las interfaces hacia el exterior no está definida de forma prescriptiva.

Nos permite tener un manejo explícito de la parte común y variable de la línea de producto de software al utilizar estereotipos.

El proceso es fácil de llevar a la práctica al estar basado en UML.

El proceso está diseñado de forma incremental, permitiendo generar artefactos de un forma sencilla que irán evolucionando.

7.2 Trabajo futuro

El presente documento nos presenta tres líneas de trabajo futuro. A continuación se enlistan:

1. Roles de trabajo: no existen roles de trabajo definidos. Sin los roles de trabajo, la aplicación del proceso híbrido en un equipo de trabajo se vuelve complicado.
2. Enriquecer proceso: se enfocó a dos métodos, UML Components y PLUS, existiendo una gama de métodos para el diseño de software basado en componentes y líneas de producto de software. Se puede enriquecer con el análisis de otro método –como KobrA– y agregar algún artefacto que sea utilizado.
3. Definición de interfaces: describir de forma detallada la dilucidación de las interfaces existentes en el sistema.

Referencias

[1] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wust, J., Zettel, J., 2001. Component-Based Product Line Engineering with UML. Addison-Wesley

[2] Gomaa Hassan, 2004. Designing Software Product Lines with UML. Addison-Wesley

[3] Chessman, Daniels John. UML Components a simple process for specifying component-based software. Addison-Wesley

[4] OMG Available Specification, Versión 2.0, May 2006. Object Constraint Language. Object Management Group