UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

LÍNEA DE PRODUCTOS SOFTWARE TRAMITACIÓN DE CAUSAS JUDICIALES

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN TECNOLOGÍAS DE LA INFORMACIÓN

RODRIGO ALEXIS PORTILLA PORTILLA

PROFESOR GUÍA:
ÉRIC TANTER

MIEMBROS DE LA COMISIÓN:
ALEXANDRE BERGEL
ROMAIN ROBBES
ROSA ALARCON CHOQUE

SANTIAGO DE CHILE
AGOSTO 2013

Resumen

El proyecto de tesis “Línea de Productos Software de Tramitación de Causas Judiciales” se enmarca en el contexto laboral de la empresa Amisoft, organización especializada en tecnologías de la información con más de 14 años en el mercado nacional. Durante los últimos años, se ha adjudicado una serie de proyectos en el sector judicial, diseñando e implementando los principales sistemas informáticos de tramitación de causas para el Poder Judicial de Chile y Ministerio de Justicia. Durante el año 2011 la organización decidió capitalizar todo el conocimiento adquirido durante este último tiempo en la creación de un producto propio que implementará las principales actividades de una organización judicial para la tramitación de causas judiciales. Una vez creado el producto, surgieron una serie de inconvenientes, ya que conociendo el negocio y teniendo una experiencia importante en el ámbito, la organización no cuenta con un *core* definido sobre el cual aplicar el desarrollo diferenciado y particular del producto para cada grupo de clientes pertenecientes al sector judicial.

Es así como surgen las líneas de productos software (LPS) que permiten la reutilización sistemática de software similar pero diferenciado por algunas características. El objetivo es sacar el máximo partido de los elementos comunes, y gestionar de una manera eficaz las variaciones. El desarrollo del proyecto de tesis buscar implementar en la organización una línea de productos dentro del dominio de tramitación de causas judiciales, permitiendo contar con una serie de componentes reutilizables y gestionar en forma eficiente la variabilidad de los nuevos productos que son creados para diferentes clientes.

Para la creación de la línea de productos se utiliza la metodología de Desarrollo Dirigido por Características (*Feature-Driven Development*) (1) el cual es un proceso ágil que se basa en construir iteraciones cortas que produzcan incrementos funcionales en el software que los clientes y personas encargadas de la gestión del proyecto puedan ver, analizar y aprobar. Del punto de vista de diseño e implementación se aplica el concepto de combinación de paquetes (“*package merge*”), propuesta de Laguna y Otros (2) que consiste fundamentalmente en añadir detalles de forma incremental y se define como una relación entre dos paquetes que indica que los contenidos de ambos se combinan. Por último, para extender la trazabilidad hasta los modelos de implementación, se utiliza el concepto de clases parciales en C#. La utilización de “mixins” o clases parciales fue propuesta originalmente en el lenguaje Flavors (3) y representa una alternativa a la herencia múltiple y una manera de manejar la variabilidad relacionada con aspectos. La intención es mantener la correspondencia uno a uno no sólo entre características y paquetes de diseño sino también con la estructura del código.

Agradecimientos

A mis hijos y esposa por el apoyo incondicional en todos los desafíos que me he propuesto durante mi vida.

A mi profesor guía Éric Tanter por su enorme apoyo y por darme la oportunidad de trabajar junto a él.

A todos los profesores del Magister en Tecnologías de la Información por su motivación y entrega de sus conocimientos.

Finalmente, a la empresa Amisoft y a sus colaboradores por el apoyo entregado para que esta tesis resultara exitosa.

Tabla de contenido

Índice de figuras

Capítulo I

1 Introducción

1.1 Contexto

El proyecto de tesis se enmarca en el contexto laboral de la empresa Amisoft, empresa especializada en tecnologías de la información con más de 14 años en el mercado. Desde sus comienzos ha participado en una serie de proyectos de gran envergadura donde ha adquirido a lo largo del tiempo la experiencia y *know how* necesarios para enfrentar las problemáticas de alta complejidad que a diario enfrentan sus clientes.

Su línea de negocio principal es el desarrollo de software a la medida, enfocado a empresas de gran tamaño, tanto en el sector público como privado. Durante los últimos años, la organización se ha adjudicado una serie de proyectos en el sector judicial, diseñando e implementando los principales sistemas informáticos de tramitación de causas para el Poder Judicial de Chile y Ministerio de Justicia. Esta experiencia adquirida durante el último tiempo, posiciona a la organización en el nicho de mercado relacionado con la administración judicial, convirtiéndose en un referente de este sector a nivel nacional.

Al ser una organización referente, propone el desafío de crear productos más eficientes y de mayor calidad, por lo tanto, también se necesita lograr eficacia en el proceso de desarrollo y comercialización del producto. Al hablar de la eficacia en el proceso de desarrollo, es fundamental aplicar la reutilización de componentes de software, debido a que ésta permite ahorrar tiempo y esfuerzo a la hora de emprender nuevos proyectos. Aplicar esta ventaja de la reutilización es esencial, ya que los componentes de software se basan en soluciones ya conocidas, lo que permite aumentar la calidad, fiabilidad y eficiencia en cada uno de los nuevos productos.

La problemática actual que enfrenta la organización, es que, conociendo el negocio y teniendo una experiencia importante en el ámbito, no cuenta con un *core* definido sobre el cual aplicar el desarrollo diferenciado y particular del producto para cada grupo de clientes pertenecientes al sector judicial. Es así como surgen las líneas de productos (LPS) que permiten la reutilización sistemática de software similar pero diferenciado por algunas características. El objetivo es sacar el máximo partido de los elementos comunes, y gestionar de una manera eficaz las variaciones.

El proyecto de tesis “Línea de Productos Software de Tramitación de Causas Judiciales”, tiene como objetivo principal aprovechar el *know how* adquirido por la organización y transformarlo en una oportunidad de negocio para la misma, creando una línea de productos dentro del dominio de tramitación de causas judiciales, permitiendo a la organización contar con una serie de componentes reutilizables y también proporcionar una herramienta para gestionar en forma eficiente la variabilidad de los nuevos productos que son creados a partir de la línea.

1.2 Fundamentos de la tesis

Amisoft ha desarrollado e implementado siete sistemas de tramitación de causas judiciales permitiendo automatizar los procesos manuales existentes y eliminando el expediente físico de los tribunales de justicia. Estos sistemas informáticos han acompañado a la modernización del Poder Judicial de Chile, convirtiendo a la organización en un referente en el sector.

Estos sistemas se producían sobre la estrategia de tomar una aplicación previa que se parecía al actual y, a partir de aquí, se evolucionaba la copia de forma totalmente independiente. El parecido de las dos aplicaciones, servía para agilizar el desarrollo, dado que estaba en el mismo dominio, por lo tanto, existían las mismas funcionalidades pero con matices diferentes. Esta estrategia en ningún caso favorecía el proceso de mantención, lo que dio como resultado tener en la actualidad varios equipos trabajando en paralelo por cada sistema implementado.

En el año 2011 la organización diseñó e implementó un proceso de desarrollo de software llamado *Amisoft Process Framework* (APF) el cual reúne las mejores prácticas del CMMI, ISO 9001 y el Proceso Unificado. Con la ejecución del APF la organización logró una mejor gestión sobre sus proyectos y mayor calidad de sus productos.

Actualmente, la organización tomó la decisión de capitalizar todo el conocimiento adquirido durante estos últimos años y crear un producto que concentrara las funcionalidades necesarias para cualquier organización judicial que resuelva procesos o causas judiciales. Este producto fue creado desde cero y su principal objetivo es la tramitación de causas judiciales. Debido a la comercialización del producto a diferentes clientes, nacieron nuevas interrogantes o desafíos, respecto a múltiples temas:

- Gestión de versiones; resguardar el conjunto de activos que produzca el producto de copias indebidas o fuentes mal utilizadas por empleados de la organización o clientes.
- Evolución del producto; ampliando las funcionalidades del producto por iniciativa propia de la organización.
- Personalización del producto para cada cliente; generar la capacidad de crear eficientemente múltiples variaciones del producto.
- Costos de la mantención; cómo reducir los costos de las mantenciones futuras si el producto se vende a varios clientes.
- Código fuente poco robusto; que resulta difícil de extender con variaciones del nuevo producto y propenso al error.

Todas esas interrogantes han llevado a la organización a no encontrar las respuestas en su propio proceso de desarrollo de software (APF), ya que claramente el foco del proceso está en el producto y no en aprovechar las potenciales sinergias que se

podrían derivar de las similitudes entre los mismos productos desarrollados por la organización.

1.3 Objetivos y resultados esperados

El objetivo general del proyecto de tesis es el desarrollo de una línea de productos dentro del dominio de tramitación de causas judiciales, permitiendo a la organización contar con una serie de componentes reutilizables y gestionar en forma eficiente la variabilidad de los nuevos productos que son creados para diferentes clientes.

1.3.1 Objetivos específicos

- Conocer y comprender el concepto de Línea de Productos Software.
- Establecer un proceso de desarrollo basado en líneas de productos software con la herramienta de proceso *Eclipse Process Framework* (4).
- Desarrollar una línea de productos para la tramitación de causas judiciales.
- Definir un modelo de características con el *plug-in Feature Modeling Tool* (5).
- Desarrollar la línea de productos con Microsoft Visual Studio 2008 para crear y configurar nuevos productos de tramitación de causas seleccionando las características necesarias.

1.3.2 Resultados esperados

Los resultados esperados, a partir del desarrollo de esta tesis, son los siguientes:

- Reducir el tiempo de entrega de nuevos productos para el sector judicial.
- Incrementar el número de productos que pueden ser desarrollados por la organización a partir de la línea de productos software.
- Reducir el riesgo en la entrega de los productos.
- Mejorar la rentabilidad de los proyectos.

Capítulo II

2 Marco conceptual

2.1 *Domain Specific Languages* (DSL)

La idea básica de los lenguajes específicos de dominio (DSL) es ser un lenguaje de programación dirigido a un dominio en particular, en lugar de un lenguaje de propósito general que está dirigido a cualquier tipo de problema de software (6).

Estos son creados para resolver problemas puntuales de un dominio, como por ejemplos consultas de base de datos (SQL), despliegue de hojas de estilo (CCS), expresiones regulares, consultas de Hibernate (HQL), buena parte de Rails, entre otros.

Los DSL se distinguen entre internos y externos. Los DLS internos utilizan un lenguaje base y dan la sensación de utilizar otro más particular. Este enfoque se ha popularizado recientemente por la comunidad Ruby a pesar de que otros lenguajes lo han implementado con anterioridad. Los DSL externos tienen su propia sintaxis y se crean compiladores específicos para ellos. En la comunidad Unix está es una práctica habitual. Los DSL pueden ser implementados para ser interpretados o generar código. Normalmente el código que genera es de alto nivel, como java o C (6).

Los DSL más comunes hoy en día son textuales, están orientados a programadores permitiendo disminuir la cantidad de líneas códigos a escribir, pero también existen los DSL gráficos que permiten a los expertos del dominio validar los modelos o diagramas realizados sin la necesidad de comprender un lenguaje ajeno a su realidad.

La razón de que estos lenguajes tengan éxito es que el enfoque que ofrecen es muy útil, reducen la complejidad de la programación y son flexibles en términos de lo que se puede lograr con ellos (7).

Para llegar a obtener un DSL, existe un ciclo de vida en la evolución del software. Primero se resuelve el problema de cualquier forma que se determine. Después, al resolver el problema en un sinfín de oportunidades, se detectan buenas prácticas que permiten resolver el problema mediante la técnica “copy/paste”. Luego con la maduración y experiencia suficiente, se opta por la abstracción de estas técnicas, lo que deriva en la creación de una librería que se implementa con un diseño particular, a continuación se obtiene un patrón de diseño al abstraer el diseño de la implementación. Una vez que se obtienen varios patrones de diseño que resuelven distintos problemas en el mismo dominio, se unen para finalmente lograr la implementación de un *framework*. Por último, el *framework* se hace cada vez más configurable terminando en la evolución de un DSL (8). En resumen, un DSL es el punto de maduración de un framework; primero hay que tener el *framework*, madurarlo y finalmente se puede destilar un lenguaje específico (9).

Por lo tanto, desarrollar un DSL implica bastante complejidad, los cuales en la actualidad no están 100% resueltas, es por eso que se tomó como base un DSL existente para modelar y gestionar la variabilidad de las líneas de producto, creado por el grupo GIRO (10) en DSL Tools y Microsoft Visual Studio 2008, dado que el objetivo principal del proyecto de tesis es la creación de una línea de producto de software que permita construir variantes de un sistema, tema que también tiene mucha atracción hoy en día tanto en la industria (11), como en la investigación (12).

2.2 Línea de Productos Software

La idea básica de lo que es una línea de productos está inspirada en los procesos de producción industrializada de los productos físicos, tales como la producción de vehículos o hardware. Consiste en el ensamblaje de partes de software previamente elaboradas.

Una línea de productos es un conjunto de sistemas software que comparten un conjunto común y gestionado de aspectos que satisfacen las necesidades específicas de un segmento de mercado y que son desarrollados a partir de un conjunto común de activos fundamentales de software de una manera prescrita (13).

Las líneas de productos pueden ser descritas en términos de cuatro conceptos básicos que están representados en la Figura 2-1:

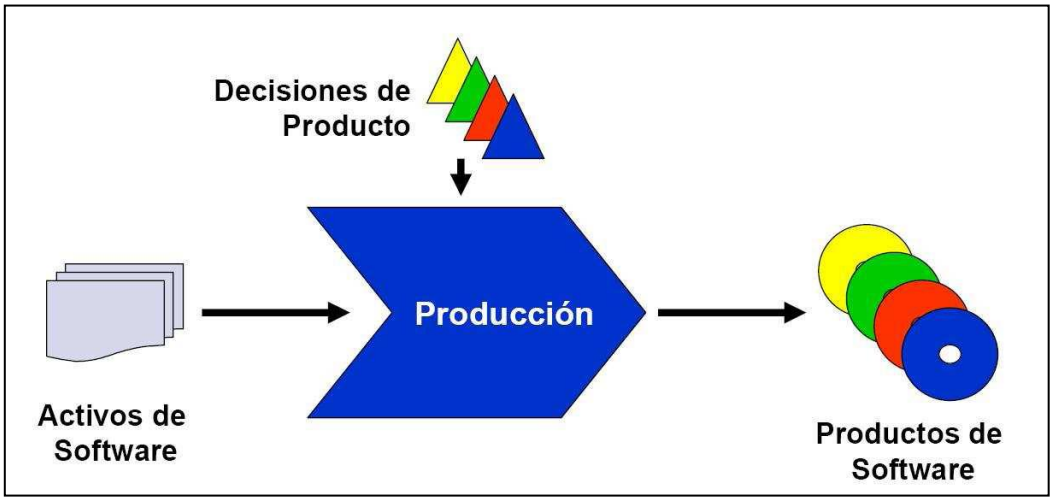


Figura 2-1: Conceptos básicos de una Línea de Productos Software.

- Activos software (*assets*): Colección de componentes software, tales como requisitos, casos de uso, arquitectura o documentación que pueden ser configurados y combinados de diferentes maneras para dar lugar a los distintos productos de una línea.
- Decisiones de producto: Cada uno de los productos pertenecientes a una línea, se diferencia de los demás gracias a la elección de las características variables y opcionales que hay en el modelo de decisión. Es decir, quedan definidos por las decisiones de producto.

- Mecanismo y procesos de producción: los medios que permiten la configuración de productos a partir de activos software reutilizables. Durante este proceso, las decisiones de producto determinan cuáles serán los componentes que van a utilizarse y cómo configurar los puntos de variabilidad entre ellos.
- Productos de software: colección de todos los productos que pueden crearse para la línea de productos. El alcance de la línea de productos está determinada por el conjunto de productos software que puede ser producido a partir de los activos reutilizables y del modelo de decisión.

2.2.1 Procesos

Los procesos de ingeniería, aspecto compartido por las distintas metodologías de desarrollo de líneas de productos, son divididos en dos equipos de trabajo (14), Figura 2-2:

- Un equipo es el encargado de la ingeniería de dominio, responsable del desarrollo de los elementos comunes al dominio, como por ejemplo: estudiar el dominio, definir los requisitos, definir las características, implementar los *core assets* reutilizable y la variabilidad, y establecer el plan de producción
- El otro equipo es el encargado de la ingeniería del producto, responsable del desarrollo de los productos para los clientes, a partir de los recursos basados en los requisitos propios del cliente. Este equipo utiliza los recursos creados por el equipo de ingeniería de dominio.

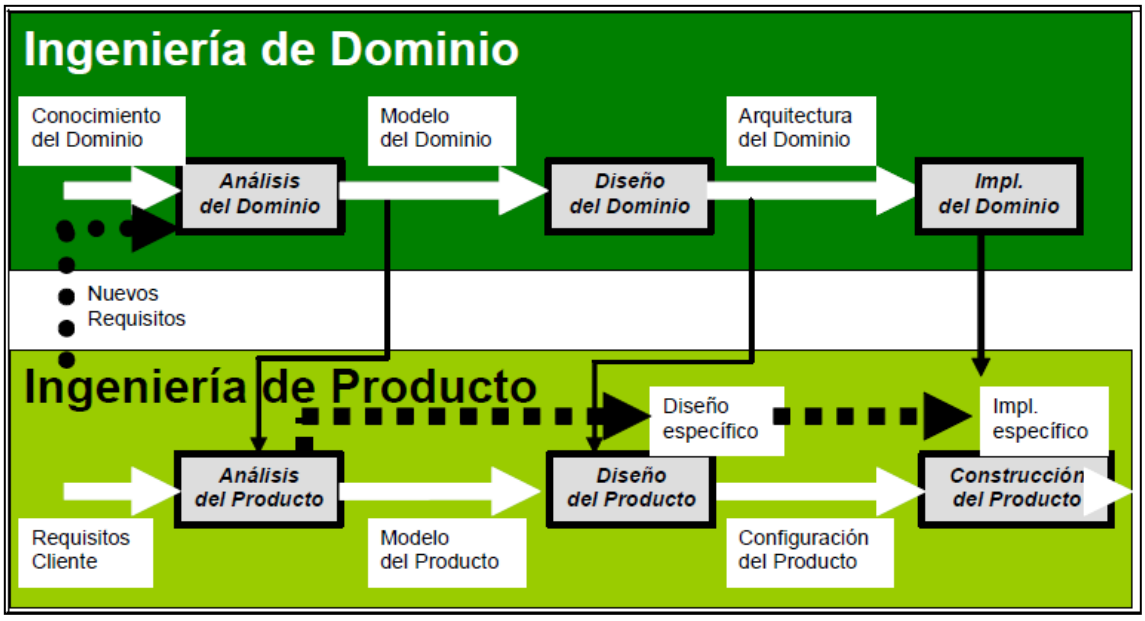


Figura 2-2: Proceso de Línea de Productos de Software.

El objetivo de la ingeniería de dominio es establecer la capacidad de producción que posteriormente se utilizará en la producción de productos específicos para clientes.

Esta actividad es iterativa, ya que se re-alimenta en el futuro de la ingeniería de producto para ir evolucionando la capacidad productiva (14).

En la ingeniería de dominio se recopilan iterativamente los requisitos comunes para toda la línea de productos. Estos requisitos se expresan en características. Luego, se realiza el análisis del dominio y diseño de los artefactos a implementar. A continuación, se implementarán las partes de código que soporten esta arquitectura. Estos elementos serán reutilizados durante la construcción de productos. Por último, se crea un plan de producción que articule toda la capacidad productiva y que sirva de guía durante el proceso de ingeniería de dominio.

La ingeniería de producto utiliza la infraestructura creada por el equipo de dominio para construir productos específicos para los clientes de la organización. Se comienza a partir de requisitos concretos de los clientes que deben estar alineados con el alcance de la línea de productos. El análisis de dichos requisitos concretos del cliente establece la viabilidad de su construcción, así como la cantidad de elementos que pueden ser reutilizados. El siguiente paso consistirá en el diseño específico del sistema. A continuación se utilizará el configurador de producto que (en base a los requisitos seleccionados) ensamblará los elementos comunes y variables para producir el producto final que se entrega al cliente (14).

2.2.2 Beneficios

El beneficio principal de utilizar la tecnología de líneas de productos es que la entrega de productos se hace de una manera más rápida y económica porque se reducen los costes de ingeniería y con una calidad mucho mayor ya que se reducen las tasas de errores.

Según Charles Krueger (15), existen una serie de argumentos a favor de las líneas de productos que se podrían dividir en beneficios tácticos y estratégicos.

Los beneficios tácticos serán los siguientes:

- Reducción en el tiempo medio de creación y entrega de nuevos productos.
- Reducción en el número medio de defectos por producto.
- Reducción en el esfuerzo medio requerido para desarrollar y mantener los productos.
- Reducción en el coste medio de producción.
- Incremento en el número total de productos que pueden ser desarrollados y mantenidos.

Por otro lado los beneficios estratégicos de negocios serían:

- Reducción en el tiempo de entrega de nuevos productos.
- Mejoras en el valor competitivo del producto.
- Márgenes mayores de ganancias.
- Mejor calidad de los productos.
- Mejoras en la reputación de la empresa.

- Mayor escalabilidad del modelo de negocios en términos de productos y mercados.
- Mayor agilidad para expandir el negocio a nuevos mercados.
- Reducción de riesgos en la entrega de productos.

2.2.3 Calidad

Los beneficios que las líneas de productos aportan a la calidad se pueden medir de dos formas. La primera mediante el grado de precisión con que cada producto se ajusta a las necesidades de cada cliente. Esta medida depende del grado de "variabilidad" de las líneas de productos. A mayor variabilidad, más probabilidades de adaptar el producto a los gustos del cliente. Sin embargo, esta variabilidad tiene un costo, y el reto es encontrar el equilibrio entre costo y variabilidad. A diferencia de los enfoques tradicionales, en las líneas de productos la variabilidad es un concepto clave. Todo el proceso de desarrollo está guiado por esta noción con el objetivo de abaratar los costos de la variabilidad, y así poder conseguir mayores cotas de variabilidad y, por tanto, de satisfacción de las peculiaridades del cliente.

Otro segundo aspecto es la tasa de defectos en los productos de las líneas de productos software. Aquí los beneficios se derivan de la reutilización de los elementos comunes. La utilización continua de estos elementos a lo largo del tiempo hace que finalmente estén muy depurados/probados. Además, los beneficios de encontrar y eliminar un defecto en un *core asset* no se limitan al producto donde se detecta el error, sino que se disemina entre todos los productos de la línea de productos (14).

Es importante tener en cuenta que los primeros productos obtenidos a través de la línea de productos que en ocasiones serán productos "test" o "piloto", van a presentar un mayor número de defectos que productos que se obtengan posteriormente. Algo similar ocurrirá con nuevas versiones de un mismo producto. El objetivo está en minimizar el número de defectos, para destinar el esfuerzo que se ha de emplear en la solución de ellos, en los aspectos indicados anteriormente.

En la Figura 2-3, se visualizan dos aspectos; la tendencia a la baja de los defectos encontrados al testear una colección de productos de una línea de productos de forma secuencial. Los defectos detectados en el producto A son corregidos y realimentados a la línea de productos, evitando de esta manera, dichos defecto se propaguen en los siguientes productos. El segundo aspecto se produce al comparar diferentes lanzamientos de los mismos productos. El segundo lanzamiento del producto A se beneficia de defectos detectados no sólo durante el lanzamiento previo de A, sino también de todos los productos de la línea de productos.

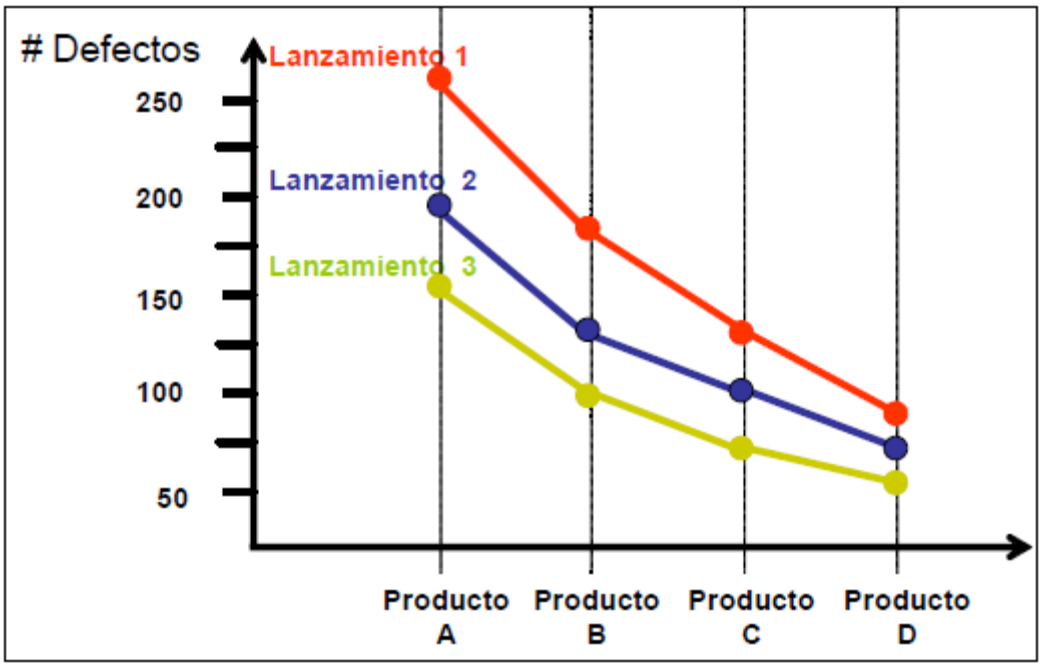


Figura 2-3: Evolución de los defectos en la línea de productos.

2.3 Trazabilidad de requisitos

La trazabilidad de requisitos es un factor reconocido de calidad en los procesos de desarrollo de software. Cuando los requerimientos se gestionan bien, la trazabilidad puede establecerse desde el requerimiento fuente hasta sus requerimientos de más bajo nivel de vuelta hasta su fuente. Esta trazabilidad bidireccional ayuda a determinar que todos los requerimientos fuente se han tratado totalmente y que todos los requerimientos de nivel más bajo pueden trazarse hacia una válida fuente. La trazabilidad se necesita particularmente cuando se lleva a cabo la evaluación del impacto de los cambios de los requerimientos sobre las actividades y los artefactos del proyecto (16). Esta propiedad es clave para la consistencia entre los diferentes modelos y etapas del desarrollo, y en el mantenimiento de las líneas de productos.

2.4 Modelo de características para gestionar la variabilidad

La variabilidad se define como la habilidad de cambio o de personalización de un sistema. Para la definición de requisitos de una línea de productos, hay que prestar especial atención al análisis de la parte común y la parte variable, estableciendo las dependencias que existen entre ellas (17).

Para ello, los métodos más utilizados son los basados en características, como FODA (*Feature Oriented Domain Analysis*) (18). Esta es una metodología desarrollada por el SEI (Software Engineering Institute) para la aplicación del análisis de dominio, definiendo las etapas del método y los resultados obtenidos en cada una de ellas. Para FODA, las características son abstracciones de las capacidades del dominio que deben ser implementadas, probadas, entregadas y mantenidas. En el modelo de características se especifican las capacidades y las restricciones tecnológicas que

deben y que pueden aparecer en los productos de la línea de productos y que son visibles al usuario final (14).

El método se basa en que una característica se pude descomponer en varias “sub-características” (obligatorias, opcionales o alternativas). El modelo de características produce una descomposición jerárquica en forma de árbol o de un conjunto de árboles.

En resumen, la variabilidad de la línea de productos queda plasmada en el modelo de característica a través de las representaciones gráficas de los puntos de variación y de sus variantes, así como de las características opcionales, externas, parametrizadas y de cambio.

2.5 Variabilidad, trazabilidad y líneas de productos basada en UML y clases parciales

Según Miguel Laguna y Bruno González (grupo GIRO), uno de los problemas clave en el desarrollo de una línea de productos es la representación y gestión de la parte común y variable de la misma. La forma habitual de definir ambos aspectos es mediante modelos de características que además permiten seleccionar la configuración de cada aplicación concreta dentro de la línea de productos. Sin embargo, la trazabilidad entre los modelos de características y los modelos de diseño (generalmente basados en UML) no es sencilla (2).

La propuesta que plantean, consiste en utilizar el mecanismo de combinación de paquetes (“*package merge*”) de UML 2 como herramienta de representación y configuración de la variabilidad de la línea de productos y reservar los mecanismos clásicos de modelado (la especialización de los modelos estructurales o la relación <<extend>> de los modelos de casos de uso) para expresar las variantes válidas en tiempo de ejecución de cada aplicación concreta. La estructura de los modelos de características se refleja directamente en las relaciones entre paquetes del modelo arquitectónico de modo que la trazabilidad de las decisiones de configuraciones es automática. La implementación directa de la combinación de paquetes mediante clases parciales, disponibles en lenguajes como C#, es una ventaja adicional que permite acerca al ideal de desarrollo sin parches.

2.5.1 “*Package Merge*” en UML 2 y variabilidad

El mecanismo de combinación de paquetes (“*package merge*”) consiste fundamentalmente en añadir detalles de forma incremental, como se visualiza en la Figura 2-4.

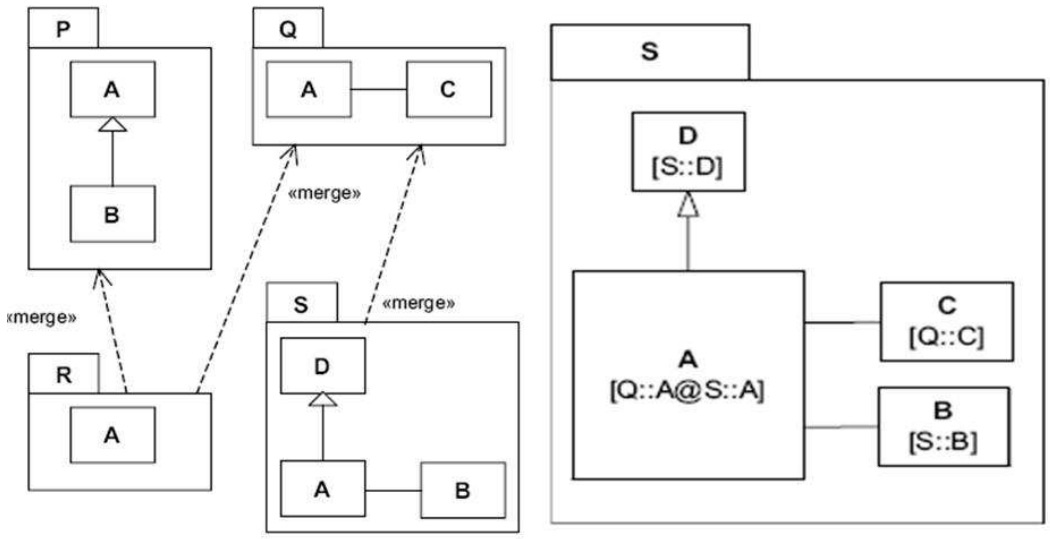


Figura 2-4 Ejemplo del mecanismo de "package merge" extraído de la especificación del meta- modelo de la infraestructura de UML2.

Según la especificación UML, se define como una relación entre dos paquetes que indica que los contenidos de ambos se combinan. Es similar a la generalización y se utiliza cuando elementos en distintos paquetes tienen el mismo nombre y representan el mismo concepto, comenzando por una base común.

Dicho concepto se extiende incrementalmente en cada paquete añadido. Seleccionado los paquetes deseados es posible obtener una definición a la medida de entre todas las posibles. Aunque lo importante es sobre los diagramas de clases, el mecanismo se puede extender a cualquier diagrama de UML, en particular los de casos de uso y los de secuencia. Evidentemente, las reglas que establece la especificación UML 2 son muy estrictas para evitar inconsistencias. Por ejemplo, no puede haber ciclos, las multiplicidades resultantes son las menos restrictivas de entre las posibles, o las operaciones deben conformar en número, orden y tipo de parámetros.

Además de clases, cualquier otro meta-tipo del meta-modelo de UML se puede utilizar con esta técnica. La semántica está definida en la especificación para los meta-tipos más comunes pero existen reglas generales aplicables al resto. La filosofía general es que el elemento resultante tiene que ser al menos tan capaz como el original antes de la combinación.

Este mecanismo permite establecer una trazabilidad clara entre los modelos de características y los artefactos UML que sean diseñados durante el proceso. La aplicación consiste en establecer, en primer lugar y para cada modelo UML del diseño, un paquete base que recoge la parte común de la línea de productos. Este paquete base se puede organizar del modo habitual (utilizando la descomposición recursiva en paquetes y las relaciones de <<import>> o <<access>> entre ellos). A este paquete base se añade un paquete por cada característica opcional, de modo que queden localizadas en ese paquete todas las modificaciones necesarias en el modelo de diseño

asociadas a la característica. El paquete añadido se conecta mediante la relación <<merge>> con su paquete base en el punto preciso de la jerarquía de paquetes. Cada punto de variación detectado en el modelo de características hace que aparezca un paquete que se combinará o no en tiempo de desarrollo del producto según la configuración de características elegidas. La representación de los paquetes no es únicamente gráfica: también es una vista lógica de los componentes software y de su organización. En la Figura 2-5 se visualiza la estructura de paquetes de clases que refleja el esquema propuesta desde modelo de características.

De esta forma, se localizan en un sólo paquete del modelo todas las variaciones que origina cada característica opcional manteniendo una correspondencia uno a uno, se separan claramente los dos tipos de variabilidad, eliminando las ambigüedades. La variabilidad de la línea de producto viene dada por la selección de paquetes. La variabilidad de aplicaciones concretas está dentro de cada paquete y se mantiene sin cambios el meta-modelo de UML.

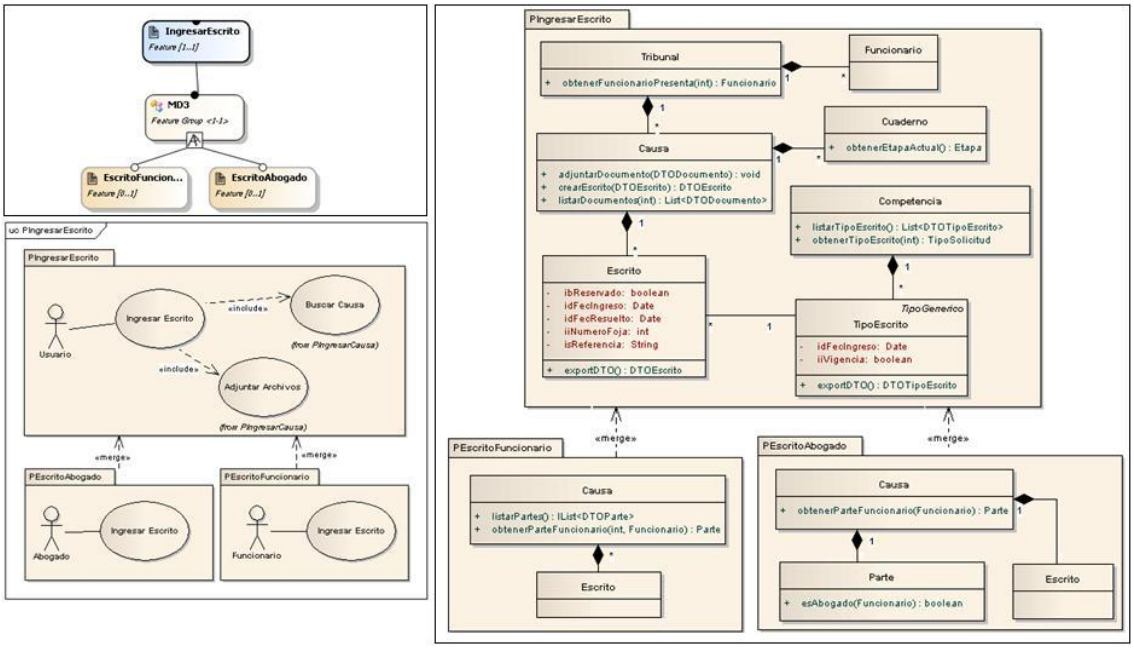


Figura 2-5: Modelo de características y ejemplo del mecanismo "*package merge*" aplicado a casos de uso y diagrama de clases.

2.5.2 Implementación de la línea de productos con clases parciales C#

Para extender la trazabilidad hasta los modelos de implementación se utiliza el concepto de clases parciales. La utilización de "mixins" o clases parciales fue propuesta originalmente en el lenguaje Flavors (3) y representa una alternativa a la herencia múltiple y una manera de manejar la variabilidad relacionada con aspectos. La intención es mantener la correspondencia uno a uno no sólo entre características y paquetes de diseño sino también con la estructura del código.

En el modelo de desarrollo, se utiliza el mecanismo de clases parciales de C# para implementar la línea de productos dentro de la estructura de soluciones y paquetes de

la plataforma Microsoft Visual Studio 2008. Si el framework que implementa la arquitectura de la línea de productos está organizada en paquetes de clases parciales (un paquete base y tantos paquetes auxiliares como variaciones existen), para derivar una producto concreto basta con importar o referenciar los paquetes que se correspondan directamente con la configuración elegida en el modelo de características. Estos paquetes opcionales pueden añadirse o no al proyecto que representa una aplicación concreta de la línea de producto utilizando los ficheros de configuración del compilador. De esta manera se establece la trazabilidad uno-a-uno desde las características hasta el código.

Estos paquetes opcionales pueden añadirse o no al proyecto que representa un producto concreto de la línea de producto utilizando el configurador del *plug-in* FMT (5).

2.6 Desarrollo Dirigido por Características (FDD)

La metodología seleccionada para el desarrollo e implementación del proyecto de tesis es el Desarrollo Dirigido por Características (*Feature-Driven Development*) (1), proceso ágil que se basa en construir iteraciones cortas que produzcan incrementos funcionales en el software que los clientes y personas encargadas de la gestión del proyecto puedan ver, analizar y aprobar.

En cada iteración se trata de construir una característica, definida como un incremento en la funcionalidad de un producto con significado para el cliente. La metodología consiste en cinco procesos secuenciales, los cuales se citan a continuación, Figura 2-6:

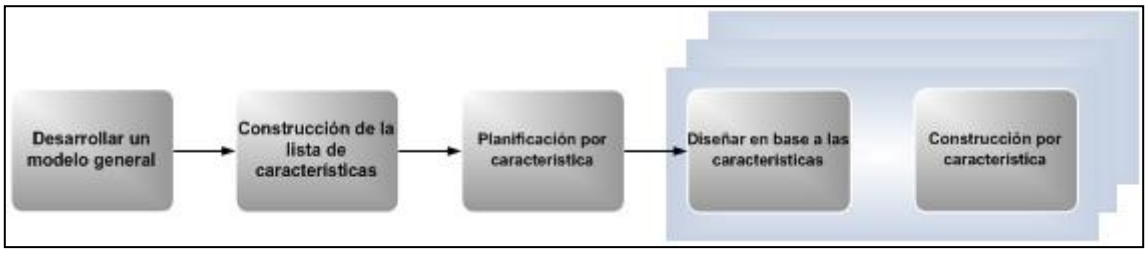


Figura 2-6: Metodología FDD.

- **Desarrollo de un modelo general**: Cuando esta fase comienza, los expertos del dominio ya tienen una idea del contexto y los requerimientos del sistema. Es probable que el documento de especificación de requerimientos ya exista. Sin embargo, FDD no hace énfasis en la recolección y administración de los requerimientos. Los expertos del dominio presentan un informe llamado *walkthrough* en el cual los miembros del equipo y el *Chief Arquitect* son informados a través de una descripción de alto nivel del sistema.
  El dominio global es dividido en diferentes áreas y se realiza un *walkthrough* detallado para cada una de dichas áreas por parte de los expertos del dominio. Luego de cada *walkthrough*, un grupo de desarrolladores realizan un modelo de objetos para el área del dominio. Además, el equipo de desarrollo discute y decide

cual es el modelo de objetos más apropiado para cada área del dominio. Simultáneamente, el modelo global del sistema es construido.

- **Construcción de la lista de características**: Los *walkthroughs*, el modelo de objetos y los requerimientos existentes ofrecen una buena base para construir una *features list* que resuma la funcionalidad del sistema a ser desarrollado. En dicha lista, el equipo de desarrolladores presenta cada una de las funcionalidades evaluadas por el cliente. Las funcionalidades son presentadas por cada área del dominio y éstas, forman un m*ajor list sets*. Dicha lista es divida en subconjuntos en base a la funcionalidad. Éstas, representan diferentes actividades con un área específica del dominio. La *features list* es revisada por los usuarios y sponsors del sistema para su validación y aprobación.
- **Planificación por característica**: En esta etapa se incluye la creación de un plan de alto nivel, en el cual la *features list* es ordenada en base a la prioridad y a la dependencia entre cada *feature*. Además, las clases identificadas en la primera etapa son asignadas a cada programador.
- **Diseñar en base a las características:** El *Chief Programmer* toma la próxima *feature* a ser diseñada, identifica las clases involucradas y contacta al *Class Owner* correspondiente. Luego el equipo, trabaja en la realización del diagrama de secuencia correspondiente. El *Class Owner* hace una descripción de la clase y sus métodos.
- **Construcción por característica**: En esta etapa, cada *Class Owner* construye los métodos de clase para cada *feature* correspondiente y luego realiza el testing unitario para cada una de las clases. A su vez, se realiza una inspección del código y luego que el código es implementado e inspeccionado, el *Class Owner* realiza un *check-in* al CMS (*Configuration Management System*). Luego, se realiza un *main build* en el cual se hace la integración con la funcionalidad antes realizada. También se realizan los testing de integración correspondiente.

Las tres primeras fases ocupan gran parte del tiempo en las primeras iteraciones, siendo las dos últimas las que absorben la mayor parte del tiempo según va avanzando el proyecto, limitándose las primeras a un proceso de refinamiento.

La ventaja principal de esta metodología es que está orientada y gobernada por características que son un concepto clave en el desarrollo de líneas de productos.

La metodología de desarrollo dirigido por características es un proceso que ayuda al equipo a producir resultados periódicos y tangibles.

- Utiliza pequeños bloques que contienen la funcionalidad del sistema, llamadas características.
- Organiza los bloques que están relacionados entre sí, en una lista llamada lista de características.
- Hace énfasis en la obtención de resultados cada dos semanas.

- Incluye estrategias de planificación que hacen que las características puedan desarrollarse en dichos lapsos.

Cabe recordar que para el desarrollo correcto de este proyecto de tesis, se debe realizar dos tipos de proyectos: el desarrollo de la familia de productos que mediante composición de características se puedan obtener productos de software pertenecientes a dicha familia y el desarrollo de los mecanismos necesarios para que la creación de productos específicos, a partir de la infraestructura creada en el punto anterior, sea lo más automática posible.

2.6.1 Roles

- **Project Manager:** Es el líder administrativo y financiero del proyecto. Una de sus tareas principales es proteger al equipo de distracciones externas y permitir que el equipo pueda trabajar en las condiciones apropiadas. En el FDD el *Project Manager* tiene la última palabra sobre los temas referidos al alcance, tiempo y personal.
- **Chief Architect:** Es el responsable del diseño global del sistema y de la ejecución de todas las etapas del diseño. También tiene la última palabra sobre las decisiones del diseño de todo el sistema. Si es necesario, este rol puede ser dividido en dos roles como ser *Domain Architect* y *Tecnical Architect*.
- **Development Manager:** Lleva diariamente las actividades de desarrollo y resuelve cualquier conflicto que pueda ocurrir con el equipo. Además, este rol incluye la responsabilidad de resolver problemas referentes a los recursos. Las tareas de este rol pueden ser combinadas con las del *Chief Architect* o el *Project Manager*.
- **Chief Programmer:** Es un desarrollador con experiencia, el cual participa en el análisis de los requerimientos y el diseño del proyecto. El *Chief Programmer* es el responsable de guiar a pequeños equipos en el análisis, diseño y desarrollo de nuevas funcionalidades. Además, selecciona las funcionalidades a desarrollar de la lista de funcionalidades de la próxima iteración en la última fase del FDD, identifica las clases y el *Class Owner* que se necesita en el equipo para la iteración. Con la ayuda de otros *Chief Programmers* resuelve problemas técnicos y de recursos, y reporta el progreso del equipo durante la semana.
- **Class Owner:** Trabaja bajo la guía del *Chief Programmer* en las tareas de diseño, codificación, testing y documentación. Él es responsable del desarrollo de las clases que se le asignaron como propias. Para cada iteración los *Class Owner* participan en la decisión de que clase está incluida en la lista de funcionalidades de la próxima iteración.
- **Expertos del dominio**: Los expertos del dominio pueden ser un usuario, un cliente, un *sponsor*, un analista del negocio o una mezcla de éstos. Su tarea es poseer el conocimiento de los diferentes requerimientos del sistema. El experto del dominio pasa el conocimiento a los desarrolladores de manera tal que asegure que estos entreguen un sistema completo.

Capítulo III

3 Desarrollo de la línea de productos software

Para el desarrollo del proyecto de tesis, la metodología seleccionada fue el desarrollo dirigido por características (FDD). Dicha metodología se ha elegido entre las conocidas como ágiles (17), al entender que por el tiempo que el alumno puede dedicarle y la necesidad de cierta flexibilidad en el desarrollo de sus requisitos, son las que mejor se adaptan a las particularidades del proyecto de tesis.

El capítulo III consiste en el desarrollo de un modelo general que describe el dominio y contexto de la línea de productos, la construcción y planificación de una lista de características y por último el diseño y construcción de las características que fueron las priorizadas para el desarrollo.

3.1 Sistemas de tramitación de causas judiciales

Los sistemas informáticos de tramitación de causas judiciales son sistemas que permiten registrar los datos y movimientos relacionados con el ingreso, actuaciones y término de la causa que recibe o inicia un tribunal de justicia, fiscalía o cualquier organización que gestione algún tipo de controversia.

Su principal objetivo es mejorar el control del avance del proceso judicial, consiguiendo, de paso, reducciones en los tiempos procesales, mayor seguridad en el resguardo de información y mayor acceso de las partes a la causa (19).

Para cada causa, el sistema crea un expediente digital donde almacena y registra la información, además, crea elementos de apoyo al seguimiento de la tramitación de causas.

Algunas de las características de estos sistemas son:

- Poder transitar por más de una competencia (La competencia es la facultad que tiene cada tribunal para conocer de los negocios que la ley ha colocado dentro de la esfera de sus atribuciones).
- Poder pasar a diferentes instancias (tribunales, cortes de apelaciones y corte suprema).
- Permite realizar auditoria y trazabilidad de la causa.

En términos muy amplios, un expediente digital contiene documentos que entregan las partes del caso, otros documentos solicitados o entregados por terceras instituciones o personas y documentos que emite el tribunal. La formación del expediente sigue reglas muy específicas que generalmente están contenidas explícitamente en normas procesales que previamente se definen dentro del sistema.

Estos sistemas están generalmente relacionados al tipo de procedimiento judicial que debe seguir la causa. Se pueden clasificar en 3 tipos, los cuales son:

- Permite la tramitación de expedientes con una lógica de organización y de funcionamiento tradicional. Dentro de este tipo se puede englobar a todos los tribunales que conocen de materias que se siguen tramitando por el proceso civil, y sus variaciones.
- Permite la tramitación de expedientes mediante procesos orales, entendiendo por ello el que las decisiones judiciales relevantes se adopten por parte de los jueces en audiencias orales y públicas.
- Permite la tramitación de expedientes de altos volúmenes de causas, usualmente repetitivos y de una complejidad baja, y para lo cual se han creado procedimientos judiciales “especiales”.

Las funcionalidades comunes que existen para este tipo de sistemas son:

- Proporcionar información acerca de las partes de la causa y de los movimientos y el estado de tramitación.
- Permitir el registro de todos los datos relevantes desde el punto de vista de la información estadística que se elaboró a partir del movimiento de las causas.
- Las actuaciones y resoluciones que emite el tribunal en el marco de una causa, se realizan por parte del personal del tribunal en el sistema, y desde aquí se imprime la resolución y se agrega al expediente.
- Integración con Microsoft Word para la generación de documentos y plantillas tipo para las resoluciones.
- Flujo de trabajo parametrizable de los procedimientos judiciales, donde permite configurar los hitos y plazos relacionados.
- Permitir el ingreso de escritos (solicitudes) de las partes de una causa para ser conocidas por el tribunal.
- Generar una distribución de causas a un tribunal o juez determinado.
- Fijación de audiencias tendiendo a la vista las agendas de los actores relevantes (juez).
- Registro de audio de lo que ocurra en la audiencia.
- Administración de notificaciones y comunicación electrónica con las partes (*E-mail*, carta certificada, entre otros).
- Generación y publicación de estados diarios (medio de notificación)
- Administración de custodias de las especies que pertenecen a una causa y que son custodiadas por el tribunal.
- Interconexión con terceras instituciones.
- Interconexión con tribunales de segunda instancia.

3.2 Contexto

Amisoft ha desarrollado e implementado siete sistemas de tramitación de causas judiciales tanto para tribunales de primera instancia, cortes de apelaciones y la corte suprema del país. Dichos sistemas fueron adjudicados por la organización a través de licitaciones públicas que realizó el Poder Judicial de Chile desde el año 2004 hasta el

2010. El primer sistema de tramitación licitado fue para los tribunales de familia, seguido de los tribunales de cobranza previsional y laboral, para luego continuar con los tribunales civiles y por último los tribunales laborales.

Todos estos sistemas fueron proyectos diseñados e implementados independientemente, dando como resultado actual no tener una estrategia de reutilización de aquellos componentes comunes entre si y tener un equipo de mantención por cada sistema desarrollado. Una de las razones de por qué no fue pensada una estrategia de reutilización ni unir todos los proyectos en uno solo, se debió a que todos nacieron bajo el alero de una reforma judicial que incluía dentro de varios aspectos los recursos necesarios para el diseño e implementación de un sistema informático para agilizar todos los procesos relacionados.

Al analizar las funcionalidades principales desarrolladas para cada sistema, se deduce que existe un número importante de funcionalidades en común y otras particulares de cada sistema, aún cuando están clasificados en diferentes tipos de sistemas de tramitación de causas. A continuación se describen dichas funcionalidades:

Funcionalidad	Sistema Familiar	Sistema Cobranza	Sistema Laboral	Sistema Civil
Ingreso de Causa	X	X	X	X
Ingreso de Causa Internet	X		X	
Ingreso de Menores	X			
Ingreso Exhortos	X	X	X	
Ingreso Exhortos Otros Tribunales	X	X	X	
Exhortante		X		
Exhortado		X		
Ingreso Incompetencia	X	X	X	
Modificación de Causa	X	X	X	X
Consulta Ingreso	X	X	X	X
Ingreso Escritos	X	X	X	X
Tramitar Expediente	X	X	X	X
Escritos Pendientes	X	X	X	X
Trámites Pendientes	X	X	X	X
Actuaciones Pendientes	X	X		
Búsqueda de Plazos	X	X	X	
Causas Sin Movimientos	X		X	
Notificaciones	X	X	X	X
Programar Audiencia	X		X	
Custodia	X	X	X	X
Consulta de Causas	X	X	X	X
Consulta de Audiencia	X		X	
Diligencias	X	X	X	
Login	X	X	X	X

Tabla 1: Variabilidad de los sistemas desarrollados por la organización.

Todos los sistemas se construyeron sobre la estrategia de tomar el último sistema desarrollado y, a partir de aquí, evolucionar la copia de forma totalmente independiente. El parecido de los dos sistemas, servía para agilizar el desarrollo, dado que estaba en el mismo dominio, por lo tanto, existían las mismas funcionalidades pero con matices diferentes.

A principios del año 2011, la organización tomó la decisión de crear un producto que concentrara las funcionalidades necesarias para cualquier organización que resuelva procesos judiciales. Este sistema nuevamente fue desarrollado desde cero, ya que el lenguaje de desarrollo en la cual se implementó fue diferente, pero una vez que se comenzó con la venta del producto, empezaron a nacer nuevas problemáticas en la evolución y la personalización eficiente del producto para cada cliente.

Nuevamente la organización cometió los mismos errores de los sistemas anteriores, al no tener una estrategia de reutilización de componentes para evitar el gran costo de tener varios equipos manteniendo y soportando todos los productos adaptados que logrará vender a los clientes. Por el contrario, si la organización hubiera adoptado una estrategia de líneas de producto, tomando en cuenta la gran experiencia en el dominio, hoy tendría una base firme en gestionar la variabilidad de estos productos reduciendo considerablemente los equipos de mantención y soporte, aumentando la calidad y mitigando el riesgo de entrega en las fechas pactadas. A lo anterior se suma la escaza información que hoy cuenta la organización a la hora de poder adaptar los productos a las necesidades particulares del cliente, lo que se traduce en adquirir compromisos, por el área comercial, que tienen escaza factibilidad técnica para su implementación, generando un descontento en las áreas operativas, debido al enorme desgaste en los “parches” que muchas veces se deben recurrir para lograr la adaptación del cliente. Esta situación genera un porcentaje importante de pérdidas en las utilidades de los proyectos, generando un nivel escaso de competitividad a la organización a la hora de competir con otro tipo de organizaciones del mercado.

En la actualidad, el mecanismo que existe para gestionar la variabilidad del producto es mediante el esquema de ocultar opciones del producto, vía el menú de usuario, para no ser visualizadas ni ocupadas por el cliente cuando no es técnicamente posible eliminar el código de las funcionalidades que no son utilizadas por el usuario.

El desarrollo del proyecto de tesis plantea una solución a la problemática descrita, diseñando características principales en el dominio judicial y construyendo esquemas automáticos que reducen significativamente el esfuerzo de crear nuevos productos y aumentando la entrega de estos a los clientes.

3.3 Desarrollo de un modelo general

3.3.1 Modelo de dominio

Un modelo de dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés (20).

Para la creación del modelo de dominio no fue necesario realizar entrevistas a usuarios expertos del dominio, debido a la amplia experiencia que el alumno posee sobre el dominio judicial y el diseño e implementación de sistemas de tramitación judicial de causas.

El modelo de dominio de la línea de productos de software de tramitación de causas judiciales está dividido en diferentes áreas, haciendo énfasis en los objetos y asociaciones. Se divide en 3 paquetes significativos para obtener una mejor compresión de su lógica. A continuación se detallan dichos paquetes:

- Autómata: Este paquete contiene un agrupamiento lógico de objetos que permiten identificar los procedimientos judiciales (flujos de trabajo) y toda la configuración necesaria para su funcionamiento correcto en el sistema.
- Roles: Este paquete contiene un agrupamiento lógico de objetos que permiten identificar los roles de los usuarios que interactúan en el sistema.
- Tribunal: Este paquete contiene un agrupamiento lógico de objetos que permiten identificar la tramitación generada por una causa y toda la información relacionada.

En la Figura 3-1, se ilustra la interacción de los objetos del dominio de las competencias que tramitan causas judiciales. Una competencia es la facultad que tiene una organización para conocer de los negocios que la ley o reglamentos ha colocado dentro de la esfera de sus atribuciones.

El objeto “Competencia” tiene una colección de “Automata”. Cada “Automata” representa un procedimiento judicial (flujo de trabajo). El objeto “Automata” a su vez contiene una colección de “CuadernoAutomata” que representa una separación del expediente digital en varias partes, sin embargo, cada “Automata” debe siempre al menos tener un “CuadernoPrinicipal”. El objeto “CuadernoAutomata” contiene una colección de “Etapa”, estas etapas pueden contener una colección de “Hito” que representa la tipificación de la técnica jurídica aplicada en el procedimiento judicial. Al tipificar los hitos, se describe el flujo de trabajo en nomenclaturas que ayudan en la obtención de información y permiten guiar la tramitación de la causa a los participantes. Con esto, se logra saber que hitos están disponibles para dictar en cada etapa. A su vez, cada “Hito” tiene asociado un “TipoTermino” que permite tipificar si el hito da por finalizado el procedimiento judicial o no, un “Plazo” que permite controlar los avances del procedimiento judicial, un “TipoEstadoProcesal” que representa el estado procesal que debe adoptar el procedimiento y un “TipoEstadoAdministrativo” que representa el

estado administrativo de la organización que debe adoptar el procedimiento. La diferencia de los dos estados mencionados anteriormente, es simplemente separar estados procesales que permiten realizar una mejor gestión sobre los expedientes.

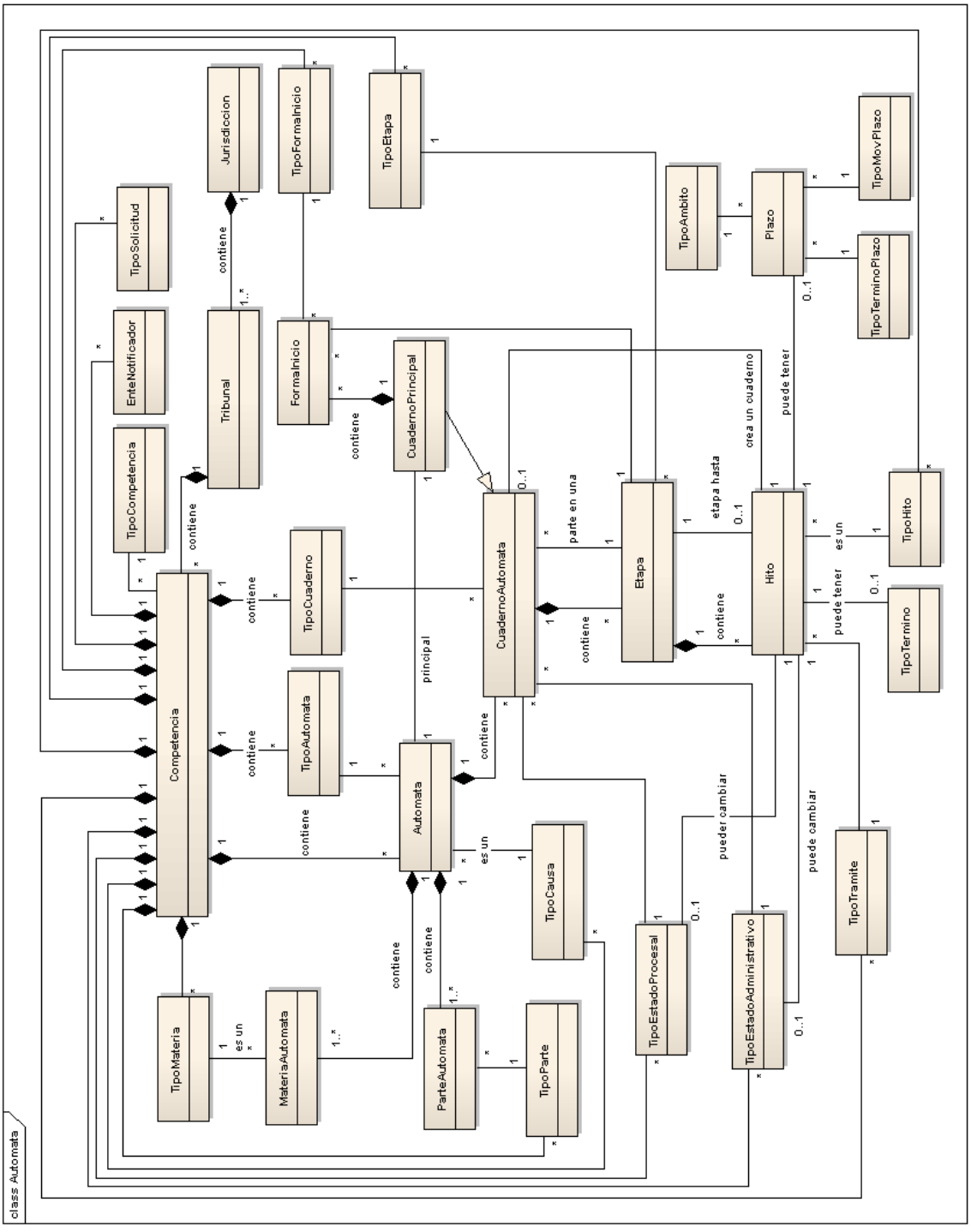


Figura 3-1: Modelo de dominio autómata.

En la Figura 3-2, se ilustra la interacción de los objetos del dominio de un tribunal que tramita causas judiciales. Un tribunal es cualquier organización judicial que tramite asuntos de controversia.

El objeto “Tribunal” tiene una colección de “Causas”. Cada “Causa” está relacionada a un “Cuaderno” principal y además puede tener una colección de otros “Cuadernos” que son agregados en la tramitación de la “Causa”. A su vez, el “Cuaderno” principal contiene una colección de “Trámites” que representan las resoluciones o actuaciones dictadas por un juez. Estos “Trámites” contienen una colección de “Hitos” que determinan lo que se está resolviendo por el juez y se asocia a la “Solicitud” (escrito) que representa la petición presentada por una “Parte” de la “Causa”. Cada “Trámite” puede o no tener un “Documento” asociado que forman el expediente digital de la causa.

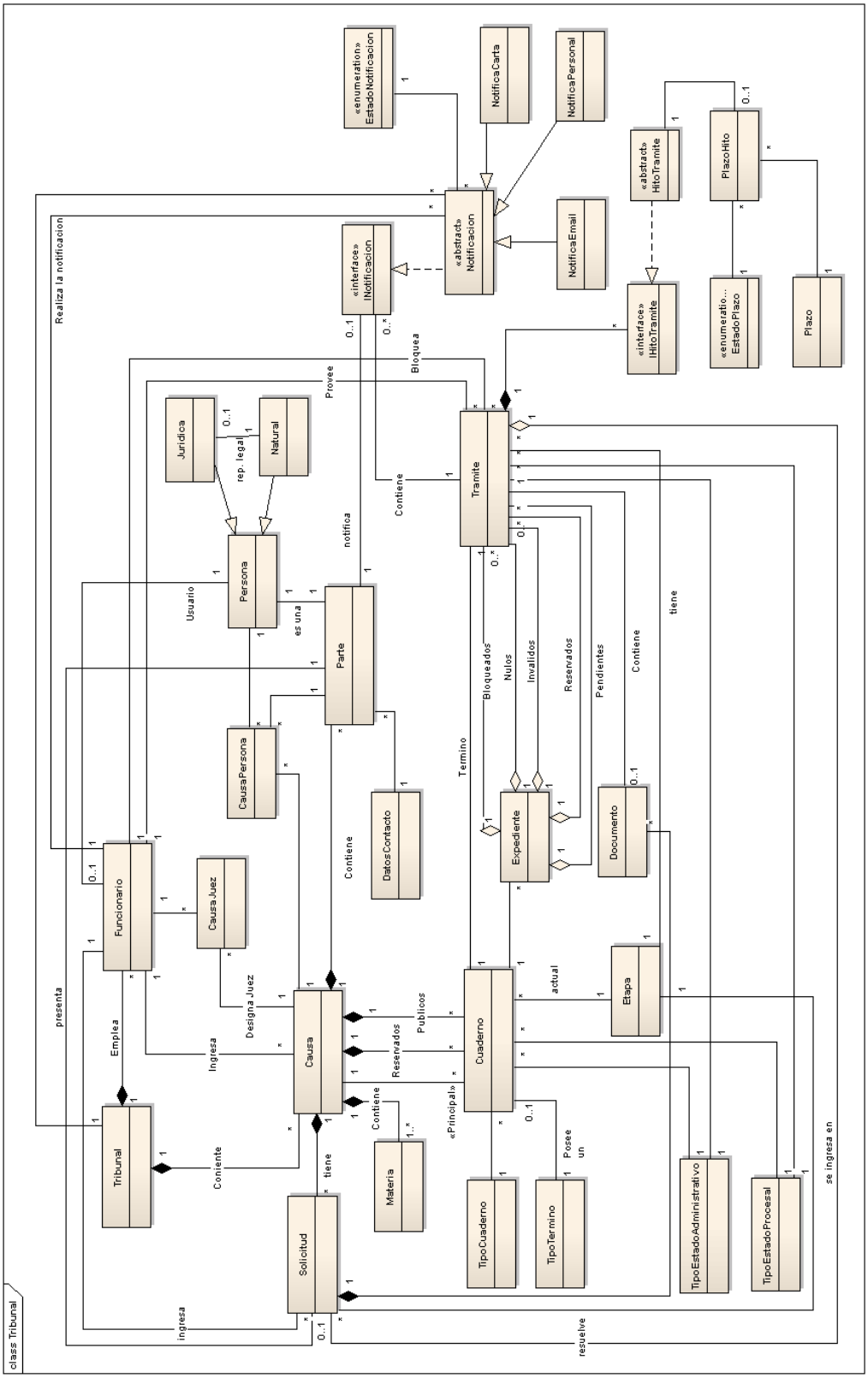


Figura 3-2: Modelo de dominio tribunal.

En la Figura 3-3, se ilustra la interacción de los objetos de los roles que interactúan en un sistema de tramitación de causas.

El objeto “Tribunal” tiene una colección de “Funcionarios” que representan las personas que desempeñan un rol dentro de la organización. Cada “Funcionario” contiene una colección de “Perfil”, es decir, puede cumplir más de un rol dentro de la organización. Este “Perfil” tiene una colección de “OpcionPerfil” que representa las opciones que puede realizar dicho perfil.

El objeto “Perfil” muestra una jerarquía de perfiles que se puede adoptar, tales como un “Notificador”, “AdministradorTribunal”, “Auditor” y “AdministradorSistema”.

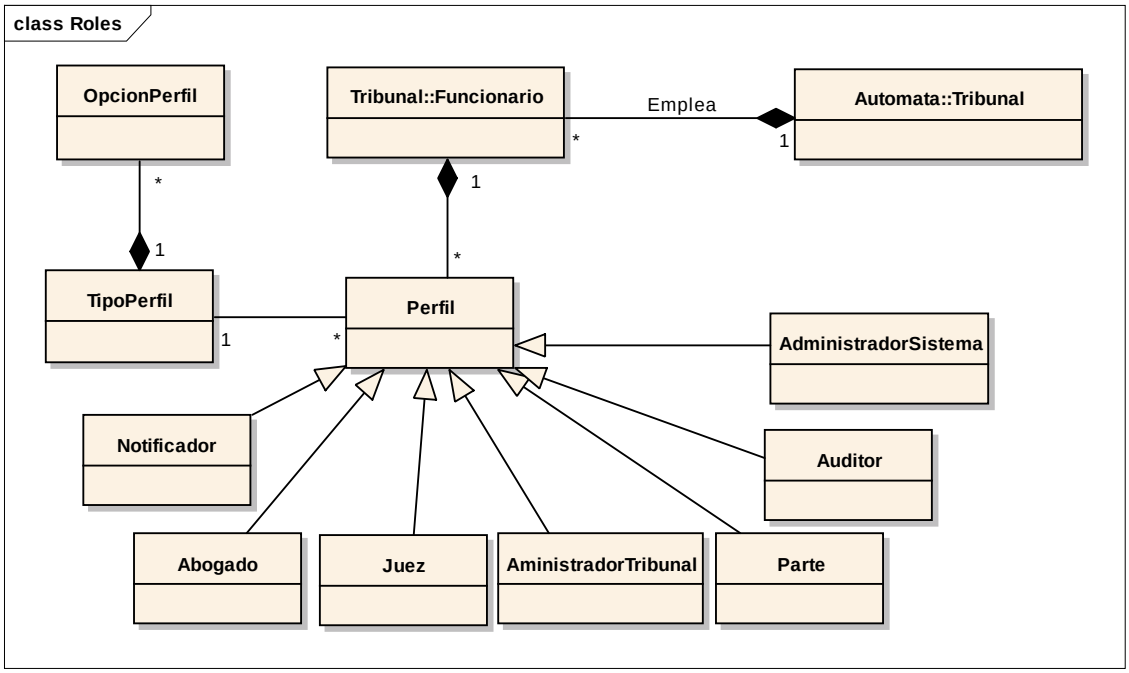


Figura 3-3: Modelo de dominio roles.

3.4 Construcción de la lista de características

Para la definición de requisitos de una línea de productos, hay que prestar especial atención al análisis de la parte común y la parte variable, estableciendo las dependencias que existen entre ellas.

Se realizó un análisis detallado de todas las características y funcionalidades previstas para la línea de productos de tramitación de causas judiciales, utilizando un modelo de características diseñado en el *plug-in* FMT (5), de las funcionalidades comunes que puede adoptar la familia de productos a implementar.

En la Figura 3-4 y Figura 3-5, se detallan las distintas características que un producto final de tramitación de causas judiciales puede adoptar, indicando las que son obligatorias, opcionales o alternativas.

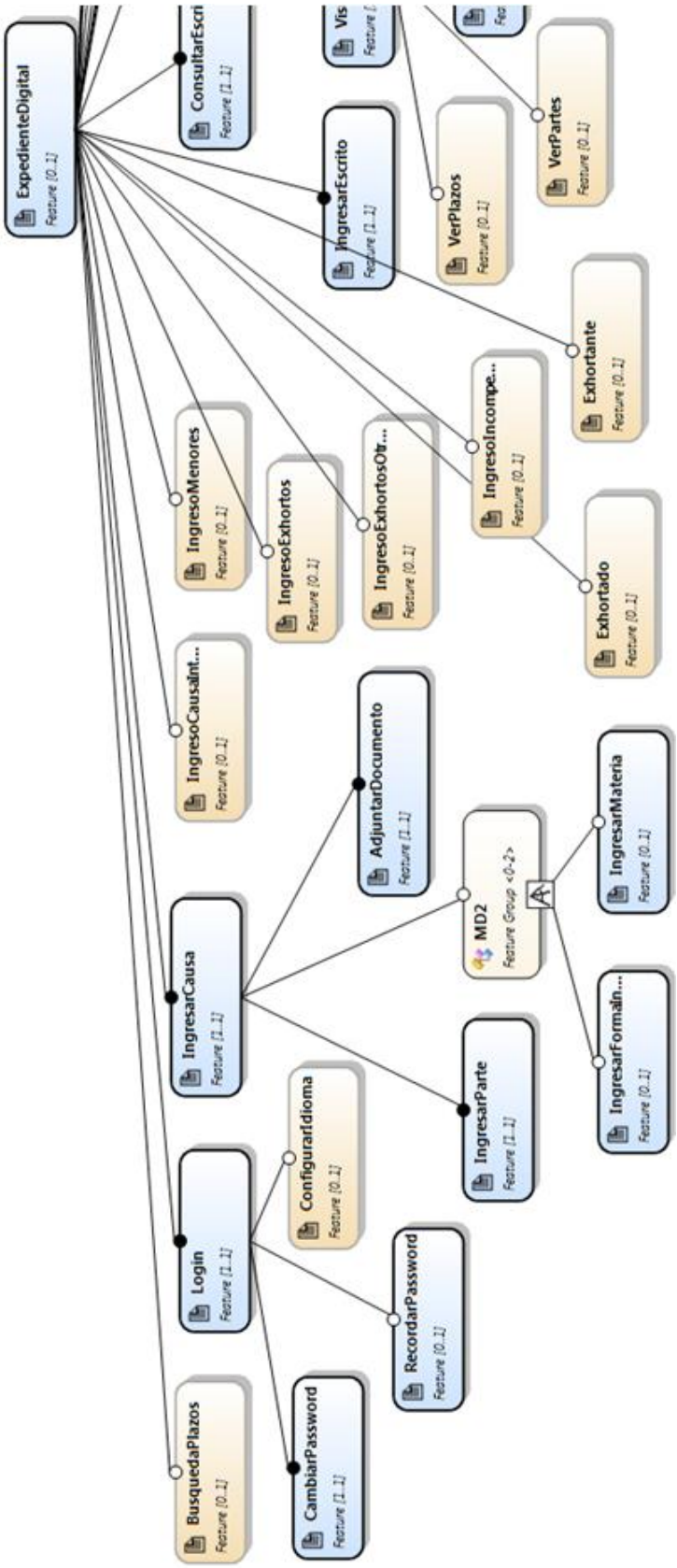


Figura 3-4: Modelo de características expediente digital (parte 1).

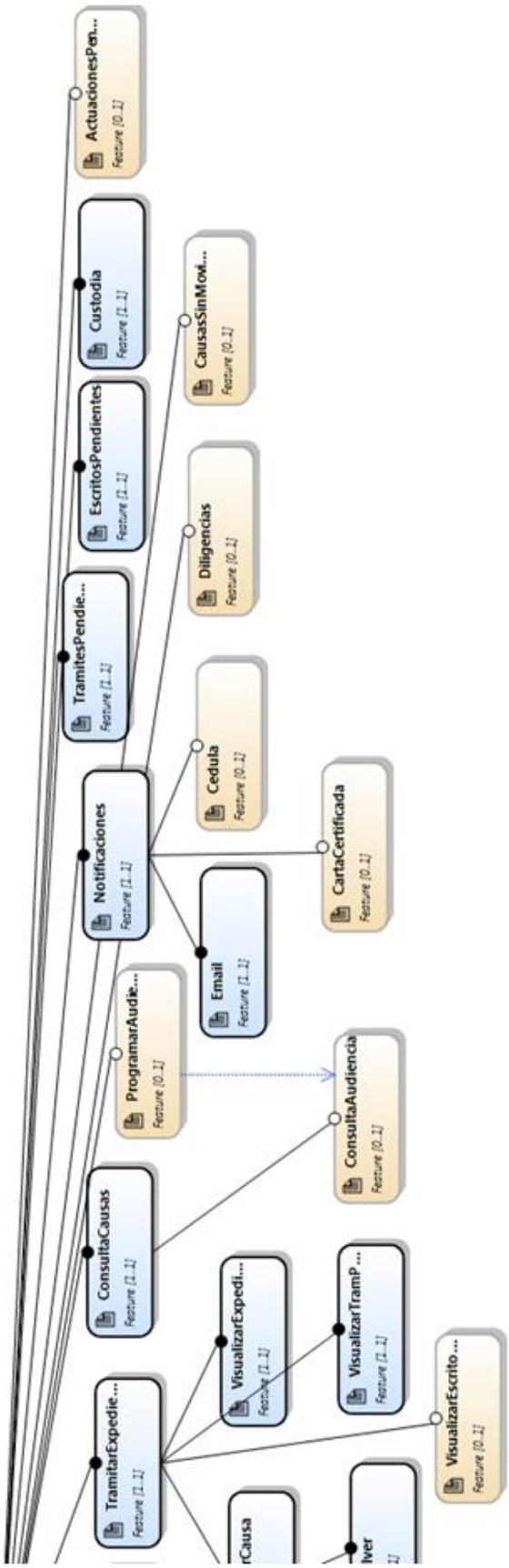


Figura 3-5: Modelo de características expediente digital (parte 2).

Dada la evolución que ha tenido hasta la fecha los diferentes sistemas de tramitación de causas ya desarrollados por la organización, se han seleccionado las características fundamentales que representan una parte importante del funcionamiento diario de una organización judicial que podría necesitarse para la tramitación de causas judiciales.

El análisis del modelo de características que se presenta a continuación es la base que se sustenta todo el proceso de diseño e implementación de la línea de productos. En la Figura 3-6, se visualiza en forma simplificada las características que cuenta la línea de productos y en la Tabla 2, se describen en qué consisten cada característica.

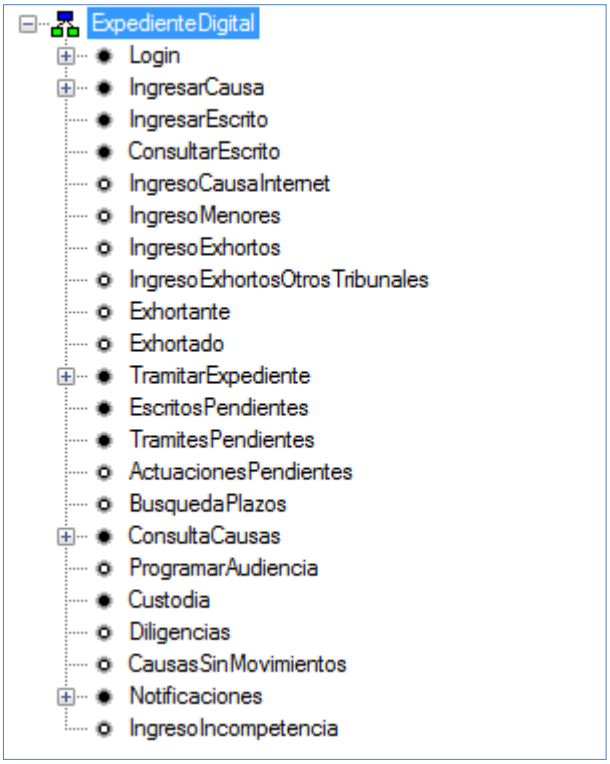


Figura 3-6: Esquema de características.

ID	Característica	Descripción
1	Ingreso de Causa	Permite realizar el registro de causas de distintas naturaleza y elementos que la conforman, tales como litigantes, procedimientos, documentos, entre otros.
2	Ingreso de Causa Internet	Permite realizar el registro de causas de distintas naturaleza a los abogados por medio de Internet.
3	Ingreso de Menores	Permite realizar el registro de causas asociadas al procedimiento de protección para la competencia de familia.
4	Ingreso Exhortos	Permite realizar el registro de exhortos derivados a la institución para su tramitación posterior.
5	Ingreso Exhortos Otros Tribunales	Permite realizar el registro de exhortos derivados de instituciones que no cuentan con un sistema informático de tramitación de causas.
6	Exhortante	Permite visualizar los exhortos enviados por la institución.
7	Exhortado	Permite visualizar los exhortos recibidos por la institución.
8	Ingreso Incompetencia	Permite realizar el registro de incompetencias derivadas a la institución para su tramitación posterior.
9	Modificación de Causa	Permite modificar los datos de principales de una causa previamente ingresada.

10	Consulta Ingreso	Permite visualizar los datos de las causas previamente ingresadas.
11	Ingreso Escritos	Permite ingresar los escritos presentados por los litigantes de las causas para que el tribunal los resuelva.
12	Tramitar Expediente	Permite dar curso y tramitar una causa, registrando los antecedentes más relevantes del proceso.
13	Escritos Pendientes	Permite generar una bandeja de trabajo para los funcionarios de la organización listando los escritos que no están resueltos por la organización.
14	Trámites Pendientes	Permite generar una bandeja de trabajo para los jueces de la organización listando los trámites que todavía no han sido firmados ni bloqueados.
15	Actuaciones Pendientes	Permite generar una bandeja de trabajo para los funcionarios de la organización listando las actuaciones que todavía no han sido firmados ni bloqueados.
16	Búsqueda de Plazos	Permite visualizar los plazos generados por trámites.
17	Causas Sin Movimientos	Permite gestionar aquellas causas que no han tenido ningún trámite durante un periodo de tiempo predeterminado.
18	Notificaciones	Permite gestionar las notificaciones emanadas por los trámites firmados o bloqueados en las diferentes causas.
19	Programar Audiencia	Permite gestionar la programación de actividades judiciales, especialmente de las audiencias que deban celebrarse dentro de la institución.
20	Custodia	Permite gestionar el registro y cuidado material de especies y documentos que eventualmente constituirán una prueba.
21	Diligencias	Permite gestionar todas aquellas diligencias emanadas de la tramitación de las causas.
22	Consulta de Causas	Permite a los usuarios del sistema y litigantes de las causas, conocer de forma rápida y dinámica, el estado de los procesos.
23	Consulta de Audiencia	Permite a los usuarios del sistema y litigantes de las causas, conocer de forma rápida y dinámica, la fecha de audiencia agendada en un tribunal.
24	Login	Permite que los usuarios inicien una sesión en el sistema ingresando su código de usuario y contraseña para autenticarse.

Tabla 2: Descripción de características.

3.4.1 Priorización de las características

Usando el diagrama de características creado en la etapa anterior con el *plug-in* FMT (5), la planificación del proyecto se realizará por iteraciones basadas en el número de características que pueden incluirse en un sistema de tramitación de causas judiciales, donde se construyó una lista de características, en la cual se identifican, agrupan, priorizan y ponderan.

Se establecieron las prioridades de las características según los siguientes criterios:

ID	Característica	A (debe estar)	B (deseable que esté)	C (deseable si se puede)	D (deseable a futuro)
1	Ingreso de Causa	X			
2	Ingreso de Causa Internet		X		
3	Ingreso de Menores			X	
4	Ingreso Exhortos	X			
5	Ingreso Exhortos		X		

	Otros Tribunales				
6	Exhortante				X
7	Exhortado				X
8	Ingreso Incompetencia		X		
9	Modificación de Causa	X			
10	Consulta Ingreso	X			
11	Ingreso Escritos	X			
12	Tramitar Expediente	X			
13	Escritos Pendientes		X		
14	Trámites Pendientes		X		
15	Actuaciones Pendientes			X	
16	Búsqueda de Plazos		X		
17	Causas Sin Movimientos				X
18	Notificaciones	X			
19	Programar Audiencia		X		
20	Custodia		X		
21	Diligencias		X		
22	Consulta de Causas	X			
23	Consulta de Audiencia			X	
24	Login	X			

Tabla 3: Priorización de características.

Según la priorización realizada la lista de característica quedo conformada por las siguientes características:

- Ingreso de Causa
- Ingreso Exhortos
- Modificación de Causa
- Consulta Ingreso
- Ingreso Escritos
- Tramitar Expediente
- Notificaciones
- Consulta Causa
- Login

3.5 Planificación por característica

Si bien, inicialmente se han identificado 24 características y priorizadas 8, sólo se implementarán 4 de ellas, las cuales son las obligatorias y factibles de implementar en este proyecto de tesis por el tiempo, esfuerzo y alcance establecido.

A partir de la lista de característica de la etapa anterior, se establecieron los hitos y cronograma para el diseño y construcción de cada característica.

Iteración	Característica	Fecha Inicio	Fecha Término
1	Ingreso de Causa	13/08/2012	24/08/2012
2	Ingreso de Escrito	27/08/2012	24/09/2012
3	Tramitar Expediente	25/09/2012	05/10/2012
4	Login	22/10/2012	31/10/2012

Tabla 4: Iteraciones del diseño y construcción.

Para generar la infraestructura que permita crear productos específicos, de una forma amigable y lo más automáticamente posible, se utilizará un lenguaje específico de dominio que modela los diagramas de características utilizando Microsoft Visual Studio 2008, DSL Tools (21) y el *plug-in* FMT (5) creado por el grupo GIRO. La idea es que el desarrollador use un lenguaje específico de dominio para modelar el sistema de tramitación de causas judiciales automatizado que mejor se adapte a las necesidades de un cliente específico. A partir de dicho modelo, esta genera de forma automática el código necesario (C#) para componer las características creadas en la fase de ingeniería del dominio, y derivar así un producto concreto, Figura 3-7.

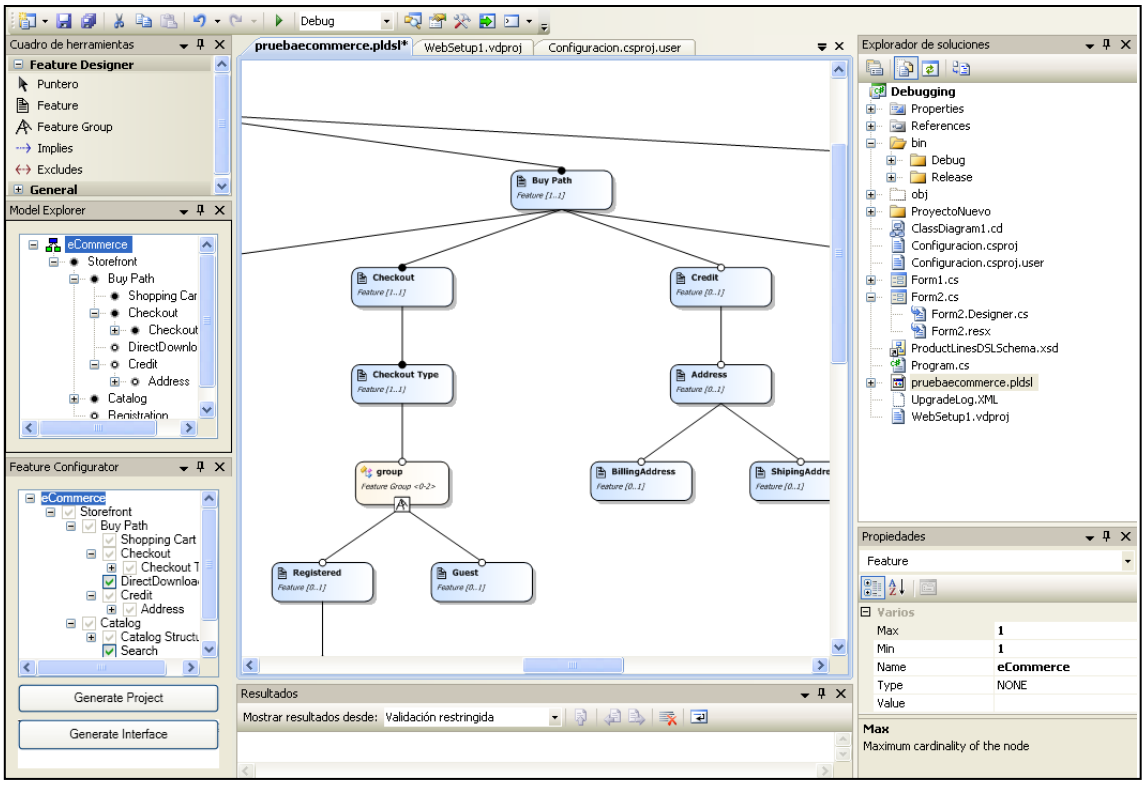


Figura 3-7: Interfaz de *plug-in* FMT.

Para cada iteración del diseño y construcción se incluirán las siguientes actividades:

- Diseño de casos de uso
- Diseño de diagramas de colaboración
- Diseño de diagrama de clases

- Diseño de la interfaz gráfica
- Implementación de clases y métodos
- Testing unitario
- Integración

Todos los diagramas son realizados con UML (22).

3.6 Diseño y construcción por características

En el diseño y construcción de cada característica, se utiliza el proceso de desarrollo dirigido por los casos de uso, pero con una versión adaptada según el esquema propuesto por Laguna y Otros (2), es decir, el análisis y diseño es dirigido por los casos de uso identificados para cada característica que fue planificada.

El esquema consiste en que cada característica identificada es un paquete, el cual contiene un conjunto de casos de uso que representan las interacciones que el usuario tiene con el sistema. Luego, se realizará un análisis a través de un diagrama de secuencia, pero sólo identificando los mensajes que llegan al sistema, para luego analizar como ese mensaje se comunica con los diferentes objetos que contiene el sistema por dentro, al plasmarlo en un diagrama de comunicación. Por último, una vez analizados todos los casos de uso del paquete, se crea el diagrama de clases que contiene los objetos y métodos analizados en el diagrama de comunicación. Este proceso se repite para las cuatro características que fueron planificadas.

En paralelo, se diseñó la arquitectura, en la cual se sustenta la creación de la línea de productos de tramitación de causas judiciales. Dicha arquitectura se basa en la web pero con un enfoque claro al dominio.

3.6.1 Arquitectura del dominio

La arquitectura del dominio se describe desde las perspectivas necesarias para profundizar lo suficiente para obtener una arquitectura robusta y que cumpla con las necesidades de la línea de productos.

Una arquitectura de software es el conjunto de decisiones sobre la organización del sistema software, la selección de los elementos estructurales y sus interfaces, con los que se compone el sistema, junto con su comportamiento tal como se especifica en las colaboraciones entre esos elementos, la composición de eso elementos estructurales y de comportamientos en subsistemas progresivamente más amplios, y el estilo de arquitectura que guía esta organización, estos elementos y sus interfaces, sus colaboraciones, y su composición (23).

El modelo propuesto para la representación de la arquitectura que adoptaran todos los productos que sean creados a partir de la línea de productos, está compuesto por el siguiente conjunto de vistas:

- **Vista de Lógica:** describe las capas y patrones arquitecturales que se usarán en el diseño y posteriormente en la implementación de la solución (ingeniería del producto), además se dan las razones del porqué de cada capa y el problema que resuelve.
- **Vista de *Deployment*:** se muestra la topología del sistema, su distribución y las pautas para su instalación. Se consideran además los requisitos funcionales del sistema tales como, disponibilidad, confiabilidad, desempeño entre otras.

3.6.1.1 Objetivos Arquitecturales

A continuación se describen los objetivos arquitecturales más relevantes y su relación con las necesidades de la línea de productos.

- **Rendimiento**: los productos creados a partir de la línea deben permitir tanto la carga como la descarga de archivos en tiempos aceptables de acuerdo con el tamaño de éstos.
- **Modificabilidad/Reuso**: los productos creados a partir de la línea deben permitir modificaciones como consecuencia de nuevos requerimientos o la evolución de los mismos.
- **Portabilidad**: los productos creados a partir de la línea deben ser capaces de trabajar con cualquier motor de base de datos, sin alterar sus componentes de negocio.
- **Usabilidad:** El diseño debe ser orientado por y para la comodidad del usuario y el funcionario de la organización, de manera que la interfaz sea intuitiva y fácil de manejar, al mismo tiempo que se fomenté altamente la interacción entre ambos. De la misma forma, el usuario debe tener la capacidad de equivocarse y regresar a un estado seguro en el que se le permita cumplir con su objetivo original sin que se le haga tedioso o complicado el proceso para llegar ha dicho fin.

3.6.1.2 Vista Lógica

El primer nivel se especifica el patrón de arquitectura para dominio. El mismo está organizado utilizando el patrón de arquitectura N-Capas con Orientación al Dominio. Las capas son agrupaciones horizontales lógicas de componentes de software que forman el sistema. Ayudan a diferenciar entre los diferentes tipos de tareas a ser realizadas por los componentes, ofreciendo un diseño que maximiza la reutilización y especialmente la mantenibilidad (24). La ventaja al dividir en capas el sistema es que permite diferentes tipos de despliegue y proporciona una delimitación clara y situación de donde debe estar cada tipo de componente funcional. Las características del patrón de arquitectura N-Capas son:

- Descomposición de los servicios de forma que la mayoría de interacciones ocurre sólo entre capas vecinas.
- Las capas del sistema pueden residir en la misma máquina o pueden estar distribuidos entre varios equipos.
- Los componentes de cada capa se comunican con los componentes de otras capas a través de interfaces bien conocidos.

- Cada nivel agrega las responsabilidades y abstracciones del nivel inferior.
- Separa de forma clara la funcionalidad de cada capa.

En la Figura 3-8 se visualiza el patrón de arquitectura N-Capas implementado.

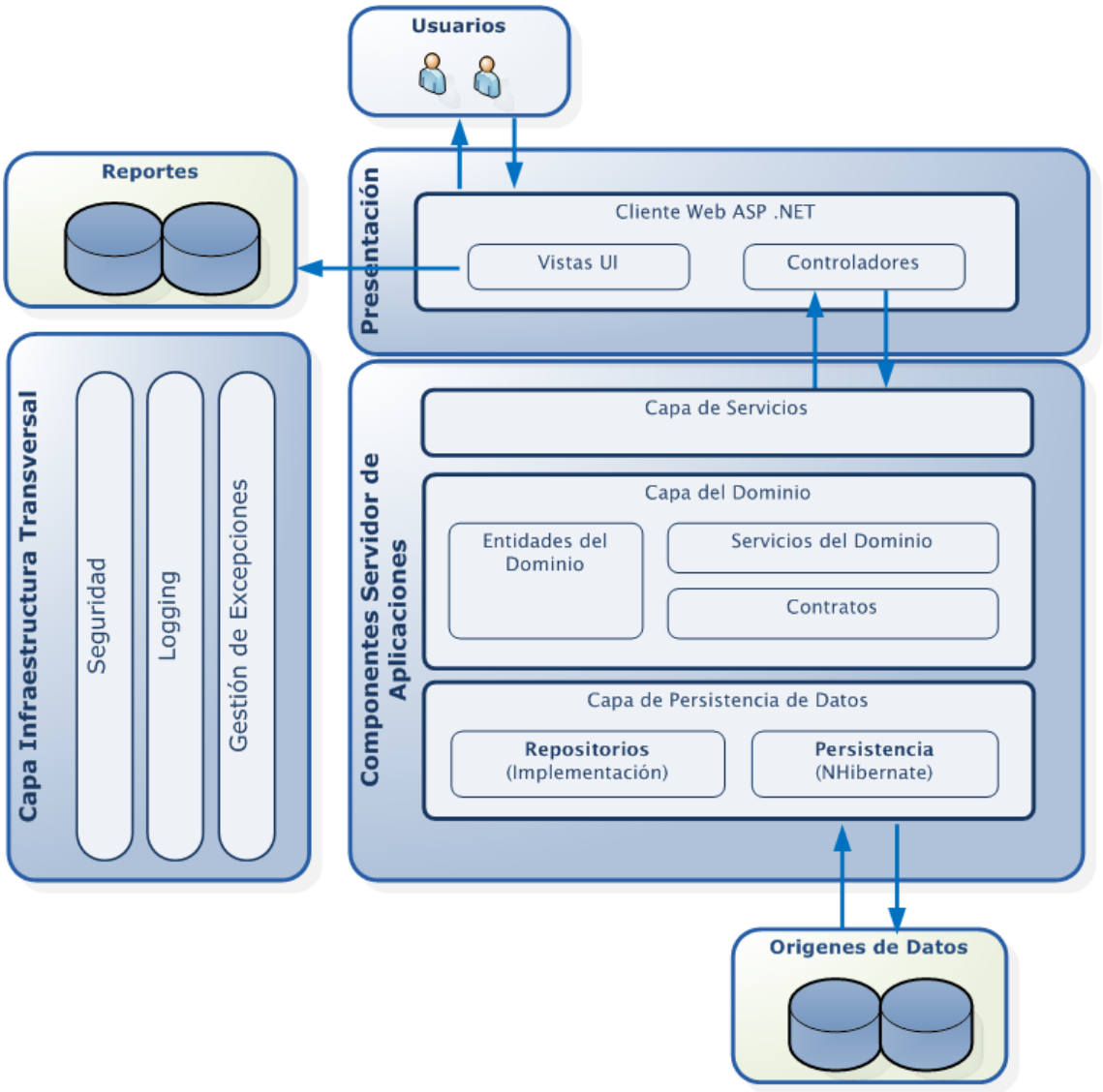


Figura 3-8: Arquitectura n-capas con orientación al dominio.

A continuación se describen las capas que componen la arquitectura diseñada:

- Capa de Presentación: Esta capa es responsable de mostrar información al usuario e interpretar sus acciones. Los componentes de la capa de presentación implementan por lo tanto la funcionalidad requerida para que los usuarios interactúen con el sistema.
    - Subcapa de Componentes Visuales (Vistas UI): Estos componentes proporcionan el mecanismo base para que el usuario utilice la aplicación. Por lo tanto, son componentes que formatean datos en cuanto a tipos de letras y controles visuales, y también reciben datos proporcionados por el

usuario. Para su implementación se utilizarán páginas ASP.NET y componentes AJAX (25).

- Subcapa de Controladores: Para ayudar a sincronizar y orquestar las interacciones del usuario, puede ser útil conducir el proceso utilizando componentes separados de los componentes propiamente gráficos. Esto impide que el flujo del proceso y lógica de gestión de estados esté programado dentro de los propios controles y formularios visuales y permite reutilizar dicha lógica y patrones desde otros interfaces o "vistas". También es muy útil para poder realizar pruebas unitarias de la lógica de presentación.

- Capa de Servicios: Proporciona un medio de acceso basado en canales de mensajes de datos. Es importante destacar que esta capa debe ser lo ligera posible y no debe incluir nunca lógica de negocio.
- Capa del Dominio: Esta capa debe ser responsable de representar conceptos de negocio, información sobre la situación de los procesos de negocio e implementación de las reglas del dominio. También debe contener los estados que reflejan la situación de los procesos de negocio, aun cuando los detalles técnicos de persistencia se delegan a las capas de infraestructura (Repositorios, etc.). Esta capa es el corazón del software, los componentes implementan la funcionalidad principal del sistema y encapsulan toda la lógica de negocio relevante. Siguiendo los patrones de arquitecturas N-Capas con orientación al dominio, esta capa tiene que ignorar completamente los detalles de persistencia de datos. Estas tareas de persistencia deben ser realizadas por las capas de infraestructura.
  - Entidades del dominio: Estos objetos son entidades de datos desconectados y se utilizan para obtener y trasferir datos de entidades entre las diferentes capas. Adicionalmente contienen lógica del dominio relativo al contenido de entidad, por ejemplo, validaciones de datos, campos pre-calculados, relaciones con otras sub-entidades, etc. Estos datos representan entidades de negocio del dominio implementado.
  - Servicios del dominio: En las capas del dominio, los servicios son básicamente clases agrupadoras de comportamientos y/o métodos con ejecución de lógica del dominio. Estas clases normalmente no deben contener estados relativos al dominio y son las clases que coordinen e inicien operaciones compuestas que a su vez invoquen a objetos de capas inferiores (como persistencia de datos).
  - Contratos de Repositorios: Aun cuando la implementación de los Repositorios no es parte del Dominio sino parte de las capas de Infraestructura (los Repositorios están ligados a una tecnología de persistencia de datos, como un ORM), sin embargo, el "contrato" (interfaz) de cómo deben estar construidos dichos Repositorios, es decir, los interfaces de los Repositorios, estos si deben formar parte del Dominio,

puesto que dicho contrato especifica qué debe ofrecer el Repositorio para que funcione y se integre correctamente con el Dominio, sin importar como está implementado por dentro.

- Capa de Infraestructura de Acceso a Datos: Esta capa proporciona la capacidad de persistir datos así como lógicamente acceder a ellos. Pueden ser datos propios del sistema o incluso acceder a datos expuestos por sistemas externos (Servicios Web externos, etc.). Esta capa de persistencia de datos expone el acceso a datos a las capas superiores, como la del dominio. Esta exposición debe realizarse en forma desacoplada.
    - Repositorios: A nivel genérico, un Repositorio "representa todos los objetos de un cierto tipo como un conjunto conceptual". A nivel práctico, un repositorio es una clase encargada de realizar las operaciones de persistencia y acceso a datos, estando ligado por lo tanto a una tecnología concreta (NHibernate). Se debe crear un *Repository* por cada "entidad del dominio". El acceso a un Repositorio debe realizarse mediante un interfaz bien conocido, un contrato "depositado" en el dominio, de forma que podríamos llegar a sustituir un Repositorio por otro que implemente otras tecnologías y sin embargo, la capa del dominio no se vería afectada.
    - Componentes Base: La mayoría de las tareas de acceso a datos requieren cierta lógica común que puede ser extraída e implementada en un componente separado reutilizable. Esto ayuda a simplificar la complejidad de los componentes de acceso a datos y sobre todo minimiza el volumen del código a mantener.

Para la implementación de la capa de infraestructura de acceso a datos se decidió utilizar el motor de base de datos relacional Microsoft SQL Server 2005, dada la experiencia en múltiples proyectos se posee en este motor. Por encima de éste se ubica la tecnología NHibernate. NHibernate es una herramienta de mapeo objeto-relacional (ORM) (26) para la plataforma .NET que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones (27). Al usar NHibernate para el acceso a datos, se logra que el sistema sea agnóstico en cuanto al motor de base de datos a utilizar, ya que NHibernate soporta los más utilizados por la industria, tales como: MySQL, PostgreSQL, Oracle, MS SQL Server, entre otros (28).

Como todas las herramientas de su tipo, NHibernate busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en un sistema: el usado en el servidor de aplicaciones (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al sistema detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información NHibernate le permite al sistema manipular los datos en la base de datos operando sobre objetos (clases). NHibernate convierte los datos entre los tipos utilizados por

C# y los definidos por SQL. Genera las sentencias SQL y libera al sistema del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos.

Los servicios que proveen la capa de infraestructura de acceso a datos son los presentados en la Figura 3-9.

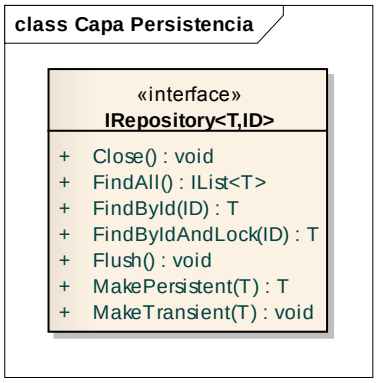


Figura 3-9: Servicios de persistencia.

- Capa de Infraestructura Transversal/Horizontal: Proporcionan la capacidad técnica genérica que dan soporte a capas superiores. Son “bloques de construcción” ligados a una tecnología concreta para desempeñar sus funciones. Los aspectos horizontales (Transversales) implementan por lo tanto tipos específicos de funcionalidad que pueden ser accedidos/utilizados desde componentes de cualquier capa. Los diferentes tipos/aspectos horizontales son: Seguridad (Autenticación, Autorización y Validación), tareas de gestión de operaciones (políticas, loggins, trazas, monitorización, configuración, etc.).

  Para la implementación de la capa de infraestructura transversal/horizontal se utiliza el framework log4net que permite elegir la salida y nivel de granularidad de los “logs”. Esta información es útil a la hora de solucionar un error en ambiente productivo o de desarrollo.

- Reportes: Para realizar consultas cuyo objetivo es únicamente visualizar reportes, se podría hacer uso de las mismas clases de lógica del dominio y repositorios de acceso de datos relacionados que se utilizan para las operaciones transacciones, sin embargo, si se requiere la máxima optimización y rendimiento, probablemente ésa no sea la mejor opción. En definitiva, mostrar información al usuario no está ligado a la mayoría de los comportamientos del dominio (reglas de negocio), ni a problemáticas de tipo concurrencia en las actualizaciones, entre otras. Todas esas problemáticas impactan en definitiva en el rendimiento puro de las consultas y lo único que se requiere es hacer en este caso realizar consultas con muy buen rendimiento. Por ende, se tomó la decisión de utilizar la herramienta Microsoft SQL Server Reporting Services 2005 para optimizar los reportes y de esta forma el acceder a ellos sea un mecanismo más sencillo y ligero. En definitiva, el objetivo

final es *"situar todo el código en cada parte adecuada del sistema, de una forma granularizada, focalizada y que se pueda probar de forma automatizada"* (24).

3.6.1.3 Vista *deployment*

La vista de *deployment* presenta la estructura física necesaria para la implementación de los productos creados a partir de la línea de productos, la cual está basada en el patrón arquitectónico N-Capas que separa la capa de presentación con la lógica de negocios.

La Figura 3-10 muestra los nodos requeridos para desplegar los productos concretos. El Servidor de Aplicaciones contiene los componentes de presentación (IU) y las capas de negocio y acceso a datos.

El servidor de base de datos contiene la base de datos SQL Server 2005, la cual tendrá acceso centralizado y único por el servidor de aplicaciones.

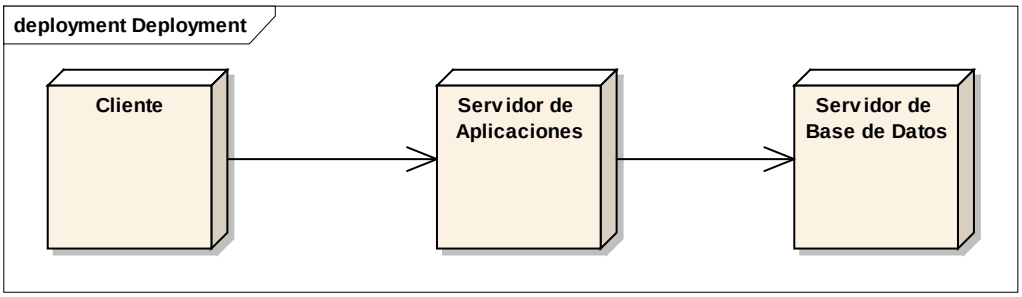


Figura 3-10: Vista de *Deployment*

Por ende la arquitectura física del dominio se compone de los siguientes componentes:

- Cliente: PC con browser (Internet Explorer, Google Chrome y FireFox).
- Servidor de Aplicaciones: Microsoft Windows Server 2008, Microsoft Internet Information Server 7.0 (IIS), Framework .NET 3.5
- Servidor de Base de Datos: Microsoft Windows Server 2008, Microsoft SQL Server 2005. El motor de base de datos puede ser reemplazado por cualquier otro que sea soportado por NHinernate, de esta manera existe cierta flexibilidad para seleccionar a un proveedor de bases de datos distinto.

3.6.2 Iteración 1: Ingreso de causa

Para esta iteración se diseñó y construyó la característica ingresar causa. Esta característica tiene como principal funcionalidad permitir registrar todos los antecedentes más relevantes de la causa, tales como fecha de ingreso, caratulado, archivos digitales e información de las partes, es decir, los datos de quienes forman parte del litigio (nombre, sexo, número identificador, dirección, teléfono, correo electrónico, entre otros) y el rol que cada uno cumplirá (demandante, demandado, abogado demandante, abogado demandado, entre otros), además se debe especificar el tipo de flujo de trabajo que deberá seguir la causa para su tramitación posterior.

3.6.2.1 Combinación de paquetes en UML y representación de la variabilidad

Con el objetivo de que mantener la trazabilidad y conectar los diagramas de casos uso, secuencia, modelo de diseño e implementación, se expresa la variabilidad utilizando el concepto de combinación de paquetes (*package merge*) de UML. Miguel Laguna y Bruno González (2), detallado en la sección 2.5.1.

La aplicación a líneas de productos consiste en establecer en el modelo arquitectónico un paquete raíz que recoge la parte común de la línea de productos (incluyendo modelos estructurales, de casos de uso y de interacción). Junto a este paquete común se añade un paquete por cada característica opcional, de modo que queden localizadas en ese paquete todas las modificaciones necesarias en el modelo de diseño asociadas a esa característica. El paquete añadido se conecta mediante la relación <<merge>> con su paquete base en el punto preciso de la jerarquía de paquetes. La aplicación de este esquema se visualiza en la Figura 3-11 donde la característica opcional ingresar forma inicio implica la existencia del correspondiente paquete en el modelo de casos de uso y diseño. Por otro lado, mediante el uso de clases parciales organizadas en paquetes, se puede establecer una correspondencia directa entre los modelos de casos de uso y diseño hasta la implementación de la línea de productos.

Como beneficio, se obtiene el localizar en un sólo paquete del modelo todas las variaciones que origina cada característica opcional manteniendo una correspondencia uno a uno, se separa claramente los dos tipos de variabilidad, eliminando las ambigüedades. La variabilidad de la línea de producto viene dada por la selección de paquetes. La variabilidad de los productos concretos está dentro de cada paquete y se mantiene sin cambios el meta-modelo de UML (2).

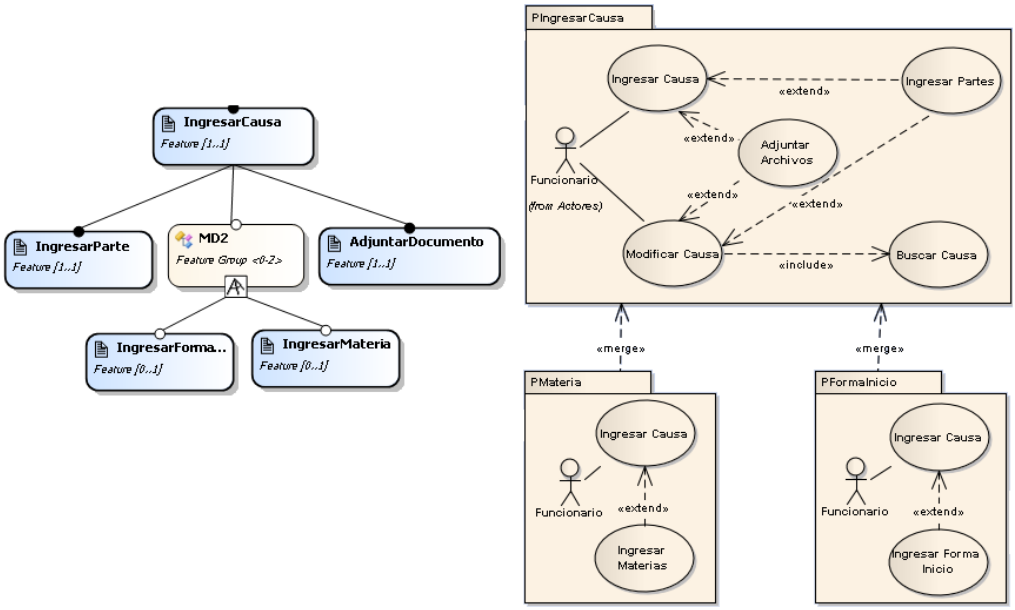


Figura 3-11: Modelo de Característica de la Iteración 1 y el mecanismo de "*package merge"* aplicado al modelo de casos de uso.

3.6.2.2 Análisis y diseño de la característica ingresar causa

La característica obligatoria ingresar causa, es indispensable en los sistemas de tramitación de causas judiciales, desde aquí parte todo el flujo de negocio que soportan estos sistemas. En el modelo de característica, se detalla la variabilidad que puede adoptar la característica para un producto concreto de tramitación de causas judiciales, indicando claramente las que son obligatorias y agrupadas en alternativas. Las obligatorias son: ingreso de parte y adjuntar documento, mientras que las agrupadas en alternativas (o grupos OR) son: ingresar forma de inicio e ingresar materias, se debe elegir una opción (o más de una) entre esas dos.

En la Figura 3-12 se representa la jerarquía de la característica ingresar causa.

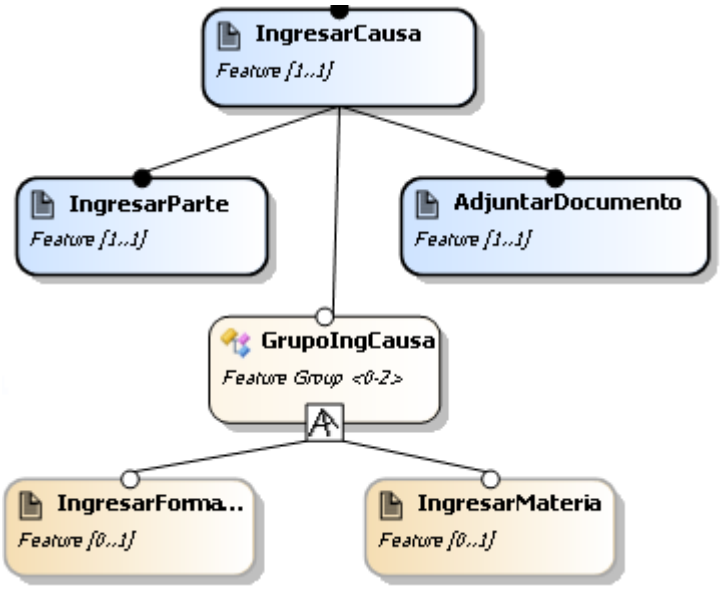


Figura 3-12: Esquema de la característica ingresar causa.

El modelo básico de la característica ingreso de causa de la línea de productos se visualiza en la Figura 3-13. Adoptando la propuesta Miguel Laguna y Bruno González (2), los paquetes de la parte inferior son opcionales e independientes entre sí. Además del paquete base, para esta iteración se desarrollaron los siguientes paquetes:

- **PIngresarCausa**: Contiene las funcionalidades de ingresar, modificar y buscar una causa, además de adjuntar documentos e ingresar las partes que conforman el proceso.
- **PFormaInicio**: Contiene la funcionalidad de seleccionar la forma de inicio que adoptara el procedimiento seleccionado para una causa al momento de su ingreso.
- **PMateria**: Contiene la funcionalidad de agregar las materias asociadas a una causa cuando se ingresa o modifica.

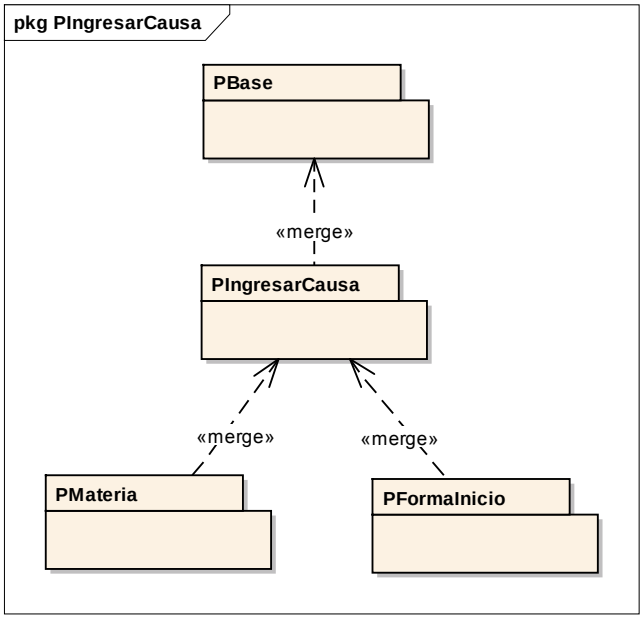


Figura 3-13: Esquema de jerarquía del paquete ingresar causa.

Este esquema de combinación de paquetes se utiliza en todas las iteraciones del diseño y construcción de la línea de productos, incluyendo los modelos de análisis (casos de uso y secuencia del sistema), diseño (diagramas de comunicación y de clases) e implementación (paquetes de clases parciales junto con ficheros HTML o XML). En la Figura 3-14 se visualiza el modelo de casos de uso de la característica ingresar causa. Si se selecciona el paquete forma inicio, se incluye en el modelo un caso de uso adicional como extensión que permite seleccionar la forma de inicio que tendrá el procedimiento judicial que adoptara la causa, además al seleccionar el paquete materia, se incluye también en el modelo de caso de uso adicional como extensión que permite agregar una o varias materias a la causa, de tal manera de tipificar los asuntos por los cuales se llegó a la controversia entre las partes.

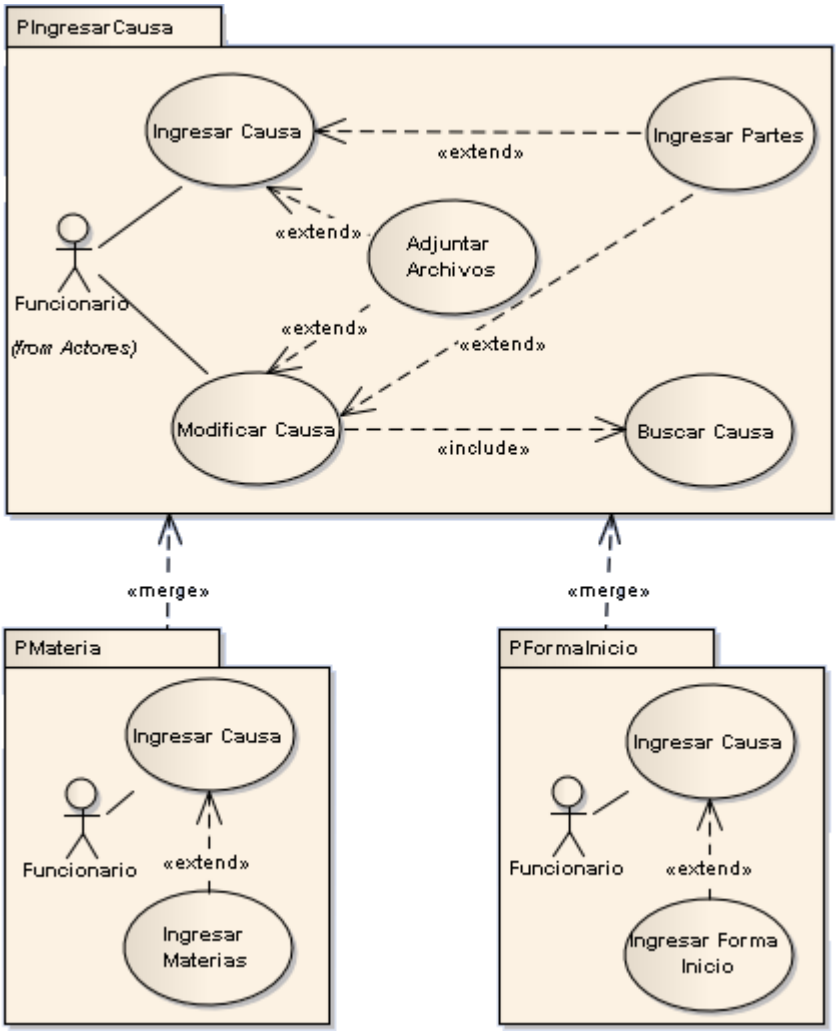


Figura 3-14: Paquetes de la característica ingreso de causa (vista casos de uso).

A continuación se desarrollan los paquetes identificados de la característica manteniendo el esquema del desarrollo dirigido por casos de uso.

3.6.2.3 Paquete ingresar causa

El paquete ingresar causa se encarga de la funcionalidad de realizar el registro de causas de distintas naturaleza y elementos que la conforman, tales como partes que intervienen en el proceso judicial, procedimientos y documentos. Esta característica es obligatoria y se mezcla con el paquete base.

3.6.2.3.1 Caso de uso: ingresar causa

Este caso de uso comienza cuando el funcionario ingresa una nueva causa al sistema.

Flujo de Eventos Principal	
	1. El funcionario selecciona la opción ingresar causa. 2. El sistema carga los procedimientos existentes. 3. El usuario selecciona un procedimiento. 4. El usuario indica la Fecha de ingreso y el caratulado de la causa. 5. El usuario ingresa los datos en la ficha de la causa.

	6. El usuario graba la causa. 7. El sistema verifica la completitud de los campos obligatorios. 8. El sistema genera y muestra el rol único asignado a la causa. 9. El sistema muestra mensaje indicando que la causa fue ingresada exitosamente.
Alternativo Adjuntar archivos	10 El funcionario solicita agregar documentos a la causa. Se ejecuta el caso de uso "Adjuntar archivos".
Alternativo Ingresar Partes	10 El funcionario solicita ingresar las partes de la causa. Se ejecuta el caso de uso "Ingresar Partes".
Alternativo Ingresar Materias	10. El funcionario solicita ingresar las materias de la causa. Se ejecuta el caso de uso “Ingresar Materia” del paquete Materia.
Precondición	El usuario debe tener una sesión válida.
Invariante Rol de la causa	El rol de la causa debe ser correlativo y generado automáticamente por el sistema. Se establecerá una letra diferenciadora antes del Rol, ello dependerá del procedimiento seleccionado.

Un diagrama de secuencia del sistema es un artefacto creado de manera rápida y fácil que muestra los eventos de entrada y salida relacionadas con el sistema que se está estudiando. Antes de continuar con el diseño lógico de cómo funcionará el sistema, es conveniente analizar y definir su comportamiento como una “caja negra”. El comportamiento del sistema es una descripción de *qué* hace el sistema, sin explicar cómo lo hace (20).

En general, un diagrama de secuencia puede ser de dos tipos diferentes: El tipo genérico documenta todos los escenarios posibles en el diagrama de secuencia, mientras que el tipo de instancia documento un sólo escenario (23). Documentar la variabilidad en los diagramas de secuencia implica la utilización del tipo genérico (29).

El diagrama de secuencia del sistema de la Figura 3-15 muestra el curso de eventos específicos del caso de uso ingresar causa, el actor que interacciona directamente con el sistema, el sistema como una caja negra y los eventos del sistema que genera el actor. El tiempo avanza hacia abajo, y la ordenación de los eventos sigue su orden en el caso de uso. Se muestra el escenario principal de éxito del caso de uso. Se indica que el funcionario genera los eventos del sistema *listarProcedimiento, listarMateria, listarFormaInicio, crearCausa* y referencia los diagramas de secuencia de los casos de uso Ingresar Partes y Adjuntar Archivos, ambos pueden ejecutarse tantas veces como el funcionario desee. Los eventos *listarMateria* y *listarFormaInicio* son alternativos y se incluye solamente en caso de seleccionar la características de ingresar materia e ingresar forma inicio.

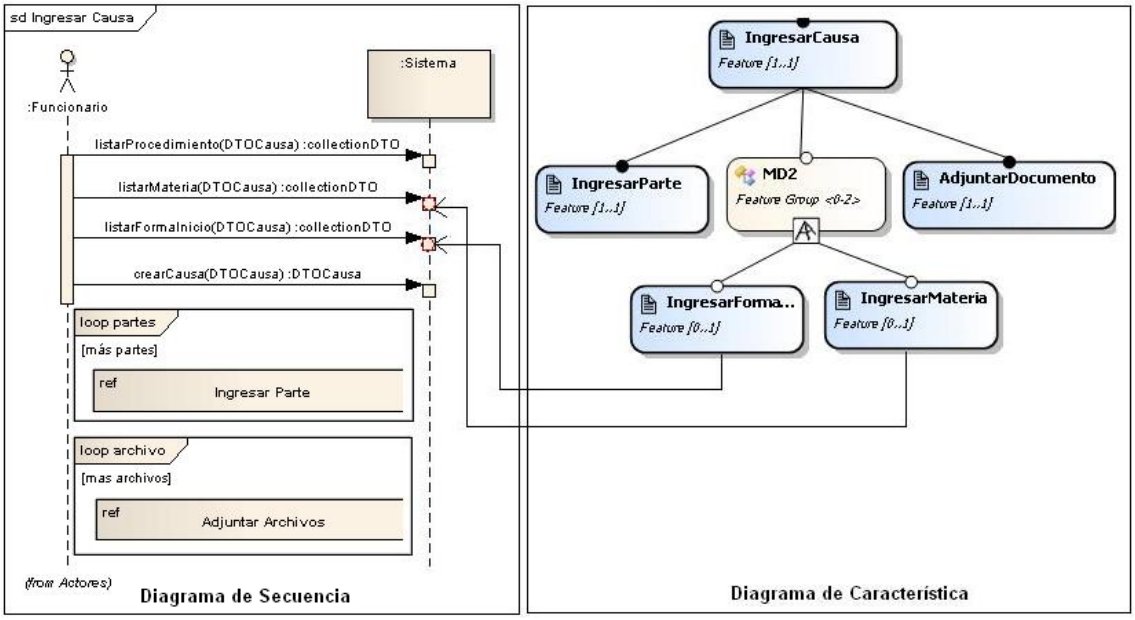


Figura 3-15: Diagrama de secuencia del caso de uso ingresar causa.

3.6.2.3.2 Caso de uso: ingresar partes

Este caso de uso comienza cuando el funcionario ingresa, modifica o elimina los litigantes involucrados en la causa.

Flujo de Eventos Principal	
	1. El sistema despliega los tipos de partes según el procedimiento seleccionado. 2. El funcionario selecciona el tipo de parte que desea ingresar. 3. El funcionario ingresa los datos de la parte. Los datos disponibles son: - Tipo de Persona - Nacionalidad - RUT - Nombres - Apellido Paterno - Apellido Materno - Calle-número - País - Ciudad - Comuna - Teléfono - Fax - Email - Otros antecedentes 4. El sistema despliega los datos del representante legal si el usuario selecciona tipo de persona “Jurídica”: - RUT - Nombres - Apellido Paterno - Apellido Materno 5. El funcionario graba la parte ingresada. 6. El sistema verifica la completitud de los datos obligatorios. Los datos

	obligatorios son: - Tipo parte - Tipo persona - Nacionalidad - Nombres - Apellido Materno - Apellido Paterno - Rut - Calle - número - País - Ciudad - Comuna 7. El sistema registra la parte y lo asocia a la causa. 8. El sistema muestra el mensaje indicando que la parte se agregó exitosamente.
Excepción Parte duplicada	7.1 La nueva parte ya existe en la lista de partes de la causa. 7.2 Retorna al paso 1
Precondición	El funcionario debe tener una sesión valida.

3.6.2.3.3 Caso de uso: buscar causa

Este caso de uso comienza cuando un usuario del sistema busca una causa, según los criterios especificados, para consultar por esa causa, ya sea para hacer una modificación, ver el expediente digital o subir escritos y documentos. Para esto selecciona un criterio de búsqueda disponible e ingresa el texto correspondiente. El sistema, en caso de encontrar causas que cumplan con dicho criterio, muestra un listado de las mismas.

Flujo de Eventos	
Principal	1. El sistema despliega los criterios de búsqueda disponibles. Estos son: rol, fecha de ingreso, RUT, razón social, nombre, apellido paterno, apellido materno, nombre del juez. 2. El usuario selecciona un criterio de búsqueda e ingresa los datos para la búsqueda correspondiente. 3. El usuario solicita realizar la búsqueda según el criterio seleccionado. 4. El sistema verifica que los datos sean válidos. 5. El sistema obtiene un listado con las causas a las cuales el usuario tiene acceso y que cumplen con los criterios especificados en la búsqueda. 6. El sistema despliega el listado con los siguientes campos: • Rol • Fecha Ingreso • Carátula • Etapa • Estado procesal • Juez • Procedimiento • Ver (link para ingresar al expediente digital) • Ingresar Escrito (link para ingresar un nuevo escrito a la causa) 7. El usuario selecciona la causa con la cual va a trabajar.

Excepción Causas del usuario	1.1 El sistema consulta el perfil del usuario conectado. 1.2 Si el usuario tiene el perfil de juez, abogado o parte el sistema consulta las causas en las cuales el usuario es litigante. 1.3 El sistema obtiene un listado con las causas a las cuales el usuario tiene acceso. 1.4. El sistema despliega el listado con los siguientes campos: • Rol • Fecha ingreso • Carátula • Etapa • Estado procesal • Juez • Procedimiento • Ver (link para ingresar al expediente digital) • Ingresar Escrito (link para ingresar un nuevo escrito a la causa) 1.5 Retorna al punto 7.
Excepción No hubo resultados	En el punto 5 el sistema no encontró resultados según los criterios de búsqueda. 5.1 El sistema muestra un mensaje que no se logró obtener resultados de la búsqueda. 5.2 Retorna al paso 2.
Precondición	El funcionario debe tener una sesión valida.
Invariante Link columna Ver	Si el usuario conectado tiene el perfil de Juez o Funcionario, la columna “Ver” permite al usuario visualizar los datos de la causa e ingresar trámites a la causa. Incluye casos de uso “Visualizar Causa” y “Resolver” del paquete Tramitar Expediente. Si el usuario conectado tiene el perfil de Abogado o Parte, la columna “Ver” permite al usuario ver el expediente digital de la causa. Incluye caso de uso “Visualizar expediente” del paquete Tramitar Expediente.
Invariante Link columna Ingresar escrito	La columna ingresar escrito es visible sólo si el perfil de usuario conectado es abogado. En caso contrario esta columna no es visible. Esta columna permite al usuario abogado ingresar un nuevo escrito a la causa seleccionada. Incluye caso de uso “Ingresar Escrito” del paquete Ingresar Escrito.

3.6.2.3.4 Caso de uso: modificar causa

Este caso de uso comienza cuando el funcionario modifica información asociada a la causa.

Flujo de Eventos	
Principal	1. El usuario selecciona la opción Modificar Causa en el sistema. 2. El usuario busca y selecciona la causa que desea modificar. Incluye caso de uso "Buscar Causa". 3. El sistema muestra los datos actuales de la causa. 4. El usuario realiza las modificaciones en los campos de la causa. 5. El usuario solicita grabar los cambios realizados. 6. El sistema actualiza los registros de la causa.
Alternativo Adjuntar archivos	7 El usuario selecciona la opción adjuntar archivo. Incluye caso de uso "Adjuntar archivos".
Alternativo	7 El usuario selecciona ingresar un nuevo litigante. Se ejecuta caso de

Ingresar Partes	uso "Ingresar partes".
Alternativo Agregar o quitar Materias	7. El funcionario solicita agregar o quitar materias de la causa. Se ejecuta el caso de uso “Ingresar Materias” del paquete Materia.
Precondición	El funcionario debe tener una sesión valida.

3.6.2.4 Paquete ingresar forma inicio

El paquete ingresar forma inicio se encarga de la funcionalidad de listar los diferentes puntos de inicio que pueden adoptar un procedimiento judicial. Esta característica es opcional y si es seleccionada se mezcla con el paquete ingresar causa.

3.6.2.4.1 Caso de uso: ingresar forma inicio

Flujo de Eventos	
Principal	1. El sistema lista las formas de inicio asociadas al procedimiento. 2. El usuario selecciona la forma de inicio que debe adoptar el procedimiento.
Precondición	El funcionario debe tener una sesión valida y estar en el contexto de la funcionalidad ingreso de causa.

3.6.2.5 Paquete materia

El paquete materia se encarga de la funcionalidad de listar las materias que pueden ser asociadas a una causa. Estas materias dependen del procedimiento seleccionado previamente. Las materias son un esquema de tipificar la disputa legal que se está discutiendo en la controversia de las partes. Esta característica es opcional y si es seleccionada se mezcla con el paquete ingresar causa.

3.6.2.5.1 Caso de uso: ingresar materias

Flujo de Eventos	
Principal	1. El sistema lista las materias asociadas al procedimiento. 2. El usuario selecciona la materia que desea agregar a la causa. 3. El sistema agrega la materia seleccionada a la causa. 4. El sistema muestra mensaje indicando que la materia fue ingresada exitosamente a la causa.
Precondición	El funcionario debe tener una sesión valida y estar en el contexto de la funcionalidad ingreso de causa.

3.6.2.5.2 Diagrama de clases

Un diagrama de clases representa las especificaciones de las clases e interfaces software en una aplicación (20). A diferencia de las clases conceptuales del modelo de dominio creado por la línea de productos, las clases de diseño de los diagramas de clases muestran las definiciones de las clases software en lugar de los conceptos del mundo real.

Una vez terminados todos los diagramas de comunicación para la realización de los casos de uso del paquete Ingresar Causa, Materia y Forma Inicio, se identificaron las especificaciones de las clases e interfaces que participan cuando la característica ingresar causa es seleccionada.

En la Figura 3-16 se visualiza el diagrama de clases de los paquetes que permite capturar todas las clases parciales, visualizando los métodos posibles que pueden tener cada clase cuando es seleccionada la característica ingresar causa y mezclada con los paquetes forma inicio y materia. Al aplicar el mecanismo *package merge* (2), el resultado de combinar estos paquetes es que aparecen por ejemplo una clase final Causa con los atributos y métodos de cada paquete y una asociación con la clase tipoFormaInicio y una composición a la clase Materia.

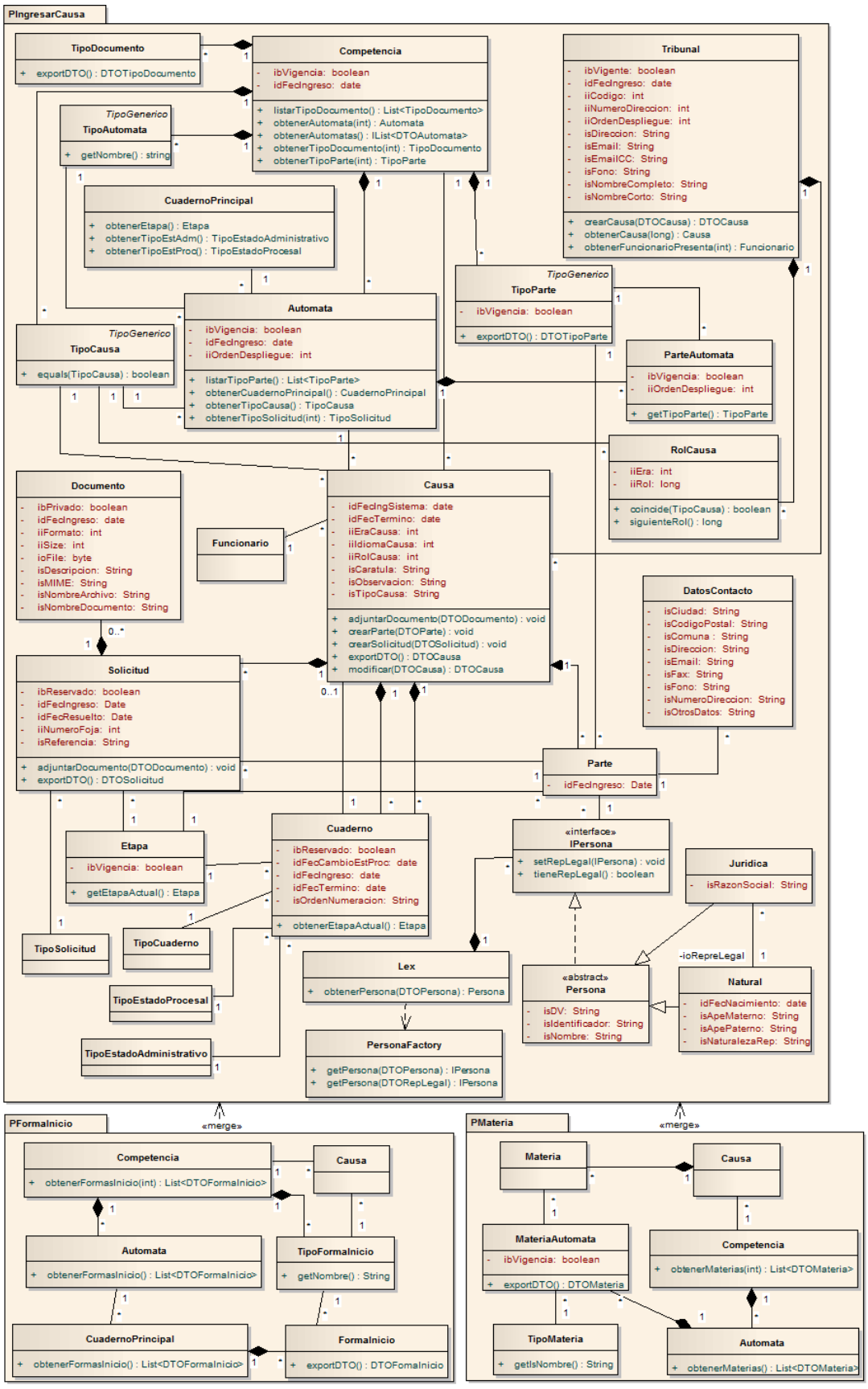


Figura 3-16: Mecanismo de "*package merge*" aplicado al diagrama de clases de la iteración I.

3.6.2.6 Implementación

La implementación de las características de la línea de productos de tramitación de causas judiciales consiste en traducir el diseño realizado al código de un lenguaje de programación que para el caso es C#. El entorno de desarrollo utilizado es Microsoft Visual Studio 2008 y con el *plug-in* FMT (5).

3.6.2.6.1 Variabilidad en la capa de presentación

En esta primera iteración se creó una página maestra (*master.page*) que proporciona una estructura y elementos comunes para todas las paginas ASPX, como la cabecera, menú y píe de página. Al crear este tipo de página mejoramos la mantenibilidad de la línea de productos al evitar duplicar innecesariamente el código de estructuras o comportamientos de las páginas que son compartidas. La página maestra se almacenó en el paquete base por ser común a todas las características de la línea de productos, además contiene una hoja de estilo *CCS* y un control *ContentPlaceHolder*, el cual define una región de la representación de la página maestra que puede sustituirse por el contenido de una página asociada a la maestra. Todas las páginas ASPX, de cada una de las características de la línea de productos, estarán asociadas en la página maestra. En la Figura 3-17 se visualiza la página maestra que está formada por un encabezado, menú, pie de página y una parte central (destacada en el recuadro rojo) que es dinámica dependiendo de las características seleccionadas.

Figura 3-17: Diseño de la página maestra de la línea de productos software.

Luego de crear la página maestra se comenzó a diseñar las páginas de cada una de las características de la primera iteración. La característica principal Ingresar Causa

contiene una página ASPX donde se ingresan los datos propios de la causa, una sección para el ingreso de las partes y otra para el ingreso de los documentos asociados a la causa. Sin embargo, existen dos características que son opcionales ingresar materia e ingresar forma de inicio, lo cual se traduce en que la página ASPX de la característica ingresar causa puede incluir el código HTML de uno o ambas características y en caso de no seleccionar ninguna de las dos, excluirlo al derivar en un producto concreto.

Para dar solución a la variabilidad de la interfaz de usuario de esta primera iteración, se utilizó un control de usuario para cada característica opcional. Estos controles son personalizados y reutilizables, se pueden crear con las mismas técnicas que se aplican a una página ASPX y también incluyen código fuente C#. De esta forma, se obtiene la misma página ASPX con más o menos funciones dependiendo de las características que se añadan de acuerdo con las necesidades que especifiquen los clientes. Por ejemplo, en la Figura 3-18 se visualiza que la característica ingresar causa no incorpora las características opcionales ingresar materia ni tampoco ingresar forma inicio. De esta manera, al ingresar una causa al sistema no contendrá una tipificación de los asuntos que se disputen y el procedimiento adoptará la forma de inicio por defecto que previamente fue configurado.

Figura 3-18: Página LEX_ING_IngresarCausa.aspx, característica ingresar causa.

Mientras que en la Figura 3-19 se visualiza la inclusión de la característica ingresar forma de inicio e ingresar materia destacadas en los recuadros rojos. Esta combinación se realiza con la aplicación de controles de usuario (*UserControl*) en tiempo de ejecución, obteniendo como ventaja separar todos los paquetes y poder reutilizarlos nuevamente tantas veces como se necesite.

Figura 3-19: Página LEX_ING_IngresarCausa.aspx seleccionado la característica ingresar forma inicio e ingresar materia.

Para añadir en tiempo de ejecución los controles de usuario se utiliza un control de ASP.NET llamado *PlaceHolder* que actúan como contenedores de los controles de usuario. En la página LEX_ING_IngresarCausa.aspx se agregaron dos *PlaceHolder* por cada característica opcional y para determinar si estos controles de usuario se deben incluir o no, se creó un archivo XML el cual contiene información de los paquetes que se deben incluir y los controles de usuario que interactúan con el paquete ingresar causa.

3.6.2.6.2 Variabilidad en la capa de dominio

Para extender la trazabilidad desde las características hasta el nivel de implementación, se utiliza el concepto de clases parciales. Aunque el nombre de la clase sea lo mismo, al estar en distintos paquetes las hace entidades diferentes. Si los paquetes a los que pertenecen son seleccionados, en el momento de la compilación del sistema se combinan en una única clase, haciendo que este modelo reproduzca en la implementación la línea de productos software la misma estrategia utilizada en los casos de uso y diseño. Por lo tanto, para derivar en un producto concreto basta con indicar al compilador los paquetes necesarios que corresponden a la configuración elegida en el modelo de característica. De esta manera se cubre el objetivo de la trazabilidad uno a uno desde las características hasta el código (30).

En la Figura 3-20 se visualiza la implementación de la capa de dominio para esta primera iteración. El paquete principal ingresar causa se divide en sub-paquetes que representan la implementación de la arquitectura definida para la línea de productos. Cada sub-paquete representa una capa lógica para un propósito definido. En el paquete ingresar causa se almacena la capa de presentación que se implementa con la página

LEX_ING_IngresarCausa.aspx como se mencionó en el punto anterior, la cual se comunica con la clase LEX_DEL_Ingreso.cs del paquete Delegate que permite desacoplar la capa de presentación con la de dominio. Esta decisión se tomó para futuro evolucionar la arquitectura de la línea de productos a una orientada a servicios que es un requisito no funcional que generalmente solicitan los clientes debido a la interacción de estos sistemas con otros de la organización.

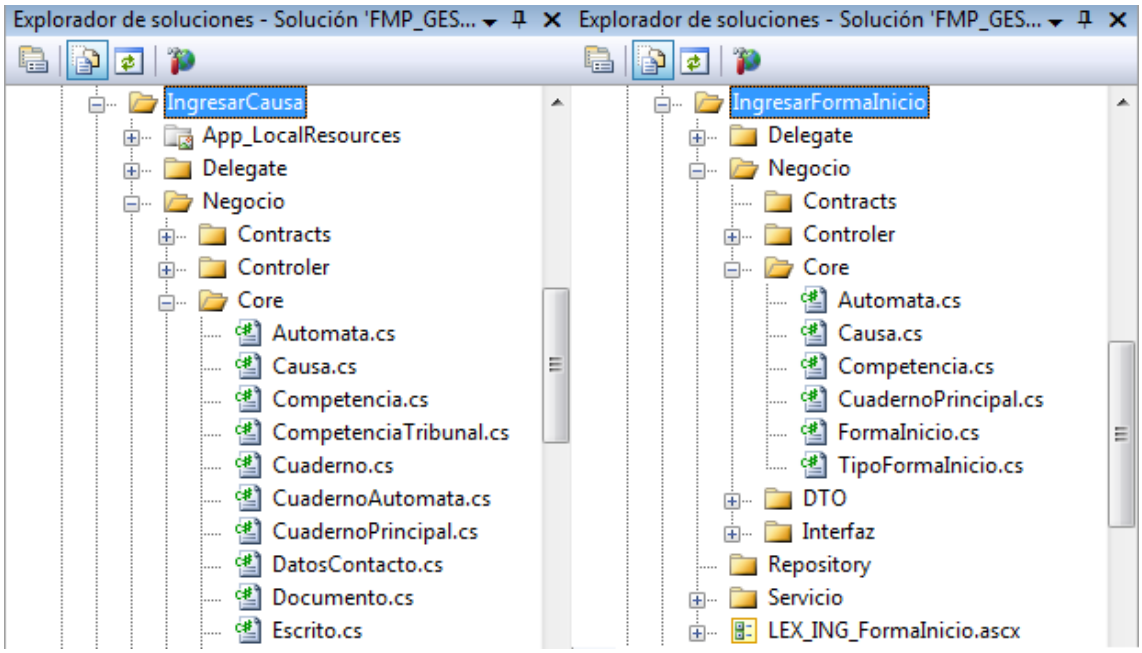


Figura 3-20: Implementación de la capa de dominio, primera iteración.

La capa de dominio se implementa en el paquete Negocio que contiene a su vez sub-paquetes, las entidades del dominio (sub-paquete Core) que almacena todas las clases parciales del diagrama de clases que se diseñaron para esta iteración, los contratos con los repositorios (sub-paquete Contracts), los servicios del dominio (sub-paquete Controller) y las interfaces de los servicios del dominio (sub-paquete Interfaz), además para evitar enviar las entidades del negocio directamente a las capas superiores se creó un sub-paquete DTO, que contiene una copia de las entidades del negocio pero sin su lógica, es decir, sólo con sus atributos. Por último, la capa del dominio se comunica solamente con la capa de servicios (paquete Servicio) que almacena la clase SRV_Ingreso que contiene los servicios que proveen el dominio. Esta clase es la que interactúa directamente con la clase LEX_DEL_Ingreso del paquete Delegate.

Una de las desventajas, al implementar las clases parciales en la capa de dominio, es que todas las clases parciales deben pertenecer al mismo *namespace* (31) para poder ser compiladas y agrupadas en una sola clase, es decir, al aplicar clases parciales necesariamente todas las clases deben estar situadas en un mismo *namespace*, por lo que no es posible emplear este mecanismo como una manera de encapsular las características. Como alternativa, se utilizó un *namespace* por cada capa definida en la arquitectura, tal como se visualiza en la Figura 3-21.

```
namespace FMP_GESLEX.ExpedienteDigital.Negocio.Core
{
    [ActiveRecord]
    public partial class Causa : ActiveRecordBase<Causa>
    {
```

```
namespace FMP_GESLEX.ExpedienteDigital.Negocio.Core
{
    public partial class Causa
    {
```

Figura 3-21: *Namespace* de capa de negocio de la característica ingresar causa.

3.6.2.6.3 Variabilidad en la capa de infraestructura de acceso a datos

Un punto importante es solucionar la variabilidad relacionada a la capa de infraestructura de acceso a datos. La solución adoptada para implementar la arquitectura previamente definida con NHinernate (32), es utilizar el *framework* Castle ActiveRecord (33) que es una capa implementada por encima de NHibernate, la ventaja es que NHibernate hace uso de archivos XML para mapear objetos a datos relacionales y ActiveRecord lo hace a través de programación declarativa que auto-genera los archivos de mapeo XML automáticamente.

Por ejemplo, si se incluyen la característica opcional ingresar forma inicio, se carga el paquete ingresar forma inicio que contiene la clase Causa (Figura 3-22) con el atributo ioTipoForIniCausa, entonces Castle ActiveRecord mapeará el campo. En caso contrario, si no se carga el paquete, el campo no se mapea logrando evitar atributos que no son utilizados nunca por la aplicación (Figura 3-23).

```
public partial class Causa
{
    [BelongsTo]
    public virtual TipoFormaInicio ioTipoForIniCausa { get; set; }

}
```

Figura 3-22: Clase parcial causa del paquete Ingresar Forma Inicio.

```
[ActiveRecord]
public partial class Causa : ActiveRecordBase<Causa>
{
    [PrimaryKey(PrimaryKeyType.Native)]
    public virtual int causaId { get; set; }
    [Property]
    public virtual DateTime idFecIngSistema { get; set; }
    [Property]
    public virtual DateTime? idFecTermino { get; set; }
    [Property]
    public virtual int iiEraCausa { get; set; }
```

Figura 3-23: Clase parcial del paquete Ingresar Causa.

Sin embargo, para la creación de la base de datos se realizó incorporando todas las tablas del modelo de datos, por lo tanto podrían existir columnas sin uso, si no se

seleccionan algunas características opcionales. Claramente éste es un tema pendiente que se debería abordar en trabajos futuros de la evolución de la línea de productos.

3.6.3 Iteración 2: Ingresar escrito

Para esta segunda iteración se diseñó y construyó la característica ingresar escrito. Esta característica tiene como principal funcionalidad permitir a los abogados que son partes de una causa, presentar un documento a la organización que en términos generales es cualquier escrito que da cuenta de un hecho, de tal manera que el juez se pronuncie sobre ese hecho.

3.6.3.1 Combinación de paquetes en UML y representación de la variabilidad

Como se detalló en la primera iteración, la variabilidad se expresa utilizando el concepto de combinación de paquetes (*package merge*) de UML, según la propuesta de Miguel Laguna y Bruno González (2).

La aplicación de este esquema respecto a la segunda iteración, se visualiza en la Figura 3-24 que existen características agrupadas en alternativas (escrito funcionario o escrito abogado), donde sólo se debe elegir una de ellas, lo cual implica la existencia del correspondiente paquete en el modelo de casos de uso y diseño. Por otro lado, mediante el uso de clases parciales organizadas en paquetes, se puede establecer una correspondencia directa entre los modelos de casos de uso y diseño hasta la implementación de la línea de productos.

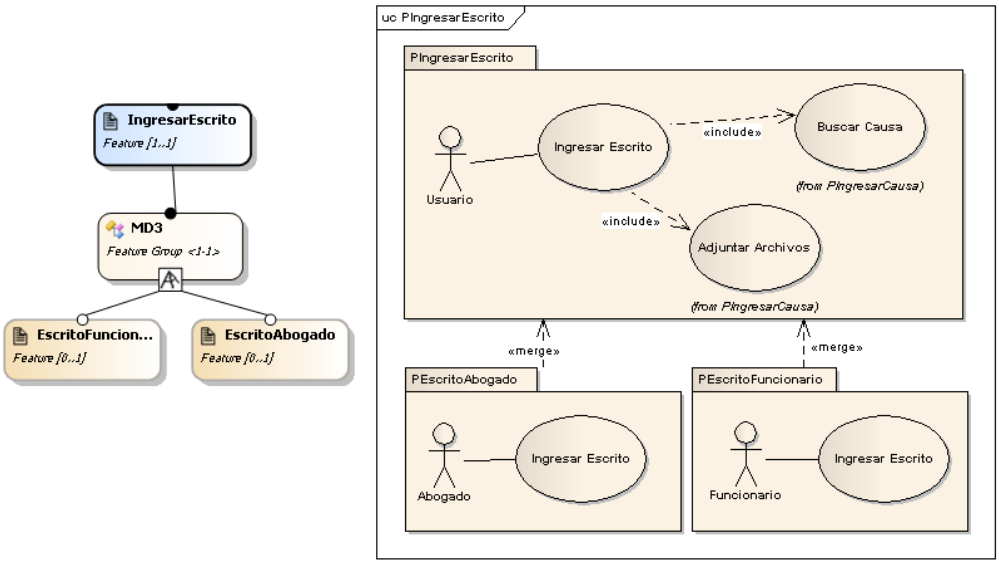


Figura 3-24: Modelo de característica de la iteración 2 y el mecanismo de "*package merge*" aplicado al modelo de casos de uso.

3.6.3.2 Análisis y diseño de la característica ingreso de escrito

En la característica obligatoria Ingresar Escrito, es posible seleccionar quien es el actor que ingresará al sistema el escrito, ya sea el funcionario o el abogado. Esta variabilidad es fundamental, ya que existen datos que no son necesarios u obligatorios en caso de seleccionar una u otra.

En el modelo de característica, se detalla la variabilidad que puede adoptar la característica ingresar escrito por un producto concreto de tramitación de causas judiciales, indicando claramente las que son obligatorias y agrupadas en alternativas. Las obligatorias son: ingresar escrito, mientras que las agrupadas en alternativas (o grupos XOR) son: escrito funcionario o escrito abogado, se debe elegir una de las dos. Si está presente la característica alternativa escrito funcionario, los escritos podrán ser ingresados sólo por los funcionarios de la organización y además se deberá seleccionar a la parte que presenta el escrito, en caso contrario si está presente la característica alternativa escrito abogado, los escritos podrán ser ingresados por los abogados de las partes que representan a través de internet, sin la necesidad de ir presencialmente a presentar el escrito en papel a la organización.

En la Figura 3-25 se representa la jerarquía de la característica ingresar escrito.

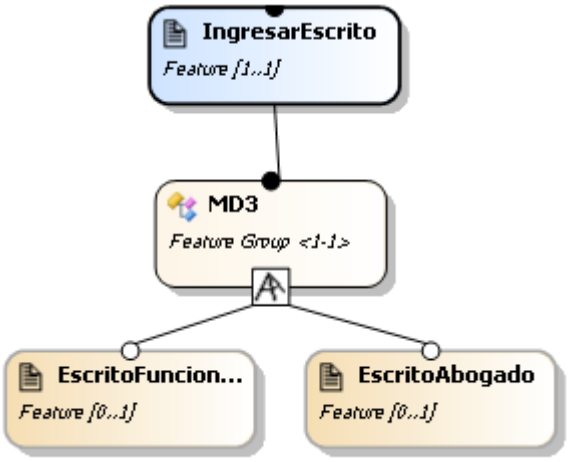


Figura 3-25: Esquema de la característica ingresar escrito.

El modelo básico de la característica ingresar escrito de la línea de productos se visualiza en la Figura 3-26. Los paquetes de la parte inferior son opcionales e independientes entre sí. Para esta segunda iteración se desarrollaron los siguientes paquetes:

- **PIngresarEscrito**: Contiene la funcionalidad de ingresar un escrito a una causa.
- **PEscritoFuncionario**: Contiene la funcionalidad para que el funcionario realice el ingreso de escrito.
- **PEscritoAbogado**: Contiene la funcionalidad para que el abogado que representa a una parte realice el ingreso de escrito a través de internet.

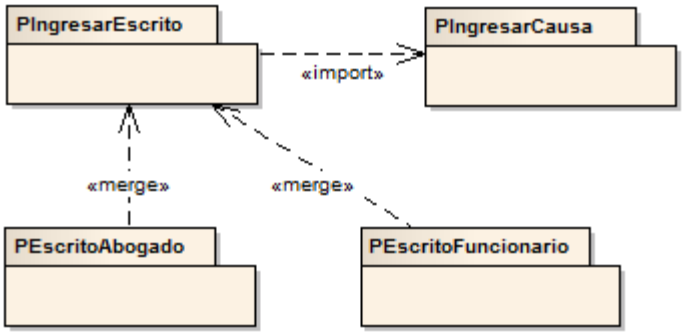


Figura 3-26: Esquema de jerarquía del paquete ingresar escrito.

En la Figura 3-27 se visualiza el modelo de casos de uso de la característica ingresar escrito. El paquete ingresar escrito importa el paquete ingresar causa, debido a que incluye los casos de uso adjuntar archivos y buscar causa. Además, si es seleccionado el paquete escrito abogado o el escrito funcionario, se incluye el actor que interactúa con el caso de uso ingresar escrito.

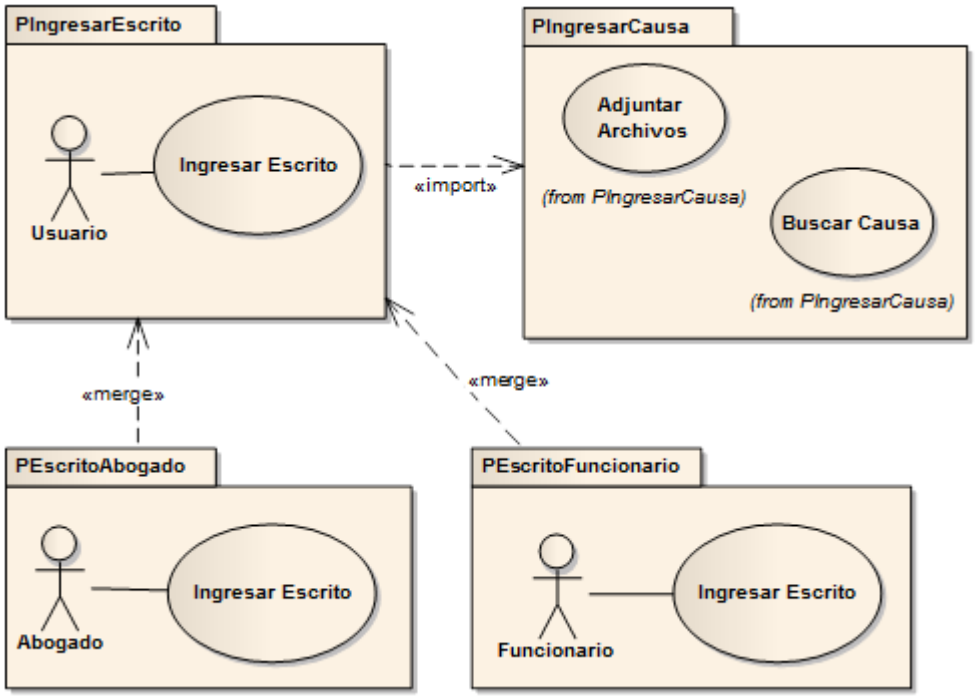


Figura 3-27: Paquetes de la característica ingreso de escrito (vista casos de uso).

A continuación se desarrollan los paquetes identificados de la característica manteniendo el esquema del desarrollo dirigido por casos de uso.

3.6.3.3 Paquete ingresar escrito

El paquete ingresar escrito se encarga de la funcionalidad de agregar escritos que son presentados por las partes, generalmente por un abogado que representa a una o varias partes del proceso judicial. Esta característica es obligatoria y se mezcla con el paquete base, tiene una dependencia con el paquete ingresar causa, debido a que incluye el caso de uso buscar causa y con el paquete adjuntar archivos.

3.6.3.4 Paquete escrito abogado

El paquete escrito abogado se encarga de la funcionalidad que el abogado realice directamente el ingreso de escrito, sin necesidad de acudir presencialmente a la organización judicial. Esta característica es alternativa y se mezcla con el paquete ingresar escrito.

3.6.3.4.1 Caso de uso: ingresar escrito (abogado)

Flujo de Eventos	
Principal	1. El abogado selecciona la opción ingresar escrito en el sistema. 2. El abogado busca y selecciona la causa a la que desea ingresar el o los escritos. Incluye caso de uso "Buscar Causa" del paquete Ingresar Causa. 3. El sistema despliega un panel con datos de la causa. Estos datos son: rol, fecha de solicitud, caratulado, etapa, estado procesal. 4. El sistema despliega los tipos de escritos disponibles según el procedimiento de la causa y la fecha de escrito que corresponde a la actual. 5. El abogado selecciona el tipo de escrito. 6. El abogado ingresa un nombre de referencia del escrito. 7. El abogado adjunta el escrito, incluye caso de uso “Adjuntar Archivo” del paquete Ingresar Causa. 8. El abogado solicita grabar el escrito. 9. El sistema almacena los datos y asocia el escrito y los documentos a la causa. 10. El sistema indica el ingreso exitoso del escrito y el(los) documentos al sistema.
Excepción Falta adjuntar archivo	11.1 No se han adjuntado archivos al escrito a grabar. El sistema indica que se debe adjuntar al menos un archivo al escrito. 11.2 Retorna a 7.
Excepción Faltan datos	11.1 No se han ingresados todos los campos obligatorios para ingresar el escrito. El sistema indica los campos que falta completar Tipo de escrito y Nombre de referencia 11.2 Retorna a 5.
Precondición	1. El abogado debe haber ingresado al sistema con su nombre de usuario y contraseña.

3.6.3.5 Paquete escrito funcionario

El paquete escrito funcionario se encarga de la funcionalidad que el funcionario de la organización realice directamente el ingreso de escrito, cuando el abogado se presenta físicamente con el escrito en la organización judicial. Esta característica es alternativa y se mezcla con el paquete ingresar escrito.

3.6.3.5.1 Caso de uso: ingresar escrito (funcionario)

Flujo de Eventos	
Principal	1. El funcionario selecciona la opción ingresar escrito en el sistema. 2. El funcionario busca y selecciona la causa a la que desea ingresar el o los escritos. Incluye caso de uso "Buscar Causa" del paquete Ingresar Causa.

	3. El sistema despliega un panel con datos de la causa. Estos datos son: rol, fecha de solicitud, caratulado, etapa, estado procesal. 4. El sistema despliega los tipos de escritos disponibles según el procedimiento de la causa y la fecha de escrito que corresponde a la actual. 5. El funcionario selecciona el tipo de escrito. 6. El funcionario ingresa un nombre de referencia del escrito. 7. El funcionario selecciona la parte que presenta el escrito. 8. El funcionario adjunta el escrito, incluye caso de uso “Adjuntar Archivo” del paquete Ingresar Causa. 9. El funcionario solicita grabar el escrito. 10. El sistema almacena los datos y asocia el escrito y los documentos a la causa. 11. El sistema indica el ingreso exitoso del escrito y el(los) documentos al sistema.
Excepción Falta adjuntar archivo	12.1 No se han adjuntado archivos al escrito a grabar. El sistema indica que se debe adjuntar al menos un archivo al escrito. 12.2 Retorna a 7.
Excepción Faltan datos	12.1 No se han ingresados todos los campos obligatorios para ingresar el escrito. El sistema indica los campos que falta completar Tipo de escrito y Nombre de referencia 12.2 Retorna a 5.
Precondición	1. El funcionario debe haber ingresado al sistema con su nombre de usuario y contraseña.

El diagrama de secuencia del sistema de la Figura 3-28 muestra el curso de eventos específicos del caso de uso ingresar escrito (funcionario). Se muestra el escenario principal de éxito del caso de uso. Se indica que el usuario genera el evento del sistema *obtenerCausa, listarTipoSolicitud, crearSolicitud, listarPartes* y referencia los diagramas de secuencia de los casos de uso buscar causa y adjuntar archivos del paquete ingresar causa. El evento *listarPartes* es el que representa la variabilidad que puede adoptar esta característica.

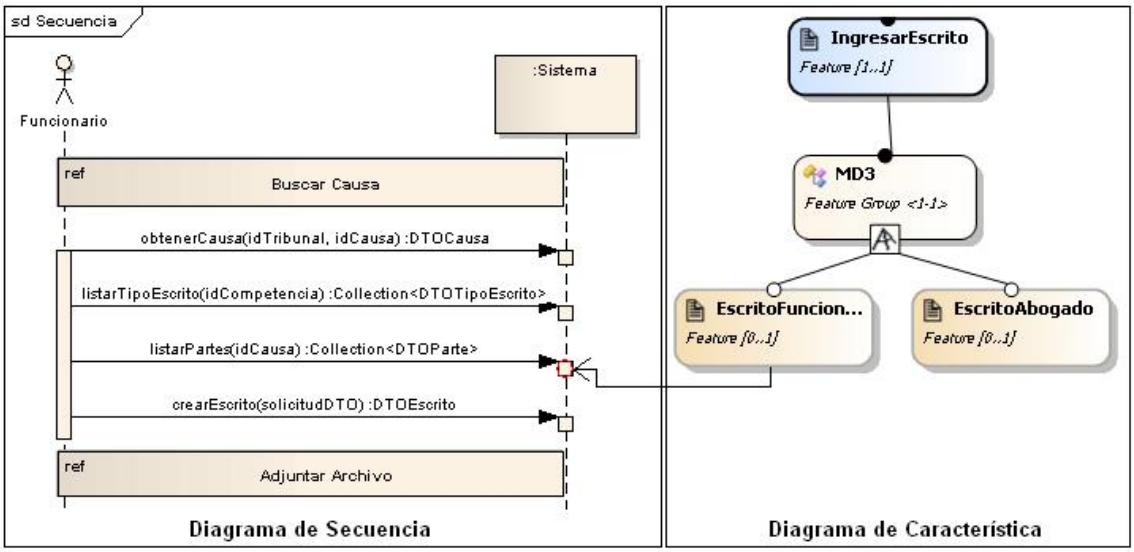


Figura 3-28: Diagrama de secuencia del caso de uso ingresar escrito (funcionario).

3.6.3.5.2 Diagrama de clases

Una vez terminados todos los diagramas de comunicación para la realización de los casos de uso del paquete ingresar escrito, escrito abogado y escrito funcionario, se identificaron las especificaciones de las clases e interfaces que participan cuando la característica ingresar escrito es seleccionada.

En la Figura 3-29 se visualiza el diagrama de clases de los paquetes de la característica ingresar escrito que permite capturar todas las clases parciales, visualizando los métodos posibles que pueden tener cada clase cuando es seleccionada esta característica. Tal como se realizó en la primera iteración, al aplicar el mecanismo *package merge* (2), el resultado de combinar estos paquetes es que aparecen por ejemplo una clase final Causa con los atributos y métodos de cada paquete.

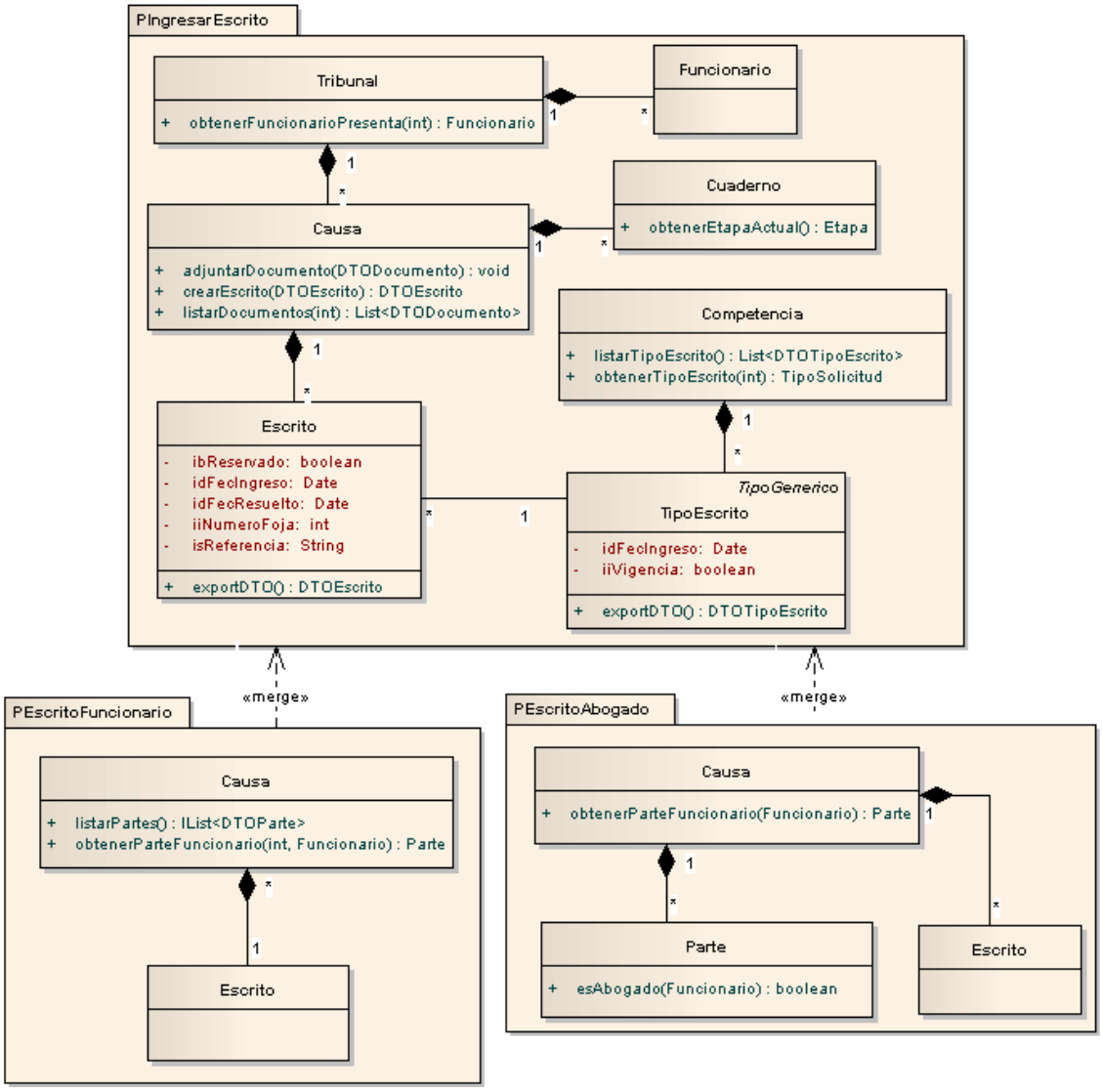


Figura 3-29: Mecanismo de "*package merge*" aplicado al diagrama de clases de la iteración 2.

3.6.3.6 Implementación

Al igual que en la primera iteración, la implementación de las características de la línea de productos se realiza utilizando Microsoft Visual Studio 2008 y el *plug-in* FMT (5).

3.6.3.6.1 Variabilidad en la capa de presentación

La característica principal ingresar escrito contiene una página llamada LEX_ING_IngresoDeEscrito.aspx donde se ingresan todos los datos del escrito. Sin embargo, para representar la variabilidad del grupo de alternativas que tiene esta característica, se creó un control de usuario llamado LEX_ING_EscritoFuncionario.ascx. Como se visualiza en el recuadro rojo de la Figura 3-30, el control de usuario está incluido en la página principal si la característica alternativa escrito funcionario es seleccionada al derivar en un producto concreto.

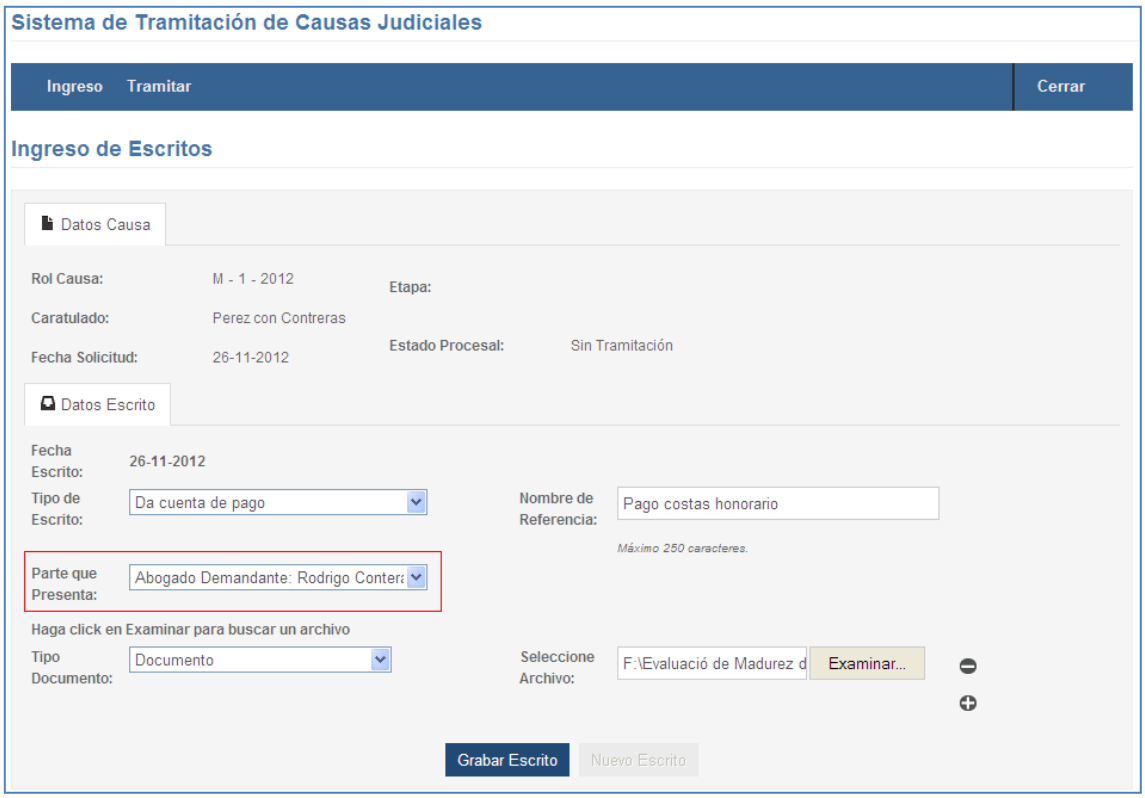


Figura 3-30: Página LEX_ING_IngresoDeEscrito.aspx seleccionado la característica alternativa escrito funcionario.

Si la característica alternativa escrito abogado es seleccionada, entonces no se incluye el control de usuario LEX_ING_EscritoFuncionario.ascx, ya que no existe variabilidad en la interfaz de usuario porque el usuario que presenta el escrito es el abogado conectado al sistema, por ende se sabe a priori quien es la parte que presenta el escrito, tal como se visualiza en la Figura 3-31.

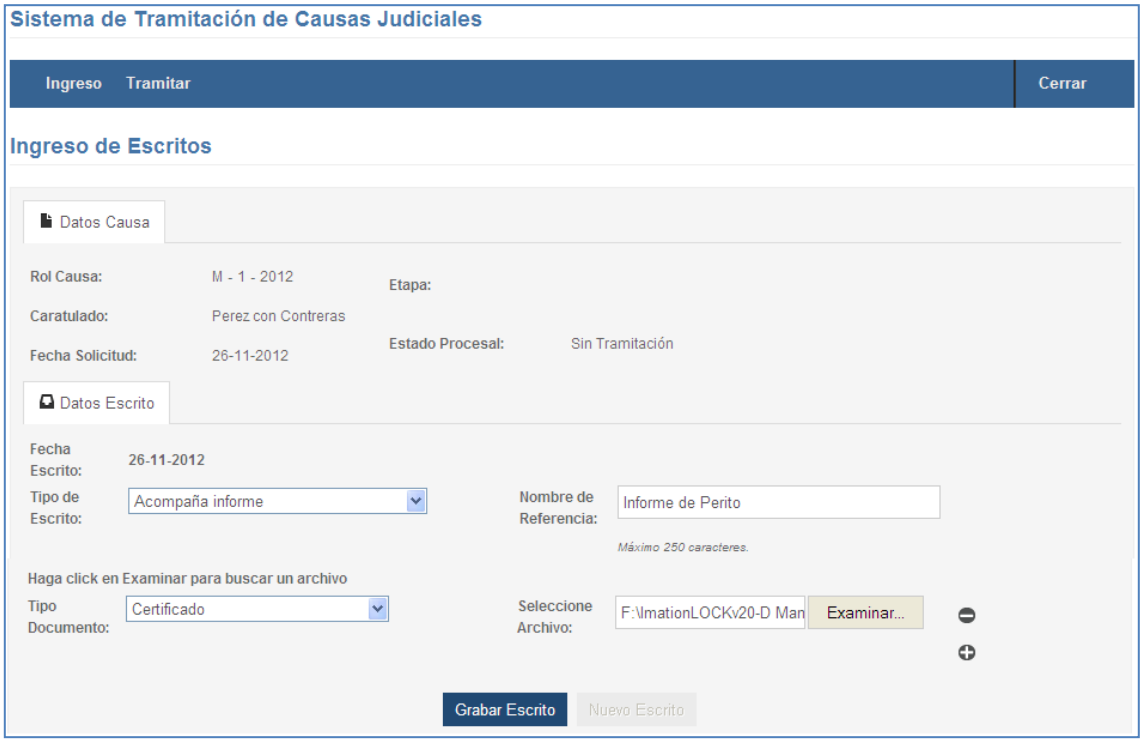


Figura 3-31: Página LEX_ING_IngresoDeEscrito.aspx seleccionado la característica alternativa escrito abogado.

Para añadir en tiempo de ejecución el control de usuario LEX_ING_EscritoFuncionario.ascx, tal como la primera iteración, se utiliza un control de ASP.NET llamado *PlaceHolder* que actúan como contenedores de los controles de usuario. En la página LEX_ING_IngresoDeEscrito.aspx se agregó un *PlaceHolder* para incluir la característica alternativa escrito funcionario y para determinar si este control de usuario se debe incluir o no, se creó un archivo XML el cual contiene información del paquete que se debe incluir y el control de usuario que interactúan con el paquete ingresar escrito. El archivo XML contiene la información si el paquete alternativo debe o no estar incluido al momento de cargar la página principal.

3.6.3.6.2 Variabilidad en la capa de dominio

Para extender la trazabilidad desde las características hasta el nivel de implementación, se utiliza el concepto de clases parciales (30).

En la Figura 3-32 se visualiza la implementación de la capa de dominio para esta segunda iteración. En el paquete ingresar escrito se almacena la capa de presentación que se implementa con la página LEX_ING_IngresoDeEscrito.aspx como se mencionó en el punto anterior, la cual se comunica con la clase LEX_DEL_Ingreso.cs del paquete Delegate que permite desacoplar la capa de presentación con la de dominio.

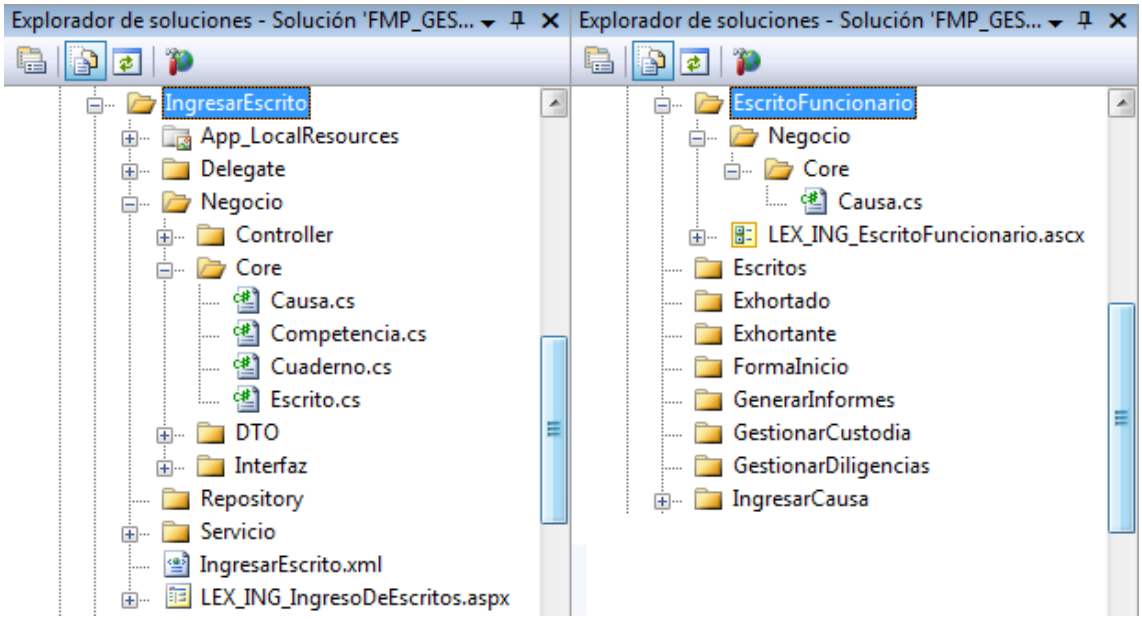


Figura 3-32: Implementación de la capa de dominio, segunda iteración.

La capa de dominio se implementa en el paquete Negocio que contiene todas las clases parciales del diagrama de clases que se diseñaron para esta iteración.

3.6.3.6.3 Variabilidad en la capa de infraestructura de acceso a datos

La variabilidad relacionada a la capa de infraestructura de acceso a datos se implementó con el *framework* Castle ActiveRecord (33) que es una capa implementada por encima de NHibernate. Sin embargo, en esta iteración no existe variabilidad en los atributos de los objetos, ya que todos los atributos se agregan en la característica principal Ingresar Escrito. Por lo tanto, siempre se ocuparan todos los campos de las tablas, a diferencia de la primera iteración, donde existe la posibilidad de que algunos campos nunca sean utilizados.

3.6.4 Iteración 3: Tramitar expediente

Para esta tercera iteración se diseñó y construyó la característica tramitar expediente. Esta característica tiene como funcionalidad principal permitir dar curso y tramitar una causa, registrando los antecedentes más relevantes del proceso. Incorpora la configuración de flujo de trabajo parametrizable que guía la tramitación de los diferentes procedimientos judiciales de cada competencia. Este flujo de trabajo configura la tramitación de los procedimientos y eventos asociados que existen en la organización, además de las etapas, plazos y los tipos de trámites dentro de las etapas. En las etapas se configuran los hitos, que es un nemotécnico, definido para representar una resolución o actuación dentro del procedimiento judicial.

3.6.4.1 Análisis y diseño de la característica tramitar expediente

Como se detalló en la primera iteración, la variabilidad se expresa utilizando el concepto de combinación de paquetes (*package merge*) de UML, según la propuesta de Miguel Laguna y Bruno González (2).

La aplicación de este esquema respecto a la tercera iteración, se visualiza en la Figura 3-33.

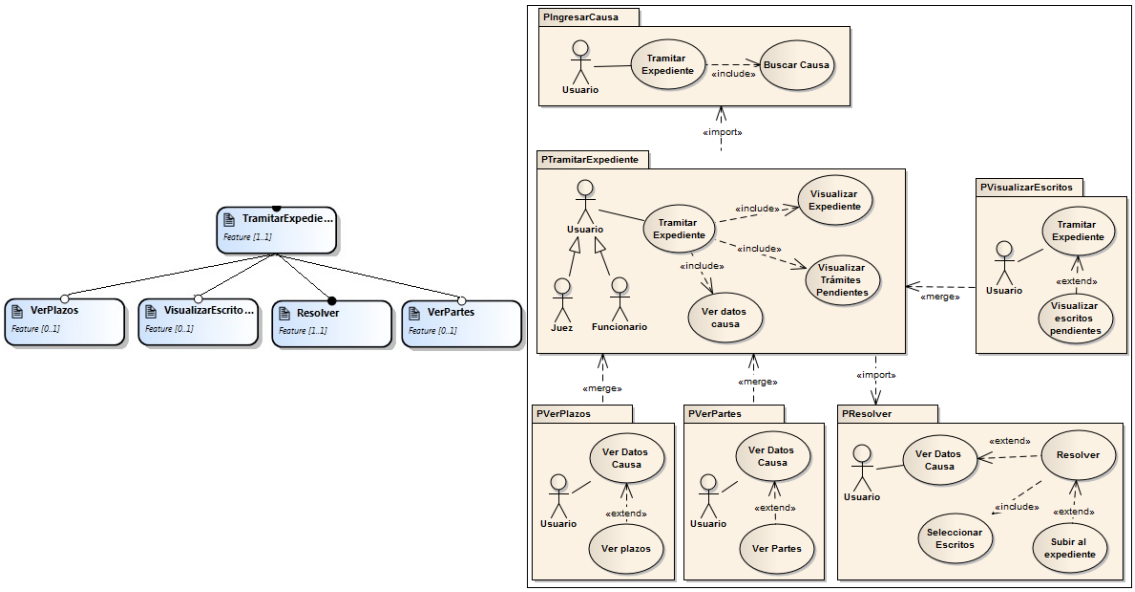


Figura 3-33: Modelo de Característica de la Iteración 3 y mecanismo de "*package merge*" aplicado al modelo de casos de uso.

La característica obligatoria tramitar expediente, es el corazón de los sistemas de tramitación de causas judiciales, ya que permite almacenar y registrar la información y elementos de apoyo al seguimiento de la tramitación de los procedimientos judiciales, además guía la tramitación de la causa, mostrando sólo aquellos hitos o nomenclaturas que puedan ser dictados en la etapa y estado procesal que se encuentra la causa. En resumen, en el expediente digital se logra visualizar en forma rápida y oportuna toda la información relevante de la causa para la dictación de una resolución o actuación por los jueces o funcionarios de la organización. Además del punto de vista de las partes, actúa como la manera de conocer el estado del proceso judicial, logrando una mayor transparencia.

En el modelo de característica, se detalló la variabilidad que puede adoptar la característica para un producto concreto de tramitación de causas judiciales, indicando claramente las que son obligatorias y opcionales. La obligatoria es la característica resolver y las opcionales son ver plazos, ver partes y visualizar escritos. Si las características ver plazos y ver partes está presente, en el expediente digital se visualizan las partes que pertenecen a la causa y los plazos que sean derivados de resoluciones o actuaciones que se han dictado durante el proceso judicial. En la Figura 3-34 se representa la jerarquía de la característica tramitar expediente.

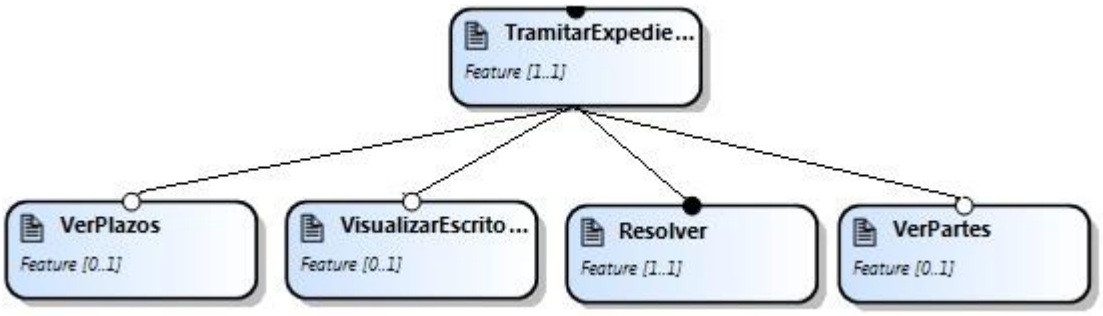


Figura 3-34: Esquema de la característica Tramitar Expediente.

El modelo básico de la característica tramitar expediente de la línea de productos se visualiza en la Figura 3-35. Adoptando la propuesta Miguel Laguna y Bruno González (2), los paquetes de la parte inferior son opcionales e independientes entre sí. Para esta tercera iteración se desarrollaron los siguientes paquetes:

- **PTramitarExpediente**: Contiene la funcionalidad de visualizar la información más relevante de la causa, de tal manera, de que las partes puedan saber en forma rápida y oportuna el estado de la causa.
- **PVerPlazo**: Contiene la funcionalidad de visualizar los plazos que tiene la causa.
- **PVerPartes**: Contiene la funcionalidad de visualizar las partes de la causa.
- **PResolver:** Contiene la funcionalidad de dar curso y tramitar una causa, registrando los antecedentes más relevantes del proceso.
- **PVisualizarEscritos**: Contiene la funcionalidad de visualizar los escritos que no están resueltos por la organización y que son presentados por las partes.

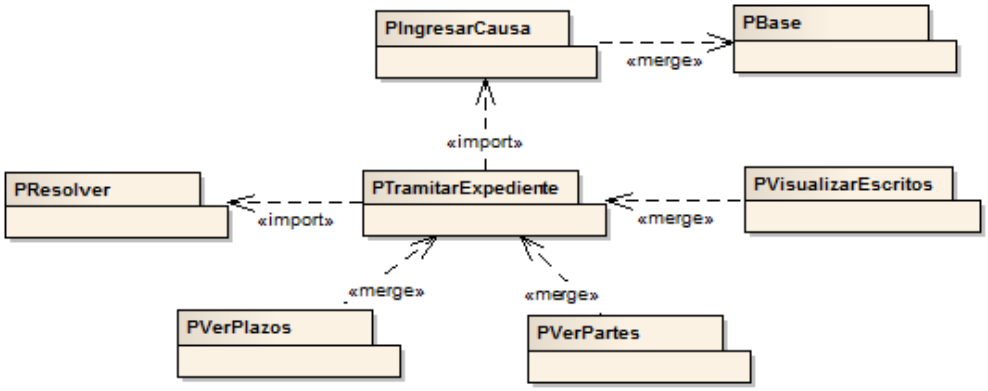


Figura 3-35: Esquema de jerarquía del paquete tramitar expediente.

En la Figura 3-36 se visualiza el modelo de casos de uso de la característica tramitar expediente. El paquete tramitar expediente importa el paquete ingresar causa, debido a que incluye el caso de uso buscar causa. Si se selecciona el paquete visualizar escritos, se incluye en el modelo un caso de uso adicional como extensión que permite visualizar los escritos que son presentados por las partes de la causa. Si se selecciona el paquete ver plazos, se incluye en el modelo de caso de uso adicional como extensión que permite visualizar los plazos generados por la causa o resoluciones dictadas por el juez. Y por último, si se selecciona el paquete ver partes, se incluye en el modelo un

caso de uso adicional como extensión que permite visualizar en el expediente digital las partes con toda su información que participan en el proceso judicial.

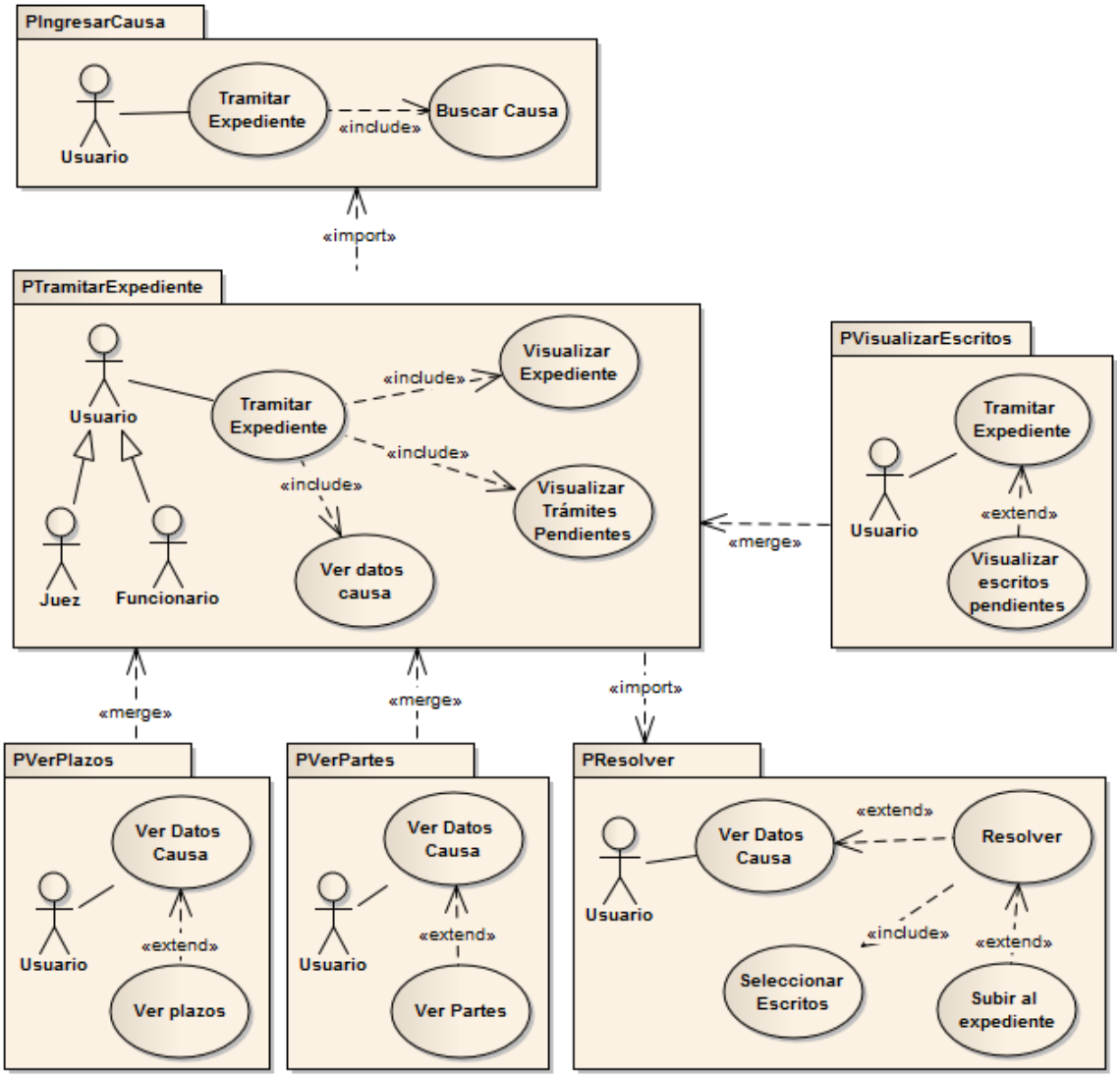


Figura 3-36: Paquetes de la característica tramitar expediente (vista casos de uso).

A continuación se desarrollan los paquetes identificados de la característica manteniendo el esquema del desarrollo dirigido por casos de uso.

3.6.4.2 Paquete tramitar expediente

El paquete tramitar expediente se encarga de la funcionalidad de obtener toda la información más relevante de la causa, tales como, etapa en la que se encuentra, estado procesal, estado administrativo, fecha de ingreso y el tipo de procedimiento judicial que adopto. Esta característica es obligatoria e importar el paquete ingresar causa, debido a que incluye el caso de uso buscar causa y adjuntar archivos.

3.6.4.2.1 Caso de uso: tramitar expediente

Flujo de Eventos Principal	1. El sistema busca la causa. Incluye caso de uso "Buscar Causa" del

	paquete Ingresar Causa. 2. El sistema muestra los datos de la causa. Incluye caso de uso "Ver datos causa". 3. El sistema muestra las partes de la causa. Incluye caso de uso "Ver Partes" del paquete Ver Partes. 4. El sistema muestra los plazos de la causa. Incluye caso de uso "Ver plazos" del paquete Ver Plazos. 5. El sistema consulta los cuadernos disponibles en la causa. 6. El sistema despliega los cuadernos de la causa. 7. El usuario selecciona un cuaderno de la causa. 8. El sistema muestra el expediente digital de la causa para el cuaderno seleccionado. Incluye caso de uso "Visualizar expediente". 9. El sistema muestra los trámites pendientes creados para el cuaderno seleccionado. Incluye caso de uso "Visualizar Trámites Pendientes". 10. El sistema muestra los escritos pendientes para la causa. Incluye caso de uso "Visualizar escritos pendientes" del paquete Visualizar Escritos.
Alternativo Tramitar Causa	11. El usuario solicita generar una nueva resolución o actuación en la causa para el cuaderno seleccionado. Se ejecuta caso de uso "Resolver" del paquete Resolver.
Precondición	1. La causa a desplegar debe existir en el sistema.

El diagrama de secuencia del sistema de la Figura 3-37 muestra el curso de eventos específicos del caso de uso tramitar expediente. Se indica que el funcionario o el juez generan los eventos del sistema *obtenerTiposTramite, listarCuadernosCausa* y referencia los diagramas de secuencia de los casos de uso ver datos causa, visualizar expediente y visualizar trámites pendientes. Los eventos de los casos de uso de ver plazo, ver partes y visualizar escritos son alternativos y se incluye solo en caso de seleccionar las características correspondientes.

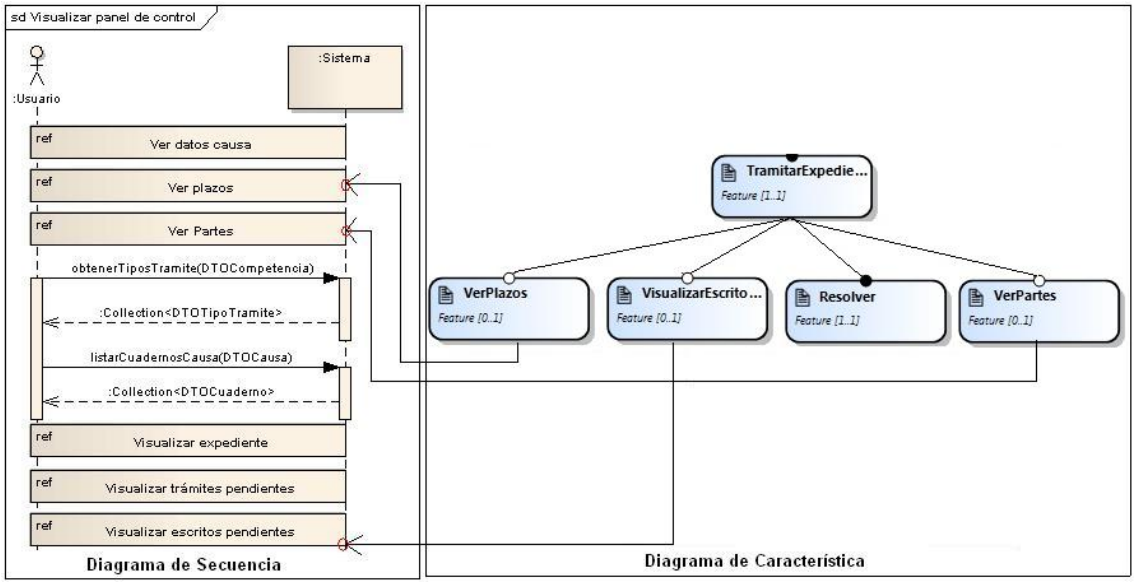


Figura 3-37: Diagrama de secuencia del caso de uso tramitar expediente.

3.6.4.2.2 Caso de uso: ver datos causa

Este caso de uso permite al usuario ver los datos de la causa tales como el rol, caratulado, etapa, fecha de ingreso, entre otros.

Flujo de Eventos	
Principal	1. El usuario solicita ver los datos de la causa. 2. El sistema obtiene los datos de la causa 3. El sistema despliega los datos de la causa. Estos son: - Rol - Caratulado - Fecha de solicitud - Fecha de inicio causa - Juez - Etapa - Estado procesal - Ver documentos - Fecha de término - Tipo de término.
Precondición	1. La causa a desplegar debe existir en el sistema.

3.6.4.2.3 Caso de uso: visualizar expediente

Este caso de uso permite al usuario visualizar el expediente digital de la causa.

Flujo de Eventos	
Principal	1. El usuario solicita ver el expediente digital de la causa para el cuaderno seleccionado. 2. El sistema consulta el expediente digital para el cuaderno seleccionado. 3. El sistema despliega el resultado de la consulta. Estos datos son: - Ver escritos (permite expandir o contraer la grilla con el(los) escrito(s) resuelto(s) con la resolución o actuación). - Tipo Trámite - Ver Documento - Fecha Trámite - Referencia del Trámite - Etapa - Estado Procesal - Juez o Funcionario que bloquea o firma el trámite
Precondición	Causa debe existir en el sistema.
Alternativo Ver Documentos	4. El usuario solicita ver el documento del trámite. 5. El sistema muestra una grilla con el detalle del documento. Los datos son: - Archivo (link al archivo) - Nombre Archivo
Invariante Trámites reservados	Los trámites reservados no son visibles para las partes, pero si son visibles para los jueces y funcionarios.

3.6.4.2.4 Caso de uso: visualizar trámites pendientes

Este caso de uso permite al usuario consultar sus trámites pendientes. Son trámites que aún no son subidos al expediente digital.

Flujo de Eventos	
Principal	1. El usuario solicita ver los trámites pendientes creados para el cuaderno seleccionado. 2. El sistema consulta los trámites pendientes para el cuaderno seleccionado. 3. El sistema despliega el resultado de la consulta. Estos datos son: - Tipo Trámite - Fecha Trámite - Referencia Trámite - Etapa - Estado Procesal - Juez o Funcionario - Ver Trámite Pendiente
Precondición	Causa debe existir en el sistema.
Alternativo Ver trámite pendiente	4. El usuario solicita ver el trámite pendiente para seguir trabajando en el trámite. 5. El sistema carga los datos del trámite para continuar trabajando. Incluye el caso de uso "Resolver" del paquete Resolver.
Invariante Trámites Pendientes	El sistema debe listar sólo los trámites pendientes que pertenecen al usuario conectado.

3.6.4.3 Paquete ver partes

El paquete ver partes se encarga de la funcionalidad de obtener el listado de los partes que pertenecen a la causa. Esta característica es opcional y se mezcla con el paquete ver datos causa.

3.6.4.3.1 Caso de uso: ver partes

Flujo de Eventos	
Principal	1. El sistema despliega las partes de la causa. Los datos disponibles son: - Tipo de Persona - RUT / N° Identificación - Nombres - Apellido Paterno - Apellido Materno - Email

3.6.4.4 Paquete ver plazos

El paquete ver plazos se encarga de la funcionalidad de obtener los plazos asociados de la causa. Estos plazos son referenciales y pueden ser generados por ámbitos de la causa y trámites. Esta característica es opcional y se mezcla con el paquete ver datos causa.

3.6.4.4.1 Caso de uso: ver plazos

Flujo de Eventos	
Principal	1. El usuario solicita ver los plazos de la causa. 2. El sistema obtiene los plazos asociados a la causa. 3. El sistema despliega la lista de plazos. Estos son: - Tipo Plazo (los valores son: Plazo Causa o Plazo Trámite) - Referencia Trámite - Fecha Inicio Plazo - Fecha Vencimiento Plazo - Duración (expresado en días) - Días Suspensión (para ambos tipos de plazo) - Estado Plazo (los valores son: Vigente, Suspendido o Vencido)
Precondición	Causa debe existir en el sistema.
Invariante Plazo de trámites reservados	Los plazos generados a partir de trámites reservados sólo son visibles para la organización. Las partes sólo pueden ver plazos cuyos trámites no son reservados.

3.6.4.5 Paquete visualizar escritos

El paquete visualizar escritos se encarga de la funcionalidad de consultar por los escritos presentados por las partes y que están pendientes de resolver por un juez. Esta característica es opcional y se mezcla con el paquete tramitar expediente.

3.6.4.5.1 Caso de uso: visualizar escritos

Flujo de Eventos	
Principal	1. El usuario solicita ver los escritos pendientes de la causa. 2. El sistema consulta los escritos pendientes de la causa. 3. El sistema muestra los escritos pendientes. Los datos son: - Tipo Escrito - Ver Documentos - Referencia - Fecha de Ingreso - Parte que presenta el escrito
Precondición	Causa debe existir en el sistema.

3.6.4.6 Paquete resolver

El paquete resolver se encarga de la funcionalidad de tramitar una causa, dictando una resolución por el juez o una actuación por un funcionario de la organización. Esta característica es obligatoria y es importado por el paquete tramitar expediente.

3.6.4.6.1 Caso de uso: resolver

Flujo de Eventos	
Principal	1. El usuario solicita generar una nueva resolución o actuación. 2. El sistema consulta los escritos pendientes de resolver para la causa, incluye caso de uso Seleccionar Escritos. 3. El sistema consulta la(s) resolución(es) / actuación(es). - Resolución(es) / actuación(es) para la etapa de la causa - Resolución(es) / actuación(es) generales

	4. El sistema consulta las partes de la causa. 5. El usuario indica si se trata de una resolución reservada. 6. El usuario indica la fecha de la resolución y la referencia. La fecha es seleccionada desde un calendario. 7. El usuario adjunta el archivo de la resolución/actuación. Incluye caso de uso "Adjuntar Archivos" del paquete Ingresar Causa. 9. El usuario guarda los cambios realizados.
Precondición	Causa debe existir en el sistema.
Alternativo Subir al expediente	10. El usuario solicita subir la resolución/actuación al expediente digital. Se ejecuta el caso de uso "Subir al expediente".

3.6.4.7 Diagrama de clases

Una vez terminados todos los diagramas de colaboración para la realización de los casos de uso del paquete tramitar expediente, ver plazos, ver partes, resolver y visualizar escritos se identificaron las especificaciones de las clases e interfaces que participan cuando la característica tramitar expediente es seleccionada.

En la Figura 3-38 y Figura 3-39 se visualiza el diagrama de clases de los paquetes de la característica tramitar expediente que permite capturar todas las clases parciales, visualizando los métodos posibles que pueden tener cada clase cuando es seleccionada esta característica. Tal como se realizó en la primera iteración, al aplicar el mecanismo *package merge* (2), el resultado de combinar estos paquetes es que aparecen por ejemplo una clase final Causa los métodos *listarPartes, obtenerEscritos* y *obtenerPlazos* de cada paquete, además al importar el paquete Resolver a la clase Causa se agregan otros cinco métodos más y una dependencia indirecta a la clase HitoTramite, está asociación se realizó para favorecer los tiempos de respuestas a la hora de identificar a que causa corresponden los hitos generados por el trámite.

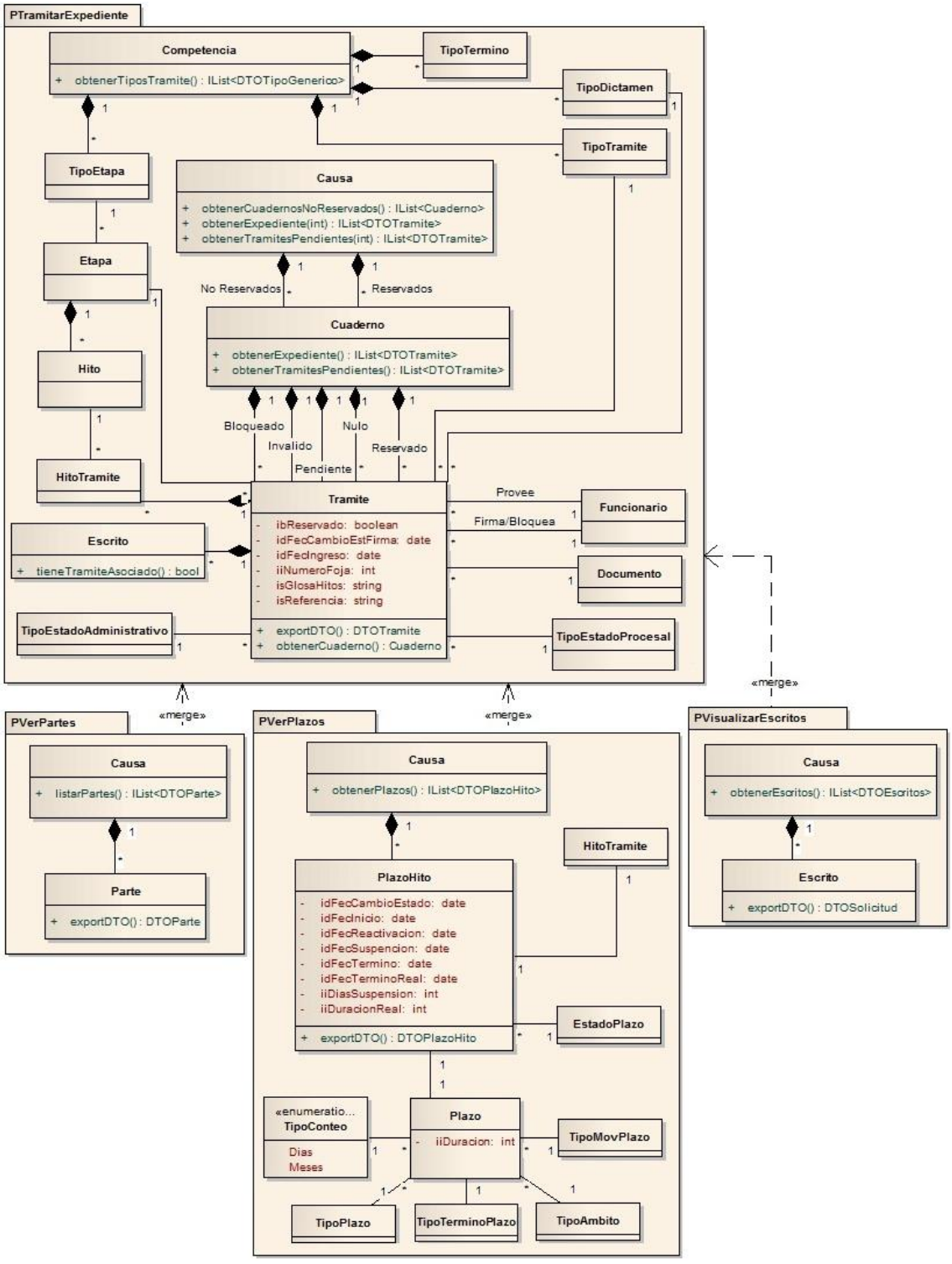


Figura 3-38: Mecanismo de "*package merge"* aplicado al diagrama de clases de la iteración 3 (paquete tramitar expediente, paquetes ver partes, ver plazos y visualizar escritos).

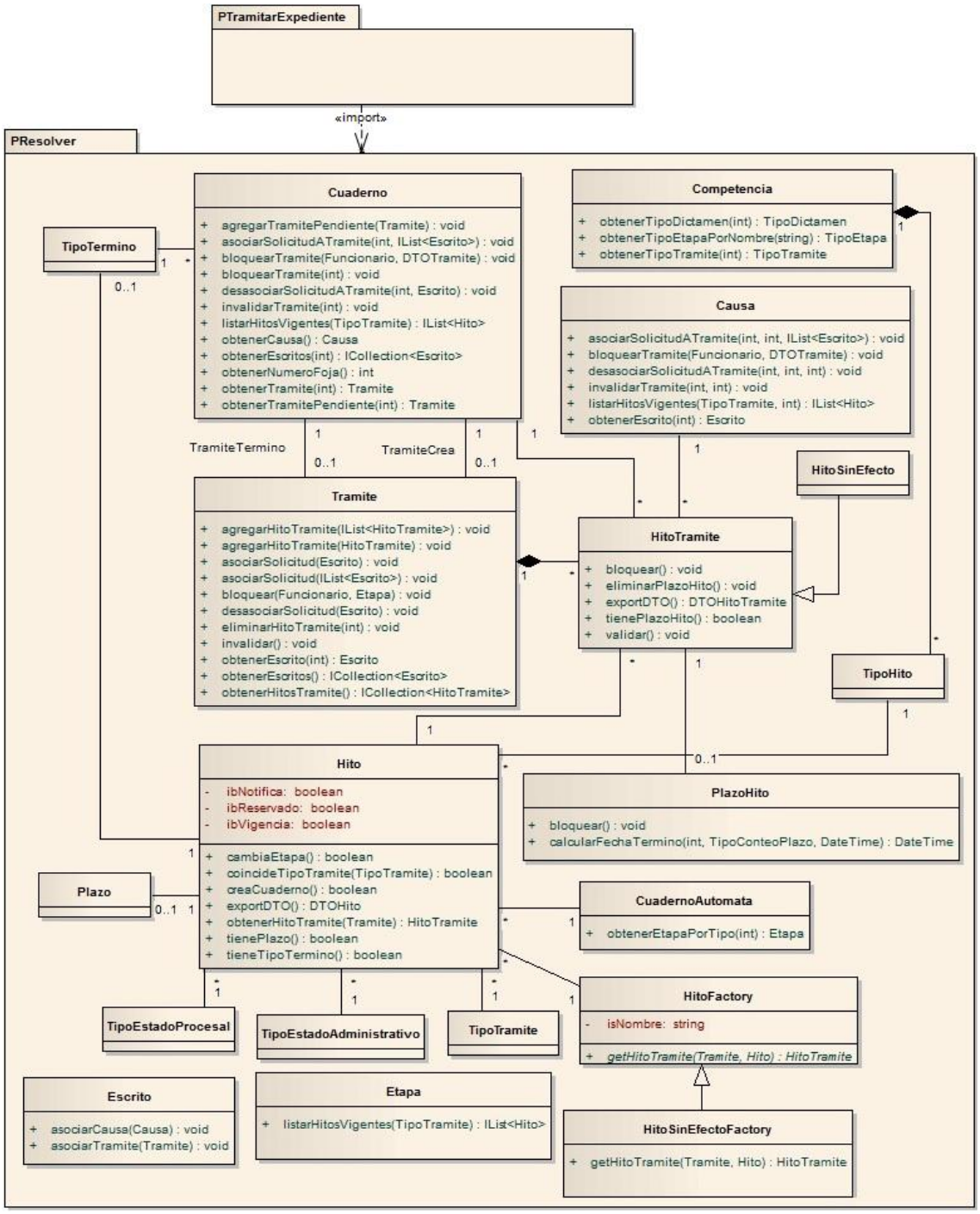


Figura 3-39: Mecanismo de "*package merge"* aplicado al diagrama de clases de la iteración 3 (paquete resolver).

Al diseñar el diagrama de clases de la característica tramitar expediente, se identificaron muchos puntos de variación que pueden implementarse a futuro en la línea de productos, como por ejemplo un cuaderno actualmente tiene trámites reservados, nulos, bloqueados, inválidos y pendientes, sin embargo, esta característica puede ser opcional para ciertas organizaciones donde sus procesos no necesariamente manejan estos cinco conceptos. O como por ejemplo en el paquete Resolver existe una fábrica de hitos (clase HitoFactory), actualmente todos los hitos (6.18) no contienen lógica de

negocio por si solos, sólo representan un nemotécnico definido para una resolución o actuación. Sin embargo, el esquema propuesto de combinación de paquetes (*package merge*) permite evolucionar a futuro la línea agregando un paquete por cada hito con su propia lógica de negocio mezclándose al paquete principal Resolver sin impactarlo. Este caso se ve reflejado en el hito “Citación audiencia preparatoria” actualmente el sistema sólo representa el hito, pero a futuro podría levantarse un pop-up con una agenda de la organización que permite gestionar las salas disponibles para realizar las audiencias. Este último punto era uno de los problemas actuales que tiene la organización respectó a la evolución del producto; ampliando las funcionalidades por iniciativa propia.

3.6.4.8 Implementación

Como se mencionó en la primera iteración, la implementación de las características de la línea de productos software se realizó utilizando Microsoft Visual Studio 2008 y el *plug-in* FMT (5).

3.6.4.8.1 Variabilidad en la capa de presentación

Para esta tercera iteración, se diseñaron dos páginas ASPX, la primera llamada LEX_TRA_TramitarExpediente.aspx correspondiente a la característica Tramitar Expediente y la segunda llamada LEX_TRA_Resolver.aspx correspondiente a la característica obligatoria resolver. En la página LEX_TRA_TramitarExpediente.aspx, que corresponde a la característica tramitar expediente, contiene tres controles de usuarios para representar la variabilidad que tiene esta característica. Como se visualizan en los recuadros rojo de la Figura 3-40 y Figura 3-41, los controles de usuario son incluidos en la página principal si algunas de las características opcionales es seleccionada al derivar en un producto concreto.

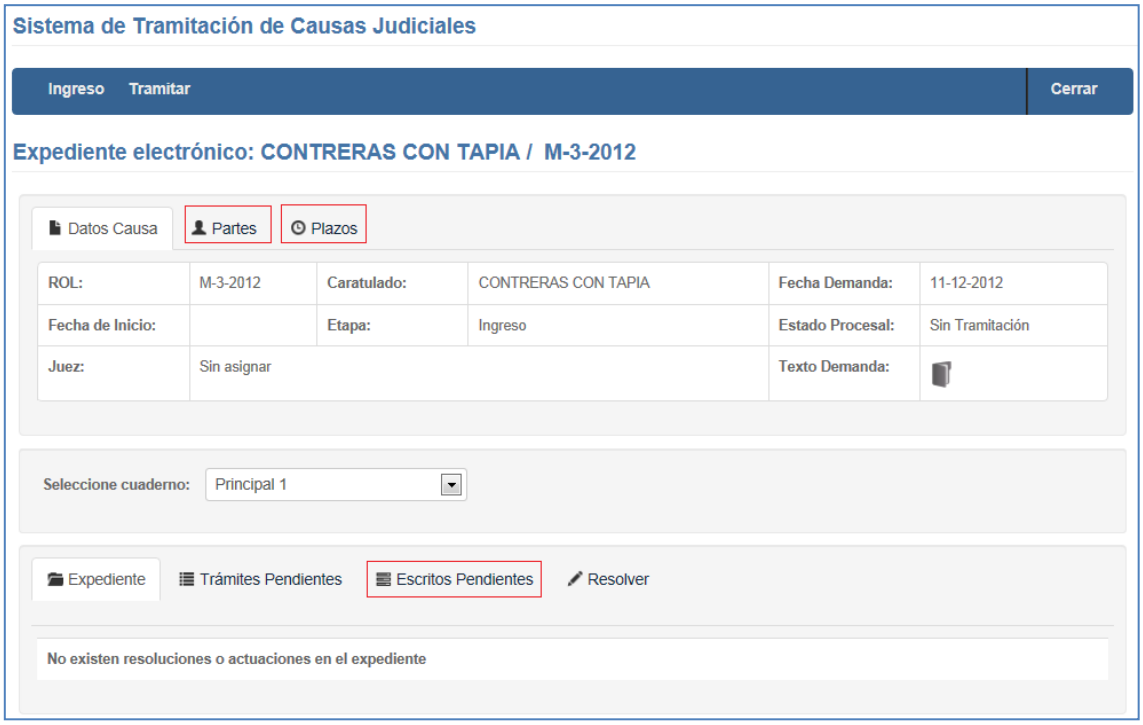


Figura 3-40: Página LEX_TRA_TramitarExpediente.aspx seleccionado las características opcionales ver partes, ver plazos y visualizar escritos.

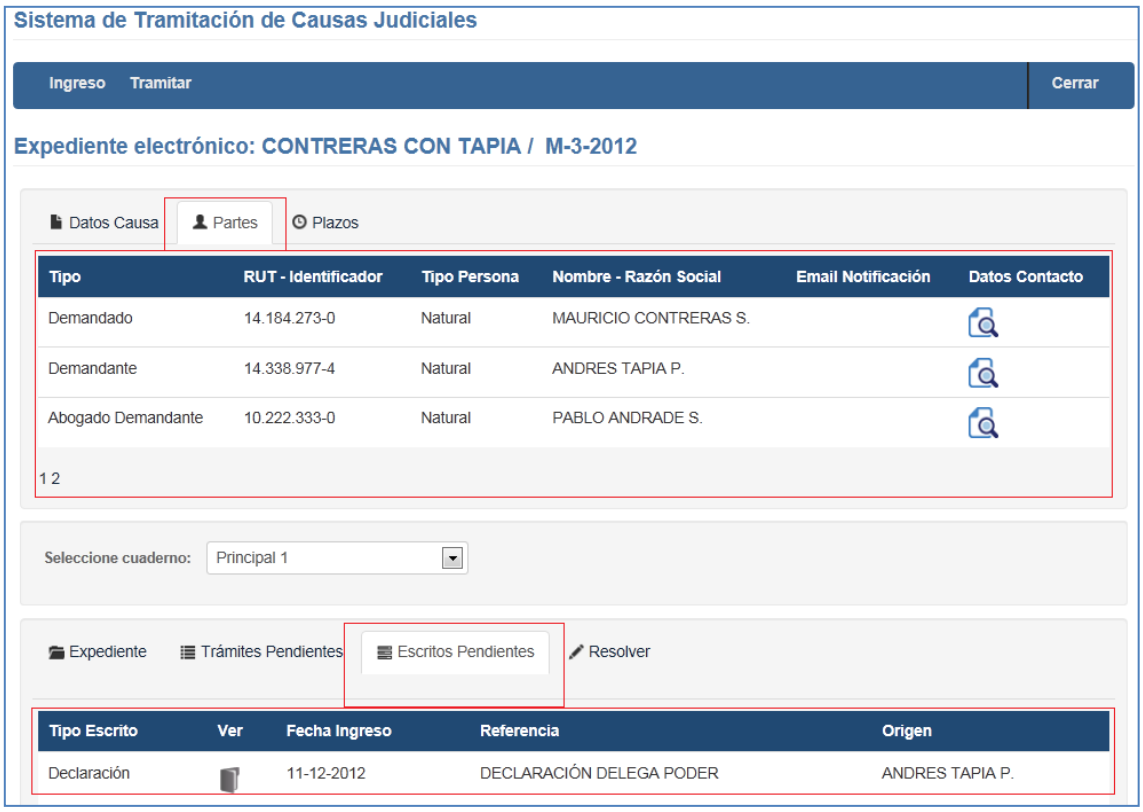


Figura 3-41: Interfaz de usuario de las características opcionales ver partes y visualizar escritos.

Por cada característica opcional se creó un control de usuario que contiene las grillas con la información respectiva y otra con el encabezado. Esta separación se realizó ya que el esquema de ocultar y visualizar la información al usuario se realiza con controles

tipo *panel* de ASP.NET, lo que permite al usuario dar la sensación de ir navegando a través de las pestañas que tiene la página principal LEX_TRA_TramitarExpediente.aspx.

Para añadir en tiempo de ejecución los controles de usuario LEX_TRA_VerPartes.ascx, LEX_TRA_VerPlazos.ascx y LEX_TRA_VisualizarEscritosPend.ascx, tal como la primera iteración, se utiliza un control de ASP.NET llamado *PlaceHolder*. En la página LEX_TRA_TramitarExpediente.aspx se agregaron seis *PlaceHolder* para incluir las características opcionales y para determinar si estos controles de usuario se deben incluir o no, se creó un archivo XML el cual contiene información del paquete que se debe incluir y el control de usuario que interactúan con el paquete Tramitar Expediente. El archivo XML contiene la información, si el paquete alternativo debe o no ser incluido al momento de cargar la página principal, a través de un *tag* llamado *activado*, como se visualiza en la Figura 3-42. Si el *tag* devuelve un valor *true* entonces el control de usuario se debe cargar, en caso contrario no es cargado en la página principal.

TramitarExpediente.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<paquetes>
  <paquete>
    <nombre>../VerPartes</nombre>
    <activado>true</activado>
    <archivos>
      <archivo>LEX_TRA_VerPartes.ascx</archivo>
    </archivos>
  </paquete>
  <paquete>
    <nombre>../VerPartes</nombre>
    <activado>true</activado>
    <archivos>
      <archivo>LEX_TRA_HeadVerPartes.ascx</archivo>
    </archivos>
  </paquete>
  <paquete>
    <nombre>../VerPlazos</nombre>
    <activado>true</activado>
    <archivos>
      <archivo>LEX_TRA_VerPlazos.ascx</archivo>
    </archivos>
  </paquete>
```

Figura 3-42: Archivo TramitarExpediente.xml

Para la característica obligatoria resolver se creó una página independiente llamada LEX_TRA_Resolver.aspx, ya que según el modelo de casos de uso, el caso de uso Resolver es alternativo al caso de uso tramitar expediente, por ende no era necesario un control de usuario, debido a que el usuario decide en qué momento dicta una resolución o actuación. La interfaz de usuario de esta característica se visualiza en la Figura 3-43, la cual se divide en 4 pasos que debe completar el usuario (juez o funcionario). El primer paso consisten en definir si la resolución es a petición de las partes, es decir que resuelve un escrito presentado por las partes de la causa, o de oficio, es decir, por iniciativa propia de la organización. Luego, el segundo paso se

activa solamente si en el primer paso se selecciona la opción a petición de parte. En caso contrario se continúa con el tercer paso, donde el usuario tipifica la resolución o actuación agregando los hitos que resuelve. Estos hitos pueden ser los disponibles en la etapa que se encuentra el procedimiento judicial o en la etapa general que son aquellos hitos disponibles en todas las etapas del procedimiento. El último paso consiste en adjuntar el documento de la resolución o actuación y guardar la dictación. Si la resolución o actuación no se sube al expediente digital, entonces queda como un trámite pendiente en el expediente digital.

Una de las características que cuenta la mayoría de los sistemas desarrollados por la organización, respecto a sistemas de tramitación de causas judiciales, tienen la opción de notificar las partes, al momento de dictar un trámite, por diversos mecanismos, ya sea a través de la misma organización, por carta certificada, por email o por centros de notificación especializados. Esta característica actualmente no está incluida dentro del proyecto de tesis, sin embargo, dado el mecanismo propuesto permitiría gestionar en forma eficiente la variabilidad de esta característica.

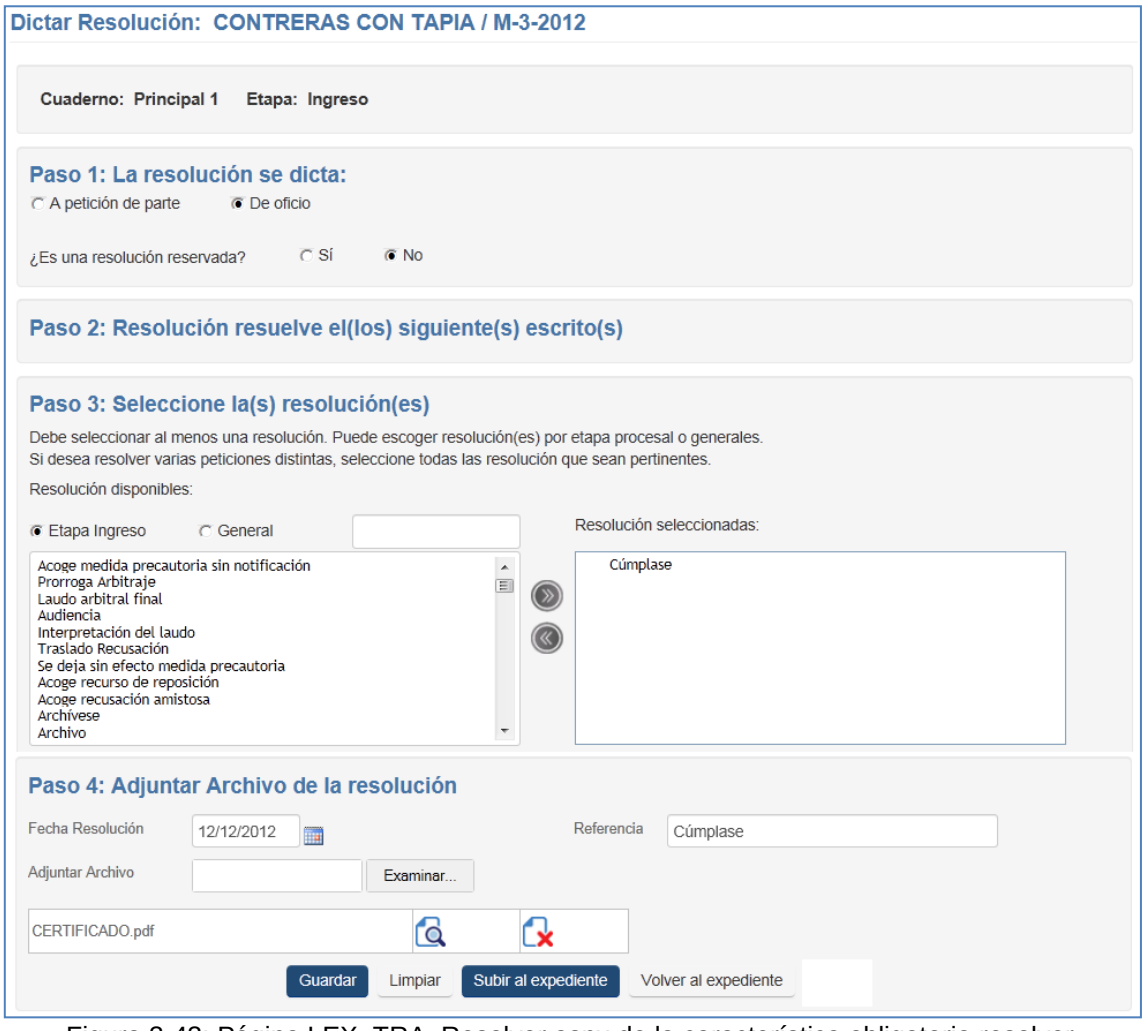


Figura 3-43: Página LEX_TRA_Resolver.aspx de la característica obligatoria resolver.

3.6.4.8.2 Variabilidad en la capa de dominio

Para extender la trazabilidad desde las características hasta el nivel de implementación, se utiliza el concepto de clases parciales (30), tal como las iteraciones anteriores.

En la Figura 3-44 se visualiza la implementación de la capa de dominio para esta tercera iteración. En el paquete tramitar expediente se almacena la capa de presentación que se implementa con la página LEX_TRA_TramitarExpediente.aspx como se mencionó en el punto anterior, la cual se comunica con la clase LEX_DEL_Tramitar.cs del paquete Delegate que permite desacoplar la capa de presentación con la de dominio. Para la característica opcional ver partes no existe implementación de la capa de dominio, ya que se reutiliza el método *listarPartes* de la clase LEX_DEL_Ingreso.cs que contiene la misma funcionalidad requerida por este paquete y que fue implementada en la primera iteración.

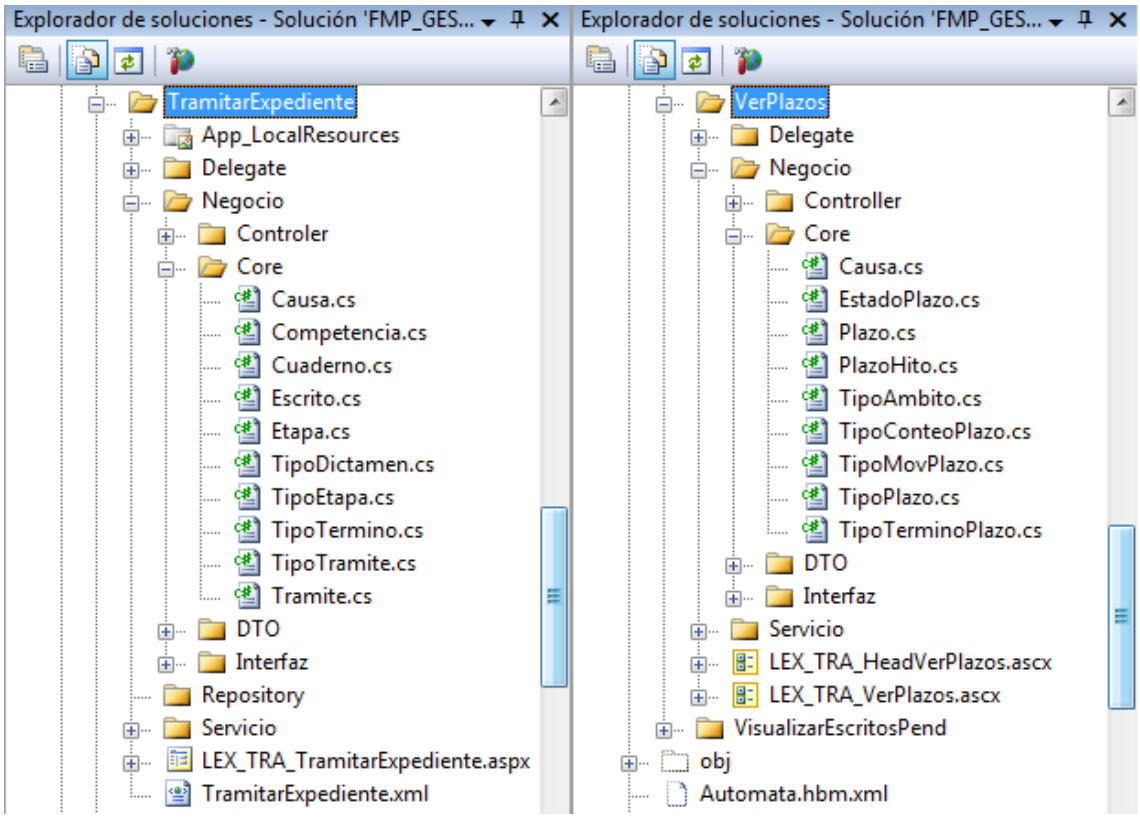


Figura 3-44: Implementación de la capa de dominio, paquetes tramitar expediente y ver plazos.

Al implementar la capa de dominio en esta tercera iteración, de acuerdo con la arquitectura y el esquema de las iteraciones anteriores, se logró validar la implementación de clases parciales debido a que fueron útiles para la separación de código y desarrollo modular de las funcionalidades de esta característica. Puesto que, las clases parciales, significan un conjunto de comportamientos fácilmente localizables o separables de las funciones básicas de la línea de productos. En contraste con el mecanismo de herencia, que representa una forma de especialización, al incluir una clase parcial se introducen cualidades adicionales por medio de la definición de sus

comportamientos. Por tanto, se ha empleado más como una técnica que aumenta el grado de modularidad de la línea de producto, pudiendo construir características de la línea de productos separados combinables con otras.

Uno de los problemas presentados en esta iteración respecto a la extensibilidad, es que si bien C# permite agregar nuevos atributos y métodos a las clases existentes, no es posible extender ni sobrescribir métodos existentes, ya que el compilador genera un error indicando que el método se encuentra duplicado. Esto significa que, no es posible dividir la implementación de un método en varias clases parciales, ya que lo que permite C#, en su última versión, es sólo declarar el encabezado del método como parcial en varias clases parciales pero la implementación debe estar siempre en una sola clase.

Este problema se presento en la clase Escrito del paquete tramitar expediente, donde se agregó un atributo llamado “ioTramiteResuelve” que permite identificar que trámite resuelve el escrito presentado en por las partes, como se visualiza en la Figura 3-45. El atributo se debía incluir en el método exportarDTO de la misma clase que permite devolver una clase DTO a las capas superiores.

```
namespace FMP_GESLEX.ExpedienteDigital.Negocio.Core
{
    public partial class Escrito
    {
        [BelongsTo]
        public virtual Tramite ioTramiteResuelve { get; set; }

        public virtual bool tieneTramiteAsociado()
        {
            return null != this.ioTramiteResuelve;
        }
    }
}
```

Figura 3-45: Clase parcial Escrito del paquete tramitar expediente.

Sin embargo, no es factible sobrescribir el método, tal como se mencionó anteriormente, teniendo que implementar como solución incluir la línea de código obligatoriamente en el paquete Ingresar Escrito donde se encontraba declaro dicho método de la clase Escrito, como se visualiza en la Figura 3-46. Esta solución es factible ya que el paquete Ingresar Escrito es obligatorio en la línea de productos. Claramente esto representa una desventaja respecto a la extensibilidad, ya que no es factible extender la implementación de métodos ya existentes, debido a que C# es un lenguaje orientado a objetos y no a características.

```
public virtual DTOEscrito exportDTO()
{
    DTOEscrito loCauDTO = new DTOEscrito()
    {
        ilId = this.escritoId,
        idFecIngreso = this.idFecIngreso,
        isTipoSolicitud = this.ioTipoSolicitud.isDescripcion,
        isSolicitante = (null == this.ioPartePresenta ? "" : this.ioPa
        isTipoSolicitante = (null == this.ioPartePresenta ? "" : this.
        iiNumeroFoja = this.iiNumeroFoja,
        ibReservado = this.ibReservado,
        isReferencia = this.isReferencia,
        ilIdCausa = (null == this.ioCausa ? 0 : this.ioCausa.causaId),
        isFuncionarioIngresa = this.ioFuncionarioIng.obtenerNombreComp
        isEtapa = this.ioEtapaCrea.ioTipoEtapa.isDescripcion,
        ilIdTramiteResuelve = (null == this.ioTramiteResuelve ? 0 : th
    };
    return loCauDTO;
```

Figura 3-46: Clase parcial Escrito del paquete ingresar escrito.

3.6.4.8.3 Variabilidad en la capa de infraestructura de acceso a datos

La variabilidad relacionada a la capa de infraestructura de acceso a datos se implementa con el framework Castle ActiveRecord (33) que es una capa implementada por encima de NHibernate. Éste fue un cambio importante que se optó, a diferencia de los productos ya desarrollados por la organización que implementaban sólo NHibernate configurando manualmente los archivos XML para realizar el mapeo entre una clase y la estructura de la tabla en la base de datos. Como ventaja en el desarrollo de esta iteración, se observó que a medida que se agregaron clases parciales con sus nuevos atributos el impacto en los paquetes ya desarrollados fue nulo. En contraste con el esquema de NHibernate, la gran queja que siempre presentaban los desarrolladores era que cada vez que se modificaban los modelos implicaba un re trabajo en los módulos ya desarrollados, debido a relaciones incorrectas o dependencias que no estaban establecidas. El esquema propuesto, permite incrementalmente agregar nuevas características sin romper lo ya se ha desarrollado. Esta afirmación permite resolver uno de los problemas actuales que existen en la organización respecto al extender con variaciones el código fuente de cada nuevo producto dando como resultado que el código este propenso a errores y sea poco robusto.

3.6.5 Iteración 4: Login

Para esta última iteración se diseñó y construyó la característica login. Esta característica tiene como principal funcionalidad permitir ingresar al sistema e iniciar una sesión válida. La forma de validar al usuario es a través de su nombre de usuario y password.

3.6.5.1 Análisis y diseño de la característica Login

Como se detalló en la primera iteración, la variabilidad se expresa utilizando el concepto de combinación de paquetes (*package merge*) de UML, según la propuesta de Miguel

Laguna y Bruno González (2). La aplicación de este esquema respecto a la cuarta y última iteración, se visualiza en la Figura 3-47.

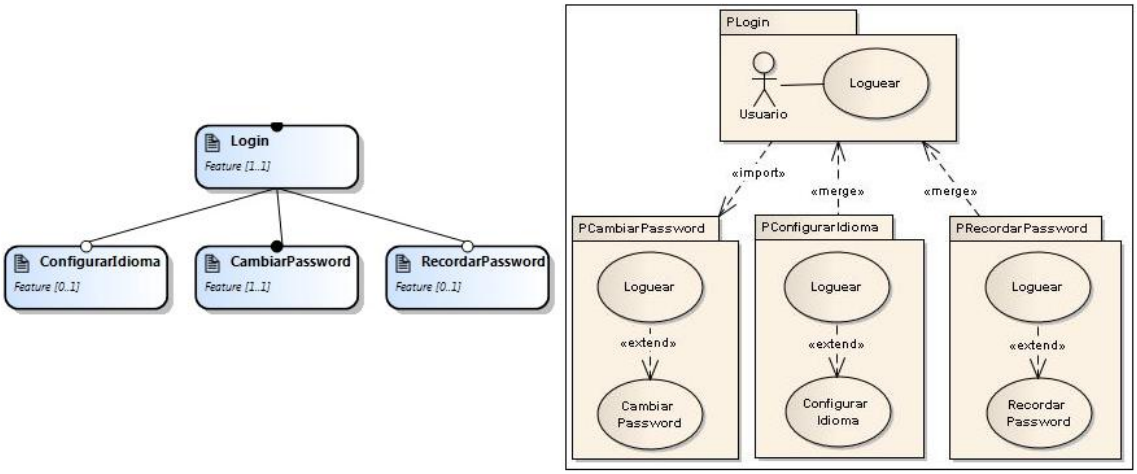


Figura 3-47: Modelo de Característica de la Iteración 4 y el mecanismo de "*package merge"* aplicado al modelo de casos de uso.

En el modelo de característica, se detalla la variabilidad que puede adoptar la característica Login para un producto concreto de tramitación de causas judiciales, indicando claramente las que son obligatorias y opcionales. La obligatoria es cambiar password y las opcionales son recordar password y configurar idioma. Si las características opcionales están presentes, al cargar la página principal de Login, se visualiza un listado de idiomas disponibles que puede trabajar el sistema y la opción de recordad la password en caso que el usuario la olvido.

En la Figura 3-48 se representa la jerarquía de la característica login.

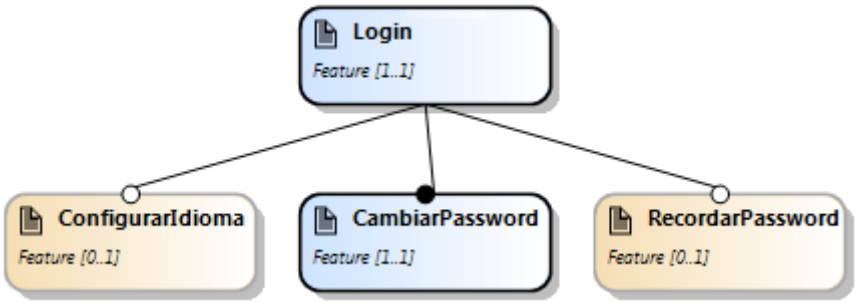


Figura 3-48: Esquema de la característica login.

El modelo básico de la característica Login de la línea de productos se visualiza en la Figura 3-49. Adoptando la propuesta Miguel Laguna y Bruno González (2), los paquetes de la parte inferior son opcionales e independientes entre sí. Para esta cuarta iteración se desarrollaron los siguientes paquetes:

- **PLogin**: Contiene la funcionalidad de permitir al usuario ingresar al sistema e iniciar una sesión válida.
- **PCambiarPassword**: Contiene la funcionalidad de cambiar la password actual del usuario.

- **PConfigurarIdioma**: Contiene la funcionalidad de permitir al usuario cambiar el idioma de su sesión en el sistema.
- **PRecordarPassword:** Contiene la funcionalidad permitir al usuario recuperar la password para ingresar al sistema en caso de haberla olvidado.

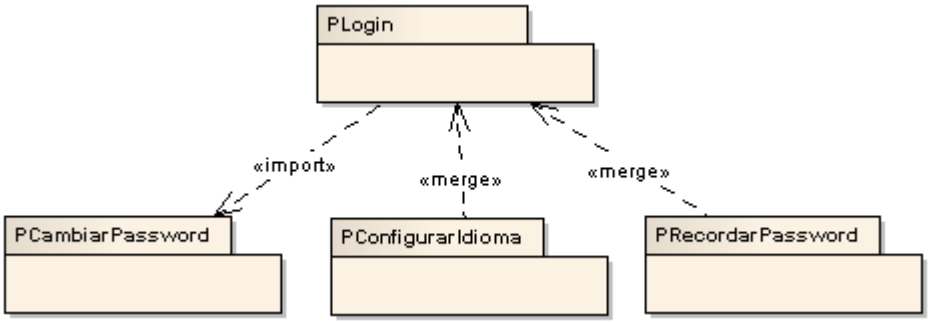


Figura 3-49: Esquema de jerarquía del paquete login.

En la Figura 3-50 se visualiza el modelo de casos de uso de la característica login. El paquete login importa el paquete cambiar password y se mezcla con el paquete recordar password y configurar idioma. Si se selecciona el paquete recordar password, se incluye en el modelo un caso de uso adicional como extensión que permite al usuario digitar su nombre de usuario para que el sistema envíe una nueva password al email registrado. Si se selecciona el paquete configurar idioma, se incluye en el modelo de caso de uso adicional como extensión que permite configurar en el idioma español o inglés la sesión del usuario.

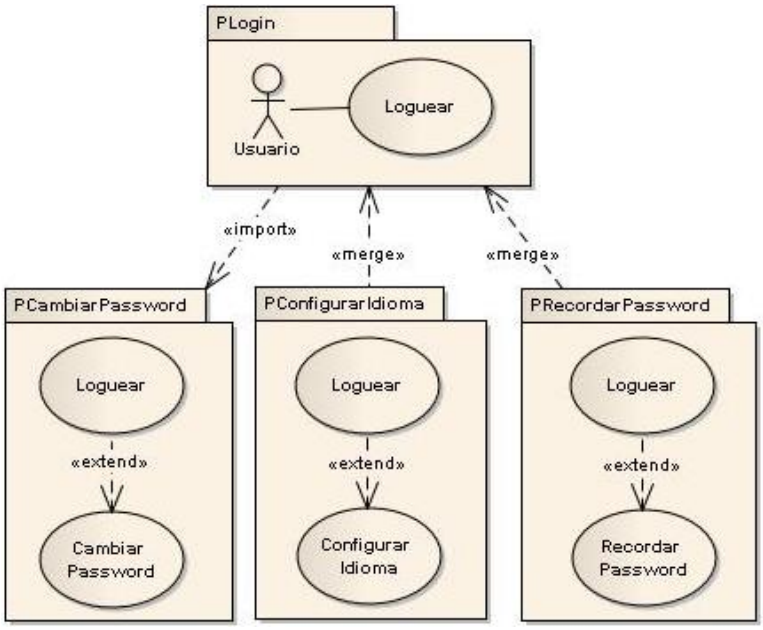


Figura 3-50: Paquetes de la característica login (vista casos de uso).

A continuación se desarrollan los paquetes identificados de la característica manteniendo el esquema del desarrollo dirigido por casos de uso.

3.6.5.2 Paquete login

El paquete login se encarga de la funcionalidad de permitir al usuario ingresar al sistema e iniciar una sesión válida. La forma de validar al usuario es mediante el ingreso de su nombre de usuario y contraseña.

3.6.5.2.1 Caso de uso: login

Flujo de Eventos	
Principal	1. El usuario abre un navegador Web e ingresa la URL del sistema. 2. El sistema muestra la página de ingreso al sistema. 3. El usuario ingresa su nombre de usuario y password. 4. El usuario solicita iniciar una sesión. 5. El sistema valida los datos del usuario 6. El sistema carga el perfil de usuario e inicia una sesión en el sistema. 7. El sistema muestra un mensaje de bienvenida al usuario.
Alternativo Cambiar Password	3.1 El usuario selecciona la opción de cambiar su password. Se ejecuta caso de uso "Cambiar Password".
Excepción Password Incorrecta	6.1 La password ingresada por el usuario es incorrecta. El sistema muestra un mensaje de error al usuario indicando el error.
Excepción Datos incompletos	4. El usuario trata de ingresar al sistema sin haber ingresado su nombre de usuario y/o password. El sistema muestra un mensaje de advertencia al usuario solicitando que ingrese los datos faltantes.
Excepción Usuario no existe	6.1 El nombre de usuario ingresado no se encuentra registrado en el sistema. El sistema muestra un mensaje de error al usuario indicando el error.

3.6.5.3 Paquete cambiar password

El paquete cambiar password se encarga de la funcionalidad de permitir al usuario cambiar su password actual.

3.6.5.3.1 Caso de uso: cambiar password

Flujo de Eventos	
Principal	1. El usuario solicita cambiar su password actual. 2. El sistema despliega la información necesaria para que el usuario cambie su password. Esta información es: - Password actual - Nueva Password - Confirmación nueva Password. 3. El usuario ingresa los datos solicitados y solicita cambiar la password. 4. El sistema valida los datos ingresados y cambia la password del usuario. 5. El sistema muestra un mensaje indicando el cambio exitoso de la Password.
Excepción Password actual	4.1 La password actual ingresada por el usuario es incorrecta. El sistema muestra un mensaje indicando el error.

incorrecta	4.2 Vuelve al punto 2.
Excepción Contraseña nueva y confirmación contraseña son distintas	4.1 La nueva password ingresada por el usuario no es igual a la ingresada en el campo confirmación nueva password. El sistema muestra un mensaje indicando el error. 4.2 Vuelve al punto 2.
Prerrequisito Sesión válida	El usuario debe haber ingresado al sistema con su nombre de usuario y contraseña.

3.6.5.4 Paquete recordar password

El paquete recordar password se encarga de la funcionalidad de permitir al usuario recuperar su password para ingresar al sistema en caso de haberla olvidado.

3.6.5.4.1 Caso de uso: recordar password

Flujo de Eventos	
Principal	1. El usuario selecciona la opción "Recordar Password". 2. El sistema muestra la información necesaria para recordar la password. Esta información es el nombre de usuario. 3. El usuario ingresa su nombre de usuario. 4. El sistema valida el nombre de usuario y genera una nueva password temporal. 5. El sistema envía la password temporal al correo electrónico del usuario registrado en el sistema. 6. El sistema muestra un mensaje informativo al usuario indicando que la password ha sido enviada a su correo electrónico.
Excepción Usuario no existe	4.1 El nombre de usuario ingresado no se encuentra registrado en el sistema. El sistema muestra un mensaje de error al usuario indicando el error. 4.2 Vuelve al punto 2.

3.6.5.5 Paquete configurar idioma

El paquete configurar idioma se encarga de la funcionalidad de permitir al usuario cambiar el idioma de su sesión en el sistema. Los idiomas disponibles son inglés y español.

3.6.5.5.1 Caso de uso: configurar idioma

Flujo de Eventos	
Principal	1. El sistema despliega los idiomas disponibles. Estos son: - Español (por defecto) - Inglés 2. El usuario selecciona el idioma con el cual desea trabajar en el sistema. 3. El sistema desplegará todas las páginas del sistema en el idioma seleccionado.

3.6.5.6 Diagrama de clases

Una vez terminados todos los diagramas de colaboración para la realización de los casos de uso del paquete login, recordar password, configurar idioma y cambiar password se identificaron las especificaciones de las clases e interfaces que participan cuando la característica login es seleccionada.

En la Figura 3-51 se visualiza el diagrama de clases de los paquetes de la característica login que permite capturar todas las clases parciales, visualizando los métodos posibles que pueden tener cada clase cuando es seleccionada esta característica. En el paquete login se visualiza que el funcionario, es decir, el usuario contiene una colección de perfiles que es representado a través de una jerarquía de todos los tipos de perfiles que puede tener el funcionario. A su vez la clase perfil contiene una asociación a la clase tipo perfil que contiene una colección de opciones de perfiles que son las páginas ASPX que debe cargar el menú del sistema cuando el usuario inicia su sesión.

Tal como se realizó en las iteraciones anteriores, al aplicar el mecanismo *package merge* (2), el resultado de combinar estos paquetes es que aparecen por ejemplo una clase final Funcionario con los métodos *exportDTO, crearPassword y cambiarPassword*.

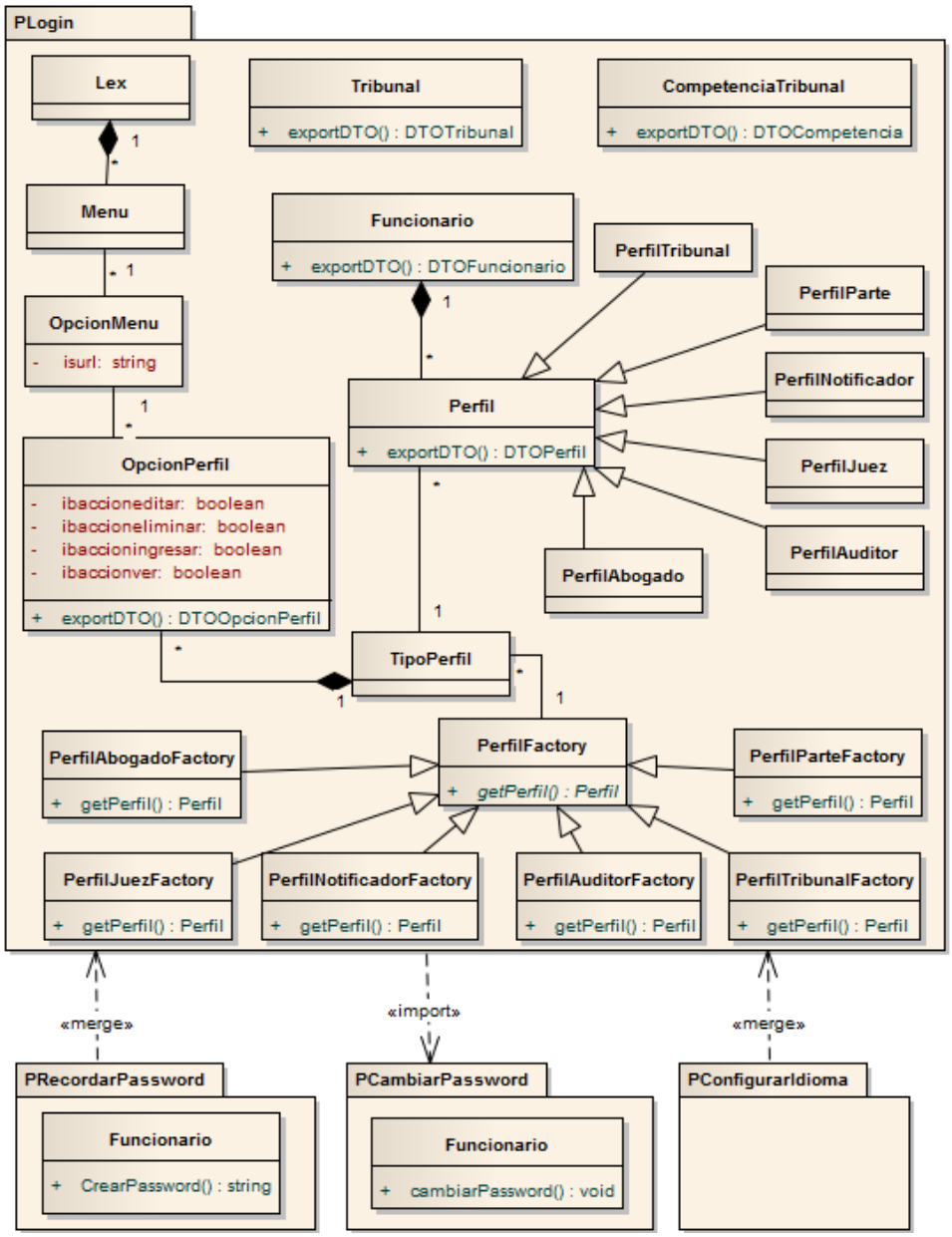


Figura 3-51: Mecanismo de "*package merge"* aplicado al diagrama de clases de la iteración 4.

3.6.5.7 Implementación

Como se mencionó en la primera iteración, la implementación de las características de la línea de productos software se realizó utilizando Microsoft Visual Studio 2008 y el *plug-in* FMT (5).

3.6.5.7.1 Variabilidad en la capa de presentación

Para esta cuarta iteración, se diseñó una página para el inicio de sesión de los usuarios, llamada LEX_Login.aspx correspondiente a la característica login. La segunda página que se diseñó, corresponde a la característica obligatoria cambiar password llamada LEX_CambiarPass.aspx.

Para representar la variabilidad en la capa de presentación de la característica Login, en la página LEX_Login.aspx se insertaron dos objetos del tipo *PlaceHolder* que permiten en tiempo de ejecución cargar los controles de usuario de las características opcionales. Se utiliza la misma técnica que en las iteraciones anteriores, donde existe un archivo XML que contiene la información de los controles de usuario que deben ser cargados o no en tiempo de ejecución en el paquete principal.

3.6.5.7.2 Variabilidad en la capa de dominio

Para extender la trazabilidad desde las características hasta el nivel de implementación, se utiliza el concepto de clases parciales (30), tal como las iteraciones anteriores.

En la Figura 3-52 se visualiza la implementación de la capa de dominio para esta última iteración. En el paquete Login se almacena la capa de presentación que se implementa con la página LEX_Login.aspx como se mencionó en el punto anterior, la cual se comunica con la clase LEX_DEL_Login.cs del paquete Delegate que permite desacoplar la capa de presentación con la de dominio. Para la característica opcional Configurar Idioma no existe implementación de la capa de dominio, ya que la traducción es realizada por la capa de presentación.

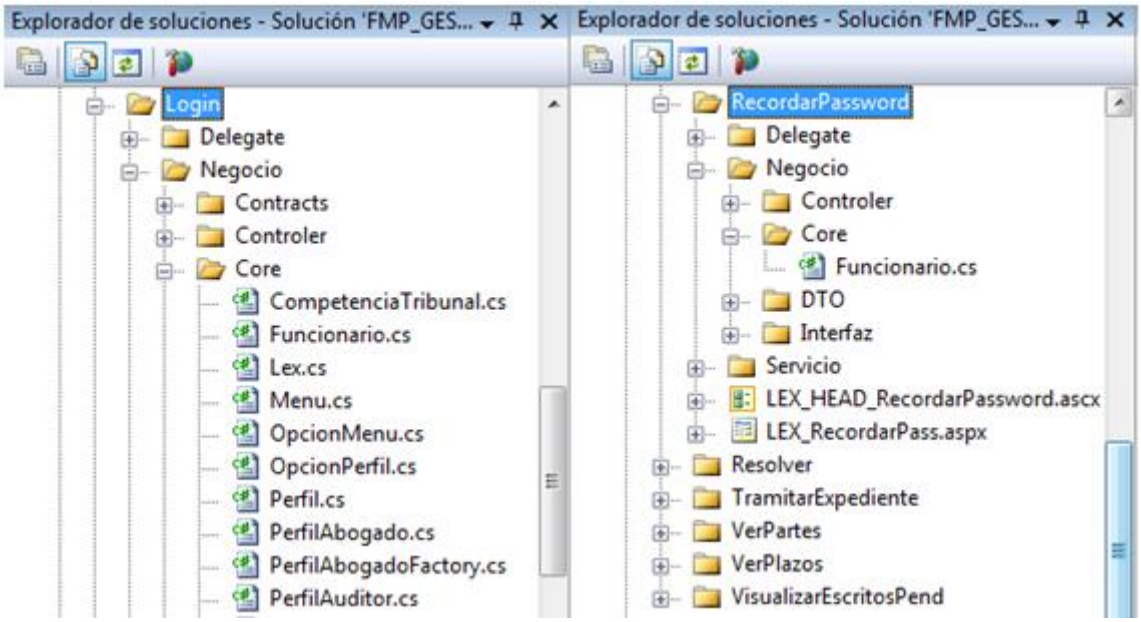


Figura 3-52: Implementación de la capa de dominio de la cuarta iteración.

3.7 Configuración de productos específicos

Como un último punto de la implementación, es permitir configurar los paquetes implementados en todas las iteraciones, que representan las características de la línea de productos, para derivar en un producto concreto de la forma más automatizada posible. Esto se logra al trabajar con el *plug-in* FMT (*Feature Modeling Tools*) (5) que permite integrar actividades del desarrollo de la línea de productos en un mismo entorno de manera automatizada, uno de los objetivos de esta tesis. Estas actividades son la creación del diagrama de característica que previamente se había realizado, para

posteriormente gestionar las configuraciones de los productos concreto en forma integrada en Microsoft Visual Studio 2008, ya que por cada característica creada en el diagrama, es representada por un paquete dentro de la solución web del proyecto.

Para derivar en un producto concreto basta con seleccionar las características que se van o no a incluir en la vista del configurador de características y presionar el botón *Generate Project,* como se visualiza en la Figura 3-53. Dando como resultado un producto final, donde sólo se concentraran las adaptaciones que solicita el cliente.

Esta funcionalidad es vital a la hora de gestionar de forma eficiente la creación de nuevos productos a partir de la línea de productos, ya que permite asegurar de que las composiciones de un conjunto características siempre deriven en productos correctos. En caso contrario, se debería recurrir a esquemas manuales que generalmente están propensos a errores del desarrollador o arquitecto.

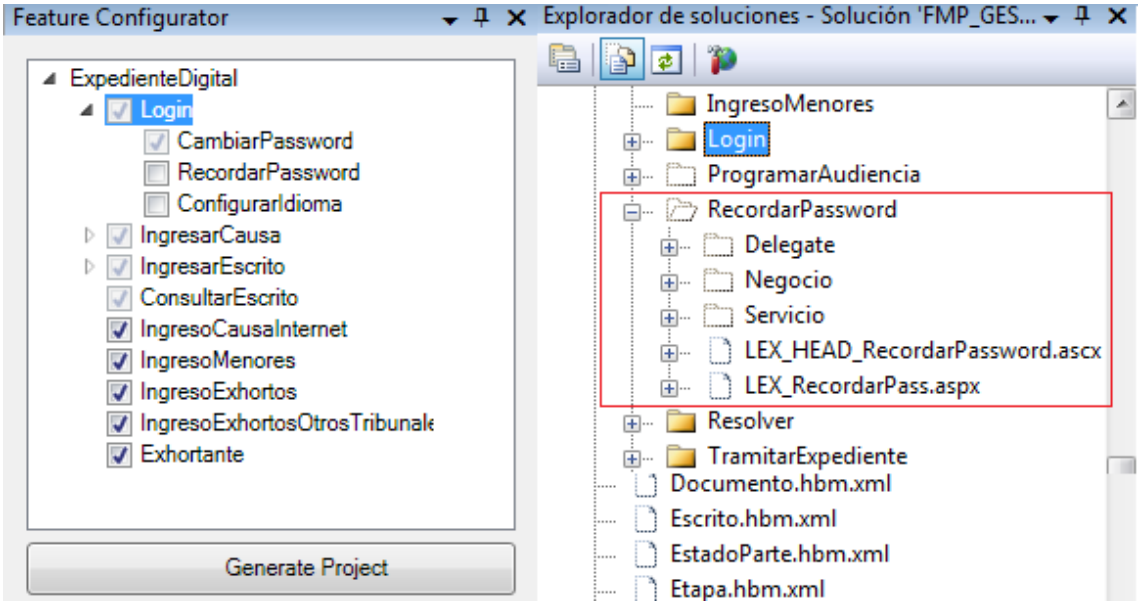


Figura 3-53: Configurador de características

En la segunda iteración, la automatización de la configuración de los paquetes implementados cobra una relevancia importante, ya que la variabilidad de la característica ingresar escrito está dada por un grupo de características alternativas (XOR), es decir, del conjunto de características, sólo una de ellas será seleccionada si la característica ingresar escrito (padre) es seleccionada.

Se representan con un conjunto de arcos agrupados con cardinalidad 1.1. La multiplicidad de los arcos indica el número de características que deben ser seleccionadas. Esta multiplicidad está representada por un número mínimo y máximo entre corchetes. Como se visualiza en la Figura 3-54 esta automatización se logra con *plug-in* FMT (*Feature Modeling Tools*) (5) que establece las validaciones necesarias para la multiplicidad de los grupos de características y la selección de características obligatorias cuando el padre de la característica está seleccionada.

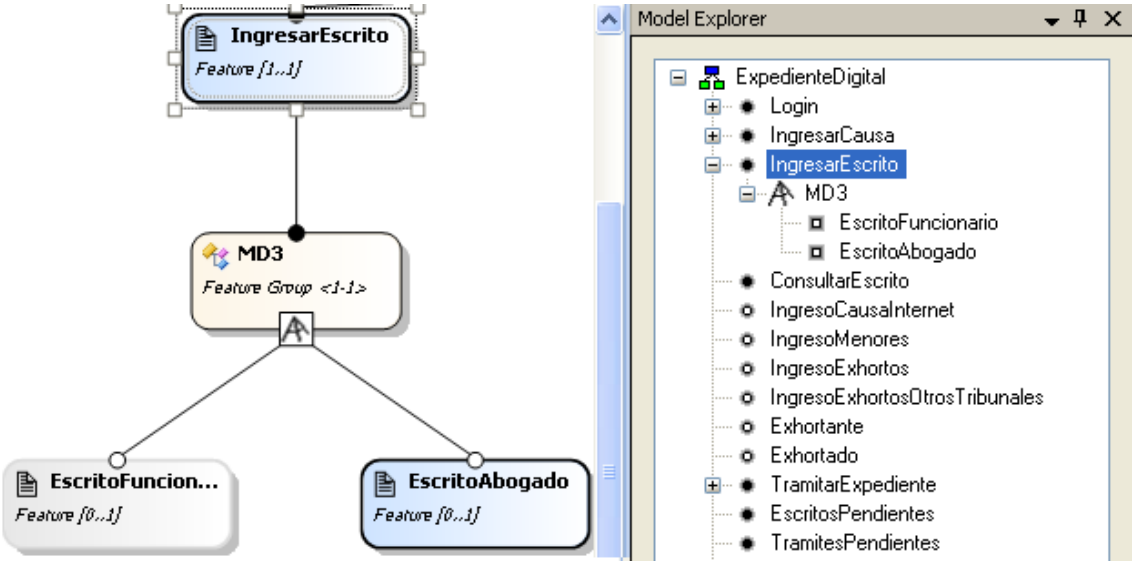


Figura 3-54: Interfaz del plug-in FMT con el diseño del modelo de característica de la segunda iteración.

Por ejemplo, si al seleccionar la característica escrito abogado el configurador de características del plug-in FMT (5) automáticamente excluye al paquete escrito funcionario, como se visualiza en la Figura 3-55. Logrando de esta forma que todos los componentes (controles de usuario y clases parciales) no estén incluidos en el proyecto web final.

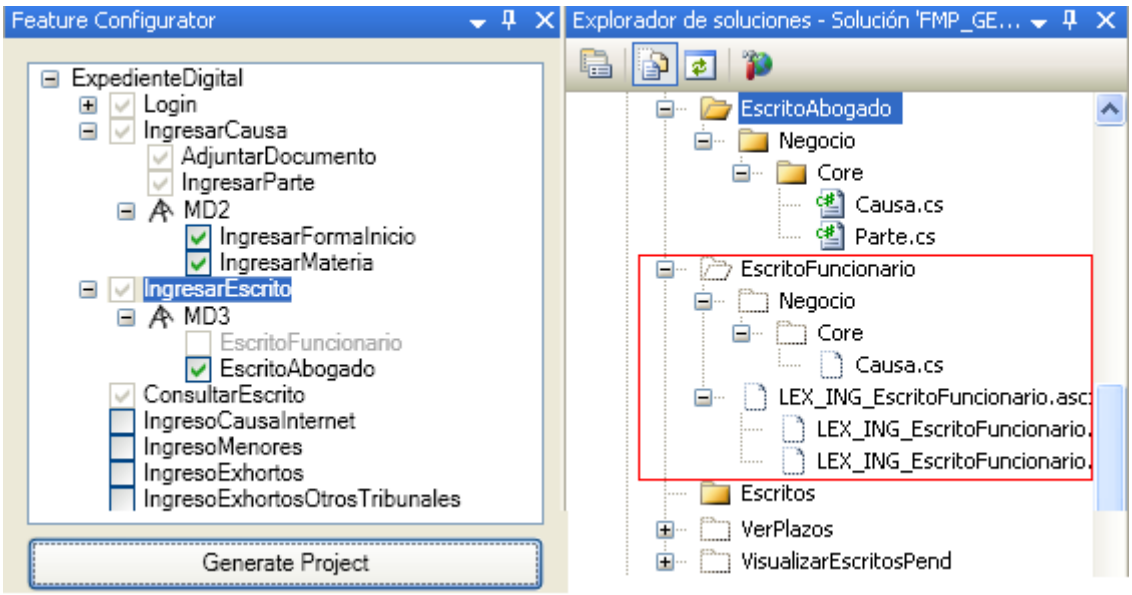


Figura 3-55: Configurador de características, segunda iteración.

Al proponer un mecanismo automatizado, como el *plug-in* FMT (5), se logra generar la capacidad de crear eficientemente múltiples variaciones del producto y de paso reducir los tiempos de entrega del proyecto.

3.7.1 Ejemplos de Productos Concretos

En la Figura 3-56 y Figura 3-57 se visualizan ejemplos de un producto concreto seleccionado la característica recordar password y configurar idioma de la última iteración que corresponden a la característica login y cuando estas no son seleccionadas.

Figura 3-56: Un ejemplo de un producto concreto seleccionado la característica login, recordar password y configurar idioma.

Figura 3-57: Un ejemplo de un producto concreto seleccionado sola característica login.

En la Figura 3-58 se visualiza un ejemplo de un producto concreto seleccionado distintas características de la primera iteración correspondiente a la característica ingresar causa.

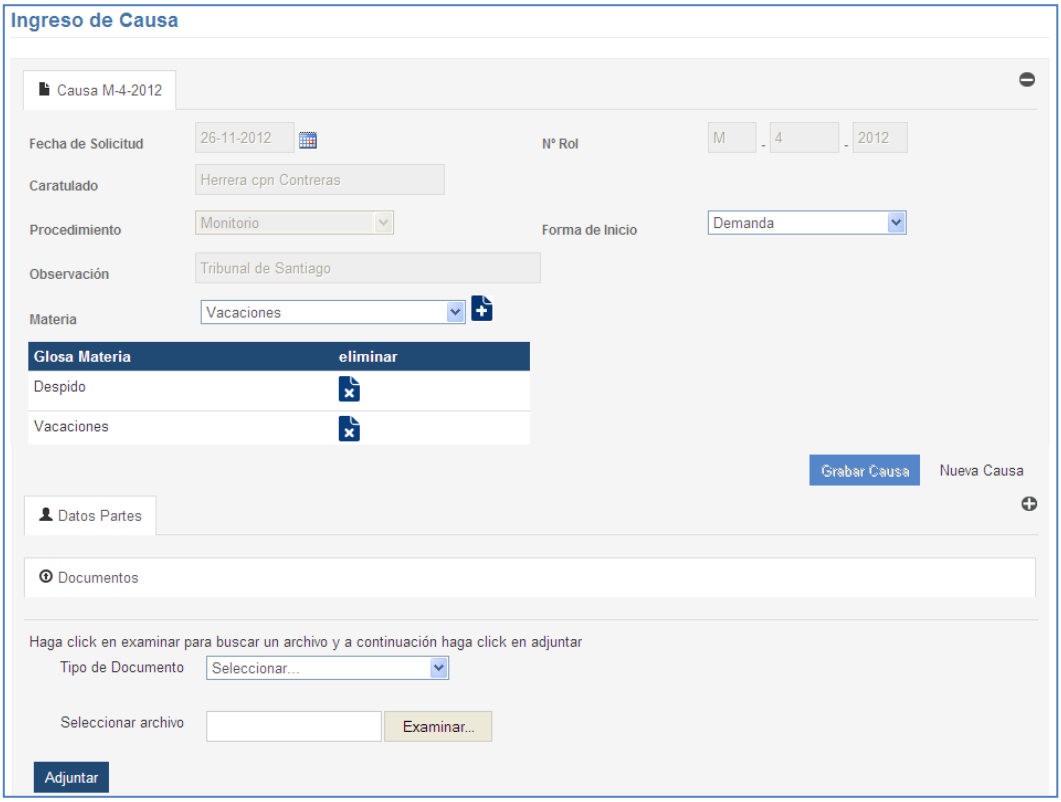


Figura 3-58: Ejemplo de un producto concreto seleccionado la característica ingresar causa y las características opcionales ingresar materia e ingresar forma de inicio.

En la Figura 3-60 se visualiza un ejemplo de un producto concreto seleccionado la característica escrito abogado que fue implementado en la segunda iteración correspondiente a la característica ingresar escrito, en la cual primero se busca la causa (funcionalidad realizada en la primera iteración Figura 3-59) y luego se ingresa el escrito.

Figura 3-59: Búsqueda de causas, funcionalidad de implementada en la primera iteración.

Figura 3-60: Un ejemplo de un producto concreto seleccionado la característica escrito abogado.

En la Figura 3-61 se visualiza un ejemplo de un producto concreto seleccionando la característica obligatoria tramitar expediente y las opciones visualizar escritos, ver partes y ver plazos.

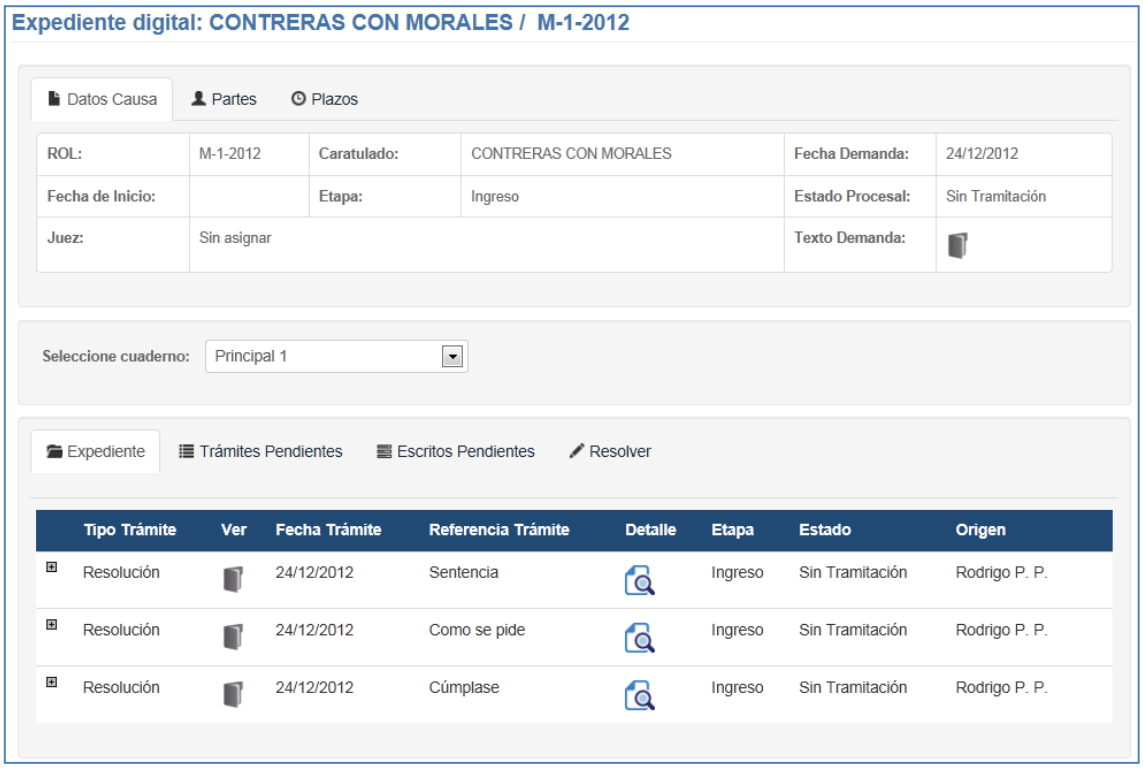


Figura 3-61: Un ejemplo de un producto concreto seleccionado la característica tramitar expediente.

3.8 Gestión de configuración

La gestión de configuración de software es la disciplina de la ingeniería de software que se encarga de gestionar el cambio de los proyectos de software. En cualquier desarrollo hay miles de cambios: código, documentos, diagramas, entregas y cualquier artefacto perteneciente al proyecto. La responsabilidad de la gestión de configuración es poder identificar cuándo se produce el cambio, por qué se produce y quién lo introduce (34).

En la actualidad, la organización implementa las prácticas del área CM (gestión de configuración) del nivel de madurez 2 de CMMI. Estas prácticas se cumplen principalmente a través de las herramientas CVSNT (35) para el control de versiones del código fuente y Project.net (36) para la documentación del proyecto. Sin embargo, no existe un esquema para la gestión de configuración de los activos que genere la línea de productos software, ya sea código, documento o cualquier otro artefacto. Por ende, lo que se busca es tener un esquema donde no se desestabilice la rama principal de desarrollo de la línea de productos software, corrompiendo la estabilidad, debido a los múltiples cambios que pueden subir los desarrolladores en el desarrollo las características.

El patrón utilizado por la organización para cualquier proyecto de software, es el desarrollo paralelo. En la Figura 3-62 se visualiza el esquema que adopta todos los proyectos de la organización.

Figura 3-62: Estructura de un proyecto con desarrollo paralelo

Este patrón, implementa el concepto de ramas (*branch*) que permite trabajar líneas paralelas de desarrollo, es decir, por cada desarrollador que integra el proyecto se le crea una rama. La ventaja de este esquema, es que el desarrollador puede realizar tantos *commits o check ins* como considere necesario durante el tiempo en que resuelve una tarea asignada por el Jefe de Proyectos. Por ende, tiene una rama para su uso privado y por lo tanto no es necesario que constantemente sincronice los cambios con los otros desarrolladores del proyecto, además la integración es controlada y centralizada por una sola persona del proyecto, el cual realiza las mezclas (*merges)* y resuelve los conflictos que pueden surgir.

Para la incorporación de la línea de producto software a la gestión de configuración de la organización, se adoptó la base del mismo patrón existente, el desarrollo por ramas

(37), sin embargo, se implementó una variante al esquema actual, debido a que debemos identificar claramente la base común de código y las variantes para cada cliente.

El esquema consiste en que la rama principal llamada *HEAD* es la que representa la base común, es decir, el conjunto de activos de la línea de productos software. Las ramas representan las variantes para cada cliente. De esta manera, se trabaja en forma paralela a partir de una versión estable de la línea de productos software. Este esquema se visualiza en la Figura 3-63.

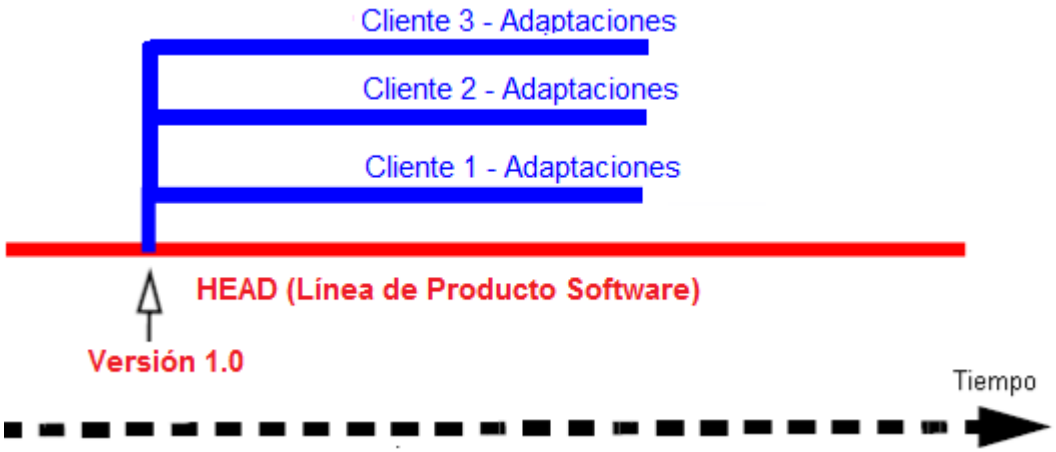


Figura 3-63: Estructura de la línea de productos software con desarrollo paralelo.

Como la organización utiliza la herramienta de control de versiones CVSNT (35), el cual soporta la herencia de ramas, las líneas de los productos concretos para cada cliente, heredarán todos sus contenidos de la rama *HEAD*. Las ramas serán creadas a partir de la rama *HEAD* pero con un esquema ligero (copiar sólo los archivos que se modificaran), al contrario como lo realiza actualmente la organización, donde se copia todo el contenido de la rama HEAD. Si la rama HEAD contenía 600 archivos fuentes, se copian los 600 archivos a la rama del desarrollador, esto produce que se pierda la trazabilidad de componentes y se tienen problemas a la hora de realizar mezclas con la rama *HEAD*. Por lo tanto, la rama del cliente sólo se copiaran aquellos archivos fuentes que sean modificados y no una copia de la rama *HEAD.*

Si la rama *HEAD* evoluciona durante el tiempo incorporando nuevos requerimientos o detectando algún defecto en la línea de productos software, dichos cambios serán propagados a todas las ramas de los clientes. Para aquellas fuentes que tienen conflictos se ejecutará el proceso de mezcla (*merge*) desde la etiqueta (*tag)* que corresponde a la marca de la última versión estable contra la rama del cliente. Este esquema se visualiza en la Figura 3-64. Sin embargo, pueden existir archivos fuentes que son incompatibles con las adaptaciones de algún cliente, por ende no se sería factible realizar la mezcla y la evolución o corrección de algún defecto debería realizarse en forma manual en la rama del cliente.

Como en el entorno de desarrollo es Microsoft Visual Studio 2008, no es factible integrar CVS en forma nativa, se utiliza el cliente TortoiseCVS (38) que permite realizar todas las operaciones de CVS a través del explorador de Windows.

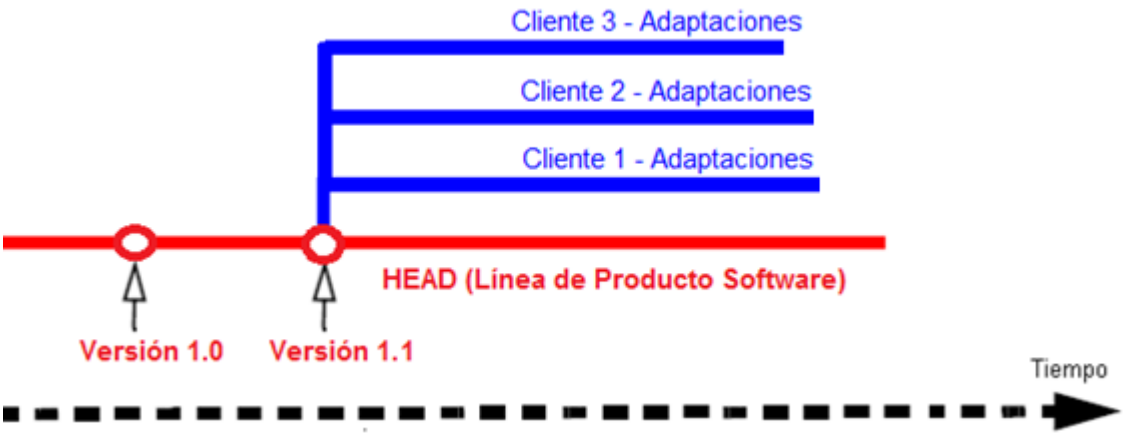


Figura 3-64: Sincronización de la rama *HEAD* con las ramas de los clientes.

3.8.1 Correcciones en las ramas de los clientes

Cuando se detectan *bugs* en las ramas de los clientes, se identifican si el o los fuentes donde se detectó pertenecen a la rama común (HEAD) de la línea de productos, de ser así, entonces se realiza una mezcla desde la rama del cliente a la rama común (HEAD), y luego se propagan todos los cambios a las ramas de los otros clientes. En caso que el o los fuentes no pertenezcan a la rama común (HEAD), los cambios no serán mezclados y quedaran sólo en la rama del cliente. Este esquema se visualiza en la Figura 3-65.

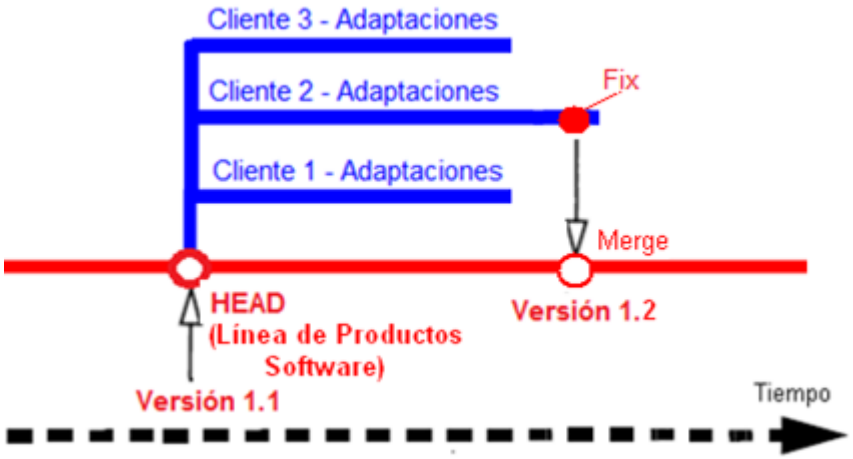


Figura 3-65: Esquema de correcciones en las ramas de los clientes.

3.8.2 Resumen de gestión de configuración

Finalmente, el uso del patrón desarrollo por ramas para la gestión de configuración, es el que mejor se adopta a las necesidades de la organización y los cambios que puede sufrir la línea de productos software en conjunto con la herramienta CVSNT (35) y TortoiseCVS (38), permitiendo resguardar el conjunto de activos que se produjeron y los

productos concretos para cada cliente. Sin embargo, este esquema claramente le falta madurez y seguramente en un tiempo más se deberá revisar para cubrir las necesidades que a esa fecha surjan debido a que sólo fue factible validarlo con la incorporación de un solo cliente con sus adaptaciones.

Capítulo IV

4 Evaluación de la línea de productos software

En este capítulo se presenta la evaluación de la línea de productos de tramitación de causas judiciales a través de dos conceptos; la implementación de la línea de productos y un comparativo con los cuatro sistemas de tramitación de causa en conjunto con creación de un producto desarrollado por la organización.

4.1 Implementación de la línea de productos software

Actualmente, la organización diseñó e implementó un producto que concentra todo el conocimiento adquirido durante estos últimos años en el dominio judicial. El paradigma de desarrollo fue el tradicional, guiado por su proceso de desarrollo de software llamado APF, el cual reúne las prácticas de CMMI, norma ISO 9001 y el Proceso Unificado. Sin embargo, una vez comercializado el producto, surgieron una serie de inconvenientes que el proceso APF no logró resolver. La propuesta del proyecto de tesis busca resolver los inconvenientes anteriormente mencionados, a través del diseño e implementación de una línea de productos software en el dominio de tramitación de causas judiciales. El desarrollo de una línea de productos supone un reto importante, teniendo en consideración que la organización es una PYME. Sin embargo, la implementación de la línea de productos utiliza las mismas herramientas y técnicas que se usan en el desarrollo de software tradicional, reduciendo la curva de aprendizaje en los equipos y minimizando los aspectos más complejos que representan la adopción de este nuevo paradigma.

A través del mecanismo de combinación de paquetes ("*package merge*") y la utilización de clases parciales, propuesto por Laguna y Otros (2), tratado en el punto 2.5 del capítulo II, se implementó un conjunto de características con una funcionalidad mínima, pero que permite una tramitación básica de causa. Además se han completado una serie de paquetes opcionales que formar un conjunto considerable de productos potenciales, aunque al seguir trabajando en la línea de productos se deberá aumentar más la variabilidad, permitiendo de paso convertir en una familia de productos de complejidad significativa en el dominio judicial. El mecanismo de combinación de paquetes demuestra la viabilidad de gestionar la variabilidad de la línea de productos utilizando únicamente modelos UML, el lenguaje de programación estándar como C# y la utilización de un único entorno de desarrollo como Microsoft Visual Studio. La implementación del mecanismo, logró simplificar la adopción de este paradigma de desarrollo, evitando la necesidad de herramientas o técnicas específicas para el tratamiento de la variabilidad y configuración de las líneas de productos.

Finalmente para la creación de productos específicos se utilizó el *plug-in* FMT (5), que permite la configuración de las características deseadas para derivar en un producto concreto. De esta manera el equipo de desarrollo sólo se concentra en las adaptaciones que solicitan los clientes y de paso se automatiza este proceso en un solo

entorno de desarrollo. Una vez implementada la línea de productos de tramitación de causas judiciales se obtiene como resultado la solución a los inconvenientes que la organización tiene actualmente, tales como:

- **Gestión de versiones**; al implementar el patrón de desarrollo paralelo, que consiste en que la rama principal llamada *HEAD* es la que representa la base común y las otras ramas representan las variantes para cada cliente, se trabaja en forma paralela a partir de una versión estable de la línea de productos. Con este esquema se evita la pérdida de código, se identifica claramente cuál fue el punto de bifurcación de un producto específico, se sabe cuáles son las adaptaciones de un producto para un cliente en relación a la línea de producto y por último los accesos a la rama principal son controlados para proteger la propiedad intelectual del código fuente o que los equipos puedan mal utilizar dichas fuentes.
- **Evolución del producto**; al implementar el mecanismo de combinación de paquetes ("*package merge*") permitió localizar en un solo punto del modelo de diseño todas las variaciones que origina cada característica opcional, de forma que se mantenga una correspondencia uno a uno y se facilite la gestión de la trazabilidad. Por otro lado, por cada característica opcional significa la creación de un paquete correspondiente, el cual contiene todo lo relacionado a la característica opcional, favoreciendo la modularidad y el nulo impacto en las otras características ya desarrolladas. Esto se logró corroborar en la implementación de la tercera iteración con el desarrollo de la característica resolver, donde a futuro es factible incluir más variabilidad sin afectar las características desarrolladas, permitiendo gestionar en forma eficiente la variabilidad de esta característica.
- **Personalización del producto para cada cliente**; al utilizar el *plug-in* FMT (5) se automatiza la capacidad de crear eficientemente múltiples variaciones de productos de acuerdo con las características solicitadas por el cliente, ya que permite configurar los paquetes implementados en todas las iteraciones que representan las características de la línea de productos, para derivar en un producto concreto y luego concentrarse en las adaptaciones. Sin este mecanismo, se requiere una configuración manual para configurar las clases que deben o no estar incluidas en una solución web final.
- **Código fuente poco robusto**; al utilizar el concepto de clases parciales en C# y el *framework* Castle ActiveRecord sobre NHibernate se logra incrementalmente extender la funcionalidad sin romper la modularidad de las características ya desarrolladas en la mayoría de los casos.
- **Costos de la mantención**; Los principales costos de mantención se derivan en la corrección de los defectos detectados en el producto. Al utilizar el enfoque de líneas de productos la tasa de defectos se reduce debido a la reutilización de la parte común, ya que la continua utilización de estos elementos hace que finalmente estén muy depurados, por ende los costos de mantención se reducen

en los productos que se obtengan posteriormente a través de la línea de productos.

Al resolver la organización los inconvenientes presentados, queda en una posición clara de optimizar los resultados obtenidos a la fecha con el producto desarrollado e incrementarlos en forma significativa al adoptar este nuevo paradigma de líneas de productos.

4.1.1 Evaluación de clases parciales de C# para la línea de productos software

Al utilizar el mecanismo de clases parciales de C# para la implementación de línea de productos se detectaron un conjunto de ventajas y desventajas que a continuación se describen:

- Ventajas:
  - Son útiles para la separación de código y el desarrollo modular de las funcionalidades. Por lo tanto, al ser aplicado a características de líneas de productos, se ha empleado más como técnica que aumenta el grado de modularidad del software, logrando construir características separadas pero combinables con otras.
  - Permiten añadir nuevos métodos y atributos a las clases parciales existentes. De esta forma, se logra incrementalmente incorporar más variabilidad a la línea de productos sin afectar al resto de las características desarrolladas.
  - No existe dependencia de tipo entre clases, ya que no se usa herencia como mecanismo de extensión, simplemente se combinan los diferentes extractos de códigos que están distribuidos en las clases parciales en una sola clase en tiempo de ejecución. Por lo tanto no se cambia el tipo de clase, sino que sigue siendo el mismo para las diferentes características.
  - Como C# no es un lenguaje orientado a característica, carece de mecanismo de encapsulación y modularidad de clases que están relacionadas a una característica, además para derivar en un producto concreto se debe recurrir a mecanismo de inclusión o exclusión de las clases parciales desde el archivo de compilación del proyecto. Sin embargo, esto por si sólo pareciera ser una desventaja del lenguaje, pero con la incorporación del *plug-in* FMT (5) se convierte una ventaja, ya que todo este proceso manual se automatiza, permitiendo gestionar la configuración de los productos concretos manteniendo una coherencia de las configuraciones producidas, seleccionado o deseleccionado las características solicitadas por el cliente.
- Desventajas:
  - Las clases parciales carecen de mecanismos para agrupar y encapsular las características. C# permite a través de los espacios de nombres (*namespace)* almacenar un conjunto de clases que están relacionadas

para una funcionalidad particular. Sin embargo, al utilizar clases parciales y estar situadas en diferentes espacios de nombre el compilador no combina las clases. Por lo tanto, todas las clases parciales deben estar situadas en el mismo espacio de nombre, impidiendo contar con un mecanismo del lenguaje que permita agrupar y encapsular las clases.

- No es posible extender los métodos ya existentes, es decir, la implementación de un método no puede estar dividida en dos clases parciales. Esto representa una desventaja importante respecto a la extensibilidad, ya que necesariamente se debe incorporar código fuente en aquellas características donde se implementó inicialmente el método, dando como resultado tener mezclado el código de diferentes características y rompiendo la modularidad.

La evaluación final de las clases parciales como mecanismo de implementación de la línea de productos por sí sola no resulta positiva, ya que no es un mecanismo suficiente para la implementación de características. Como punto destacado, si permite la extensibilidad mediante la incorporación de nuevos atributos y métodos sin recurrir al mecanismo de herencia, lo cual evita posibles conflictos con el sistema de tipo o conversiones innecesarias.

El mecanismo de clases parciales debe ser combinado con otras técnicas de modelado y generación de composición de características, tal como se describe en el proyecto de tesis al utilizar la combinación de paquetes ("*package merge*") y el *plug-in* FMT (5) propuesto por Laguna y Otros (2).

4.2 Comparativa

En el contexto presentado en el capítulo III, en el cual se enmarca el proyecto de tesis, la organización desarrolló los principales sistemas informáticos de tramitación de causas judiciales para el Poder Judicial de Chile. Dichos proyectos fueron desarrollados en forma independientes, aunque todos comparten un conjunto común de funcionalidades, como se especificó en el punto 3.2 del capítulo III. Al desarrollar e implementar la línea de productos se encontró que es una aproximación realista y que los productos que son creados a partir de ella, cubren un conjunto importante de necesidades que existen en las organizaciones que tramitan causas judiciales. Sin duda la variabilidad que cubre la línea de productos en relación al dominio judicial es pequeña y sencilla, ya que fue sólo desarrollada por una persona y en el contexto de un proyecto de tesis. Sin embargo, el número de productos que se pueden crear a partir de ella son 18, un número considerable y prometedor si a futuro se sigue trabajando en aumentar la variabilidad.

La comparación de los sistemas de tramitación de causas en conjunto con el producto propio que desarrolló la organización se basa en aspectos de costos y variabilidad que a continuación se presentan.

4.2.1 Costos

Para realizar una comparación entre los sistemas ya desarrollados por la organización y la línea de productos, se toma como supuestos la cantidad de personas que trabajaron en la construcción de cada sistema y los meses de duración que tuvieron los proyectos. Estos datos se multiplicaron por 50 que representan la cantidad de UF (unidades de fomento) mensual que en promedio puede costar a la organización una persona del equipo, estos valores no necesariamente representan la realidad actual, pero sirven como base comparativa. Sin duda estos montos varían de acuerdo con los tipos de roles que necesita cada proyecto, pero se realiza para simplificar la comparación con las líneas de productos y que representa generalmente, en el presupuesto de los proyectos de software, el ítem con los costos más altos.

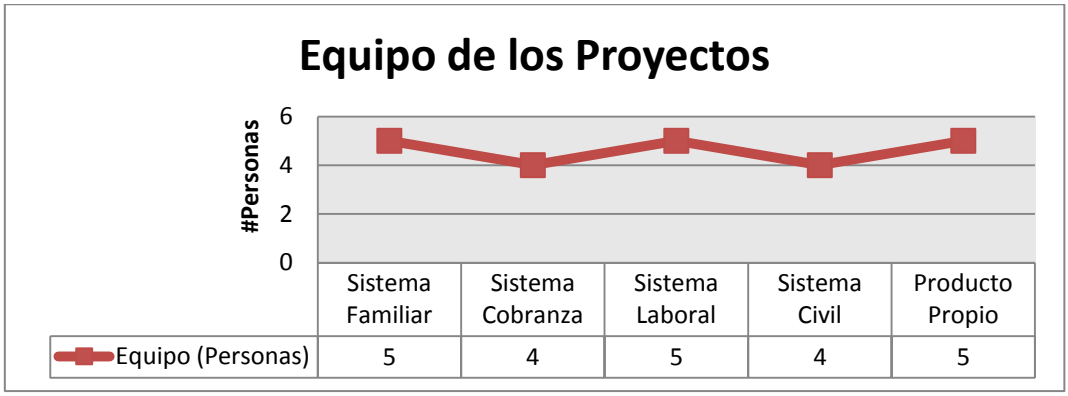


Figura 4-1: Cantidad de personas que participaron en los proyectos desarrollados por la organización.

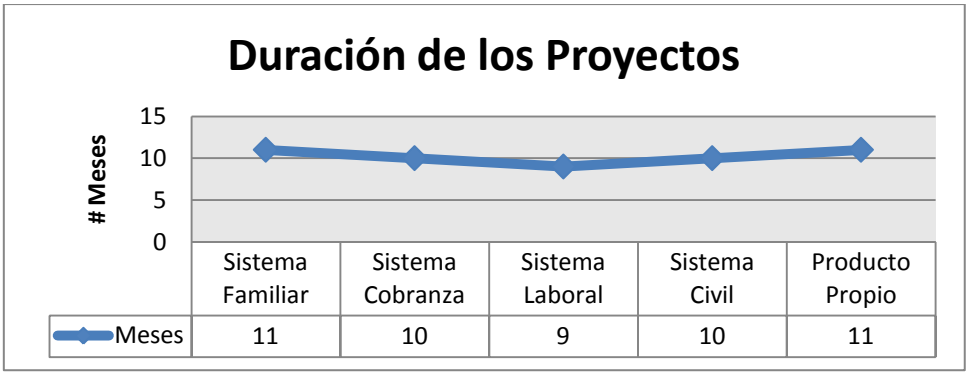


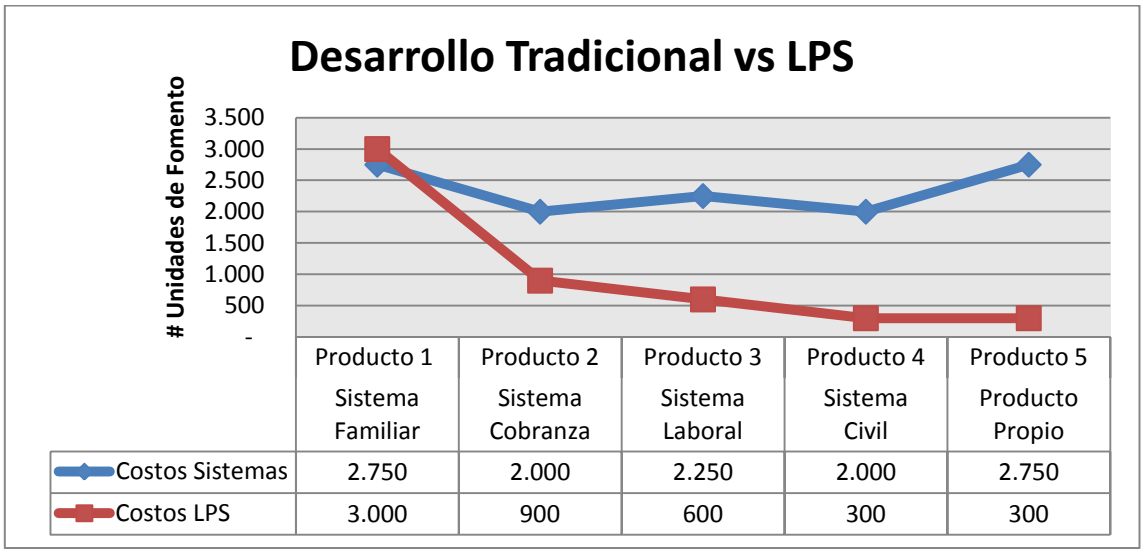
Figura 4-2: Cantidad de meses que tuvieron como duración los proyectos desarrollados por la organización.

Para la línea de productos, se tomó como base que se necesitarían 12 meses y 5 personas para implementar el total de características que puede soportar la línea de productos y de esa manera generar el primer producto concreto. Tomando los mismos supuestos de los sistemas ya desarrollados, la inversión inicial que debería adoptar la organización asciende al monto de 3.000 UF, como se visualiza en la Figura 4-3, si ahora se compara con el costo del primer sistema desarrollado por la organización, son levemente más elevados. Sin embargo, a partir de la creación del segundo producto la organización debería comenzar a rentabilizar la implementación de la línea de productos. Desde ese punto, la creación de nuevos productos sería menos costosa que

el enfoque actual de desarrollo, debido a que la cantidad de personas se reduce en un tercio y el esfuerzo a la mitad. Si ahora se compara con el tercer producto, la cantidad de personas se reduce a la mitad debido a que la mayoría de las funcionalidades están desarrolladas y las adaptaciones del Sistema Laboral se traducen en configuración de los procedimientos judiciales, por ende se visualiza un ahorro del 1.650 UF, acumulando hasta la liberación del quinto producto la suma total de 6.650 UF.

En el actual proceso de desarrollo (APF) no se obtienen ahorros importantes por ser sistemas ni productos similares, la estrategia de reutilizar componentes es oportunista y se basa en la técnica del *copy/paste*. El esfuerzo de desarrollar el Sistema Familiar es similar al esfuerzo que conlleva el Sistema Cobranza, cada sistema evolucionó en forma independiente y es gestionado en la actualidad por equipos independientes.

Claramente la organización, del punto de vista de costo, no se vió fuertemente impactada ya que todos estos sistemas fueron licitados con presupuesto propio e incluía los costos de mantención (evolutiva y correctiva) por el cliente. Por lo tanto, no visualizaba una motivación de negocio para cambiar dicha situación. Sin embargo, al desarrollar un producto propio el escenario cambió radicalmente, donde la misma estrategia no resulta beneficiosa para la organización. Al aplicar el paradigma de líneas de productos la organización se beneficia tanto en aspectos de ingeniería y negocio, logrando ser más competitiva y eficiente ya que los esfuerzos de las mantenciones en la línea de productos son capitalizados por todos los productos.



	Producto 1 Sistema Familiar	Producto 2 Sistema Cobranza	Producto 3 Sistema Laboral	Producto 4 Sistema Civil	Producto 5 Producto Propio
Costos Sistemas	2.750	2.000	2.250	2.000	2.750
Costos LPS	3.000	900	600	300	300

Figura 4-3: Desarrollo actual de la organización versus línea de productos.

Como resumen, se puede concluir que se cumple el principal objetivo de una línea de productos software al reducir los costos de desarrollo y mejorar la calidad de los productos concretos pertenecientes a la familia de productos ya que es más rentable dedicar más esfuerzo a la ingeniería del dominio que a la del producto conforme aumenta el número de productos, por ende si la organización, desde un principio hubiera adoptado el enfoque de líneas de productos, el ahorro en costo, tiempo y

esfuerzo, habría sido significativo dada la variabilidad que soporta en la actualidad la línea de productos desarrollada.

4.2.2 Variabilidad

La variabilidad es un concepto clave en las líneas de productos. No se trata de comprender cada sistema individual de forma completamente independiente, sino de considerar la línea de productos como un todo y especificar las variaciones de los distintos sistemas individuales (34). La variabilidad soportada por la línea de productos, representa la mayoría de las características comunes y opcionales que comparten todos los sistemas de tramitación de causas judiciales más el producto propio desarrollado por la organización. En la Tabla 5 se visualiza una matriz de características versus los sistemas y el producto desarrollado. Las líneas de color verde representan las características que fueron implementadas en el proyecto de tesis, mientras que las de color amarillo son las características en común que no fueron implementadas. Las filas que no se encuentran destacadas representan las características opcionales que tiene cada sistema.

Teniendo en consideración la variabilidad que soporta la línea de productos y realizando una comparación entre el sistema civil y el producto propio casi no existen adaptaciones para los clientes, ya que la mayoría de las características de ambos las soportarían la línea de productos. Estos sistemas en términos de negocio, se rigen bajo configuraciones procesales del código de procedimiento civil, por lo tanto si se hubieran producido a través de la línea de productos el esfuerzo, tiempo y costo habría sido marginal, aumentando considerablemente los márgenes de ganancia para la organización, de paso reduciendo el riesgo de entregar el sistema en las fechas pactadas con el cliente. En el caso del producto propio la duración del proyecto estaba considerada para 7 meses, terminando en 11 meses debido a las incorporaciones de nuevas funcionalidades que ya habían sido incluidas en otros sistemas. Este aspecto cobra especial relevancia, debido a que todos estos sistemas fueron adjudicados a través de licitaciones públicas donde se establece en la oferta realizada los plazos comprometidos y en caso de no cumplir, el cliente aplica multas en contra de la organización. Si bien la variabilidad que soporta la línea de productos permite la creación de los mismos sistemas y productos desarrollados por la organización, a futuro se debe trabajar en extender la variabilidad de las características requeridas para minimizar aún más el esfuerzo dedicado a las adaptaciones de los productos.

Característica	Sistema Familiar	Sistema Cobranza	Sistema Laboral	Sistema Civil	Producto Propio
Ingreso de Causa	X	X	X	X	X
Ingreso de Causa Internet	X		X		
Ingreso de Menores	X				
Ingreso Exhortos	X	X	X		
Ingreso Exhortos	X	X	X		

Otros Tribunales					
Exhortante		X			
Exhortado		X			
Ingreso Incompetencia	X	X	X		
Modificación de Causa	X	X	X	X	X
Consulta Ingreso	X	X	X	X	X
Ingreso Escritos	X	X	X	X	X
Tramitar Expediente	X	X	X	X	X
Escritos Pendientes	X	X	X	X	X
Trámites Pendientes	X	X	X	X	X
Actuaciones Pendientes	X	X			
Búsqueda de Plazos	X	X	X		X
Causas Sin Movimientos	X		X		
Notificaciones	X	X	X	X	X
Programar Audiencia	X		X		
Custodia	X	X	X	X	
Consulta de Causas	X	X	X	X	X
Consulta de Audiencia	X		X		
Diligencias	X	X	X		
Login	X	X	X	X	X

Tabla 5: Matriz de características versus los sistemas de tramitación de causas desarrollados por la organización.

Capítulo V

5 Conclusiones y trabajos futuros

En este capítulo se presenta un resumen de las conclusiones, objetivos alcanzados y trabajos futuros sobre el desarrollo de la línea productos en el dominio de tramitación de causas judiciales.

5.1 Conclusiones

El proyecto de tesis “línea de productos software de tramitación de causas judiciales” ha mostrado los aspectos más relevantes de las líneas de productos, características, actividades que lo componen, metodología y mecanismos para su implementación. La ventaja principal de este paradigma es la idea de planificar el desarrollo visualizando más allá de un único producto, pensando las variaciones que se puede dar entre los diferentes productos, teniendo como objetivo disminuir los tiempos de salida de los productos que conforman la línea y mejorar el proceso de producción de los mismos.

Las líneas de productos, al igual que cualquier otro paradigma del desarrollo de software, no es la solución para cualquier tipo de problema. La gestión del desarrollo se vuelve más compleja con las líneas de productos y está sujeta a mayores riesgos que las de los desarrollos realizados de forma individual, y por eso, antes de tomar la decisión de implantar una línea de productos, hay que evaluar desde el punto de vista técnico y económico si el esfuerzo de hacerlo merece la pena (39).

Otro punto importante en el contexto de la línea de productos es determinar la variabilidad que soportará, éste es un aspecto clave de este tipo de paradigma. A través del modelo de características permite modelar, analizar y configurar la variabilidad de la línea de productos. Todas estas tareas son complejas y deben ser automatizadas para minimizar el esfuerzo y errores en su ejecución por parte del equipo de trabajo.

La desventaja que se observó es el mayor esfuerzo requerido para el análisis, diseño e implementación de las características de la línea de productos. Esto significa que las organizaciones deben recurrir a una inversión mayor en tiempo, esfuerzo y costo que el enfoque tradicional del desarrollo de software, obteniendo los beneficios en el mediano y largo plazo. En consecuencia este paradigma podría estar menos propenso a ser utilizado por organizaciones pequeñas que tengan un presupuesto limitado o que los plazos de entrega sean muy acotados en sus proyectos. Sin embargo, para aquellas organizaciones que son especialistas sobre un dominio se traduce en una ventaja al reducir los tiempos de estudio del dominio para determinar la variabilidad, reutilizar componentes o código que previamente fueron desarrollos en otros sistemas y por último la experiencia necesaria para determinar claramente el alcance de las necesidades de los clientes.

La tesis planteada y que se ha pretendido demostrar en el desarrollo de este documento, validó que al aprovechar todo el conocimiento adquirido durante los últimos

años sobre el dominio de tramitación de causas judiciales por la organización, es factible transformarlo en una oportunidad de negocio real, mediante la creación de una línea de productos que permite a la organización contar con una variabilidad que al menos soporta las características de los sistemas que previamente fueron desarrollados. La variabilidad de los nuevos productos que son creados a partir de la línea es gestionada en forma eficiente a través del *plug-in* FMT (5). Si la organización hubiera adoptado este tipo de paradigma desde la creación del primer sistema de tramitación de causas judiciales, los ahorros de costos en el desarrollo y mejoras en la calidad de los productos habrían sido significativos, de paso obteniendo beneficios en aspectos de ingeniería, como la reducción en el esfuerzo requerido para el desarrollo y mantención de los productos y de negocio, como la reducción del tiempo en la entrega de nuevos productos. Sin embargo, el costo inicial del desarrollo de la línea de productos supone una inversión fuerte al principio para la organización, pero ya desde el segundo producto se logran maximizar el retorno de la inversión (ROI) gracias a los ahorros en los costos de desarrollo y mantención de los productos.

Respecto a las conclusiones de los mecanismos técnicos utilizados se ha demostrado la viabilidad de implementar la línea de productos por medio del mecanismo de combinación de paquetes de UML 2 y clases parciales de C#, propuesto por Laguna y Otros (2), permitiendo de paso simplificar la adopción de este paradigma de desarrollo, evitando la necesidad de herramientas, lenguajes o técnicas específicas que no fueran conocidas por la organización. Por otro lado, la combinación de paquetes permitió derivar la implementación de cada producto concreto a partir de la configuración de características deseadas por el cliente, con el *plug-in* FMT (5), automatizando todo este proceso en un único entorno de desarrollo como Microsoft Visual Studio. Los mecanismos implementados permitirán a la organización reutilizarlos a futuro en nuevos dominio, ya que contará con equipos capacitados y con la experiencia suficiente en las herramientas utilizadas, dejando solamente la curva de aprendizaje sobre el estudio del dominio. El uso de estas técnicas y herramientas habituales deja a disposición este paradigma de desarrollo al alcance de la mayoría de los programadores y analistas dentro de la organización como de otras.

Sin duda el desarrollo de una línea de productos no es una tarea fácil, ni tampoco es una tarea que puede ser desarrollada sólo por una persona, se deben integrar un equipo transversal a la organización (área comercial, de *marketing*, desarrollo, soporte, entre otras). Estos equipos tienen diferentes intereses y miran desde su perspectiva la variabilidad de la línea de productos. El intercambio de información entre estos equipos suponen un reto para la organización que deberá ser abordada a futuro e incorporada a sus procesos en la medida que aborde en nuevos dominios y mercados.

5.2 Objetivos alcanzados

El objetivo principal del proyecto de tesis se cumplió al desarrollar una línea de productos dentro del dominio de tramitación de causas judiciales, permitiendo a la

organización contar con una serie de componentes reutilizables y gestionar en forma eficiente la variabilidad de los nuevos productos que sean creados para diferentes clientes en un mismo entorno de desarrollo como Microsoft Visual Studio 2008. Al desarrollar la línea de productos se lograron los objetivos específicos propuestos:

- Se ha obtenido una comprensión y conocimiento del concepto de línea de productos software, se focalizó principalmente en el mecanismo de combinación de paquetes de UML 2, propuesto por Laguna y Otros (2).
- Se ha desarrollado una línea de productos para la tramitación de causas judiciales aplicada a sistemas Web.
- Se ha definido un modelo de características con el *plug-in Feature Modeling Tool* (5) en Microsoft Visual Studio 2008 para crear y configurar nuevos productos de tramitación de causas seleccionando las características deseadas por el cliente para posteriormente incorporar las adaptaciones solicitadas.
- Se ha dotado de una solución real y factible de implementar a la problemática actual que tienen la organización con la comercialización de su producto, en aspectos de ingeniería y negocios.
- Se ha conseguido un conocimiento suficiente para implementar en otro dominio el paradigma de líneas de productos en la organización.

Al cumplir los objetivos propuestos en el desarrollo de la línea de productos podemos concluir que los resultados esperados se obtienen al:

- Reducir el tiempo de entrega de nuevos productos para el sector judicial, ya que la reutilización no es oportunista sino planificada.
- Incrementar el número de productos que pueden ser desarrollados por la organización a partir de la línea de productos software. En la actualidad, la variabilidad soportada por la línea de productos asciende a 18 productos, número que se incrementa en función evolucione la variabilidad propuesta.
- Reducir el riesgo en la entrega de los productos. Al existir características comunes ya implementadas, los equipos sólo se concentran en las adaptaciones que solicitan los clientes y no en rehacer lo que previamente había realizado la organización en proyectos similares.
- Mejorar la rentabilidad de los proyectos. Al implementar la línea de productos se generar ahorros en los costos de desarrollar el producto y mejoras en la calidad. Por ende se necesitan menos personas para las mantenciones de los productos.

5.3 Trabajos futuros

Algunos de los temas que deben ser abordados a futuro para mejorar el desarrollo e implementación de la línea de productos tienen relación con lo siguiente:

- **Variabilidad en la capa de presentación:** El mecanismo de combinación de paquetes de UML 2 y la implementación de dichos paquetes con clases parciales de C# es un avance a la hora de desarrollar la línea de productos, pero dicho mecanismo no incorpora el soporte a la variabilidad en la capa de

presentación. La solución implementada que se utilizó para generar variabilidad en las páginas ASPX de .NET fue la combinación de controles *PlaceHolder,* controles de usuario y un archivo XML que permite determinar si se debe incluir o no el control de usuario en la característica principal. Este aspecto se debe trabajar a futuro para la generación automática de la interfaz de usuario en proyectos web utilizando definiciones parciales de interfaces, tales como XFORMS u otro mecanismo que permitan evitar dicha configuración en forma manual.

- **Variabilidad en la base de datos:** Para dar solución a la variabilidad en la capa de acceso a datos se utilizó el *framework* Castle ActiveRecord en conjunto con el mecanismo de clases parciales, lo que permitió que el mapeo entre los atributos de las clases y campos de la base de datos fuera más encapsulada, debido a si alguna característica opcional no era seleccionada al derivar en un producto concreto, la clase no mapeaba los campos que pertenecían a la característica. Sin embargo, para la creación de la base de datos obligatoriamente se creó incorporando todas las tablas y campos del modelo de datos. Lo que se traduce en que pueden existir productos con campos o inclusive tablas que nunca serán utilizadas por la aplicación. Claramente éste es un tema pendiente que se debería abordar en trabajos futuros de la evolución de la línea de productos.
- **Arquitectura orientada a servicios:** Algunos clientes solicitan que sus sistemas sean creados bajo una arquitectura orientada a servicios para la integración con sus sistemas *legacy o word class.* Actualmente la arquitectura soportada por la línea de productos está basada en una orienta al dominio, desde el punto de vista lógico. Sin embargo, debido a la restricción del *plug-in* FMT no es factible generar una arquitectura orientada a servicios, convirtiendo la capa de dominio en un proyecto WCF de .NET. Un punto a trabajar sería evolucionar el *plug-in* FMT que permita soportar variabilidad a nivel de una solución completa de Microsoft Visual Studio y no como es actualmente a un solo proyecto Web.
- **Pruebas en la Línea de Productos**: En el desarrollo del proyecto de tesis no se hace mención a las pruebas de la línea de productos, este punto debe ser trabajado a futuro para determinar claramente qué parte del software debe ser probada en cada uno de los dos procesos, la ingeniería del dominio o la ingeniería del producto, ya que la prueba exhaustiva de las partes comunes es imposible debido a la variabilidad que soporta actualmente la línea de productos. Por último, se debe evaluar la posibilidad de reutilizar las pruebas que se realizan a nivel del dominio y a nivel del producto.

6 Glosario Judicial

6.1 Tribunales

Son aquellos a quienes corresponde el conocimiento de todos los asuntos judiciales que se promuevan en el orden temporal dentro del territorio de la República, cualquiera que sea su naturaleza o la calidad de las personas que en ellos intervengan, salvo las excepciones legales y constitucionales.

6.2 Instancia

Cada uno de los grados jurisdiccionales de un proceso en el cual el tribunal conoce los hechos y el derecho.

6.3 Competencia

La competencia es la facultad que tiene cada juez o tribunal para conocer de los negocios que la ley ha colocado dentro de la esfera de sus atribuciones.

6.4 Los abogados

Los abogados son personas revestidas por la autoridad competente de la facultad de defender ante los Tribunales de Justicia los derechos de las partes litigantes.

6.5 Proceso judicial

Un conjunto sucesivo de actos de las partes de un conflicto de relevancia jurídica, de ciertos terceros y del tribunal, desarrollados en forma progresiva ante este último, de acuerdo con las normas de procedimiento que la ley en cada caso señala, a través del cual el Juez desempeña la función jurisdiccional que le ha encomendado el Estado, cuyo ejercicio normalmente concluye con la dictación de la sentencia definitiva, en la cual éste consigna la solución del asunto controvertido.

6.6 Las partes

Son los sujetos entre los cuales se produce la contienda o conflicto de relevancia jurídica y son conocidas con el nombre de demandante y demandado en el juicio civil y como querellante o querellado o procesado en las causas criminales.

6.7 Actuaciones judiciales

Son actos jurídicos procesales realizados por el tribunal o ante él por las partes o ciertos terceros, de los cuales debe dejarse testimonio en el expediente.

6.8 Los exhortos

Son comunicaciones escritas por las cuales un tribunal que está conociendo de un asunto encarga a otro la realización de determinadas actuaciones judiciales que deben practicarse dentro del territorio de este último, como por ejemplo tomar declaraciones a testigos que resida en el territorio del tribunal exhortado.

6.9 Las notificaciones

Son aquellos actos jurídicos procesales que tienen por objeto poner en conocimiento de las partes de un proceso o de terceros una determinada resolución judicial u otra actuación procesal precisa.

6.10 Resoluciones judiciales

Son aquellos actos jurídicos procesales emanados del tribunal, por medio de los cuales éste da curso progresivo a los autos y resuelve tanto el asunto controvertido en el proceso como las cuestiones accesorias que se promuevan durante la tramitación.

6.11 Demanda

La demanda es el acto procesal de actor, en virtud del cual éste ejercita la acción, sometiendo al conocimiento del tribunal la pretensión de que se le reconozca algún derecho que le ha sido desconocido o menoscabado.

6.12 Recurso procesal

Medio que la ley concede a la parte que se cree perjudicada por una resolución judicial para obtener que ella sea modificada o dejada sin efecto.

6.13 Cuaderno

Conjunto de trámites que constituye el procedimiento mismo que se someterá a la decisión del tribunal.

6.14 Plazos

Desde el punto de vista procesal son los espacios de tiempo fijados por la ley, el tribunal o las partes, para ejercitar una determinada facultad procesal o realizar un acto jurídico procesal. Desde este punto de vista los plazos son hechos jurídicos procesales.

6.15 Escritos

El escrito es el acto solemne que contiene las solicitudes que presentan las partes al tribunal y que debe reunir los requisitos exigidos por la ley.

6.16 Trámite

Son aquellos actos jurídicos procesales emanados del tribunal, por medio de los cuales éste da curso progresivo a los autos y resuelve tanto el asunto controvertido en el proceso como las cuestiones accesorias que se promuevan durante la tramitación.

6.17 Jueces

Es la autoridad pública que sirve en un tribunal de justicia y que se encuentra investido de la potestad jurisdiccional para aplicar la ley y las normas jurídicas.

6.18 Hitos

Son un nemotécnico definido para representar las resoluciones o actuaciones dictadas por un juez o funcionario de la organización que representan hitos dentro de distintas etapas del procedimiento judicial.

7 Bibliografía y Referencias

1. **Stephen R. Palmer, John M. Felsing.** *Desarrollo Dirigido por Características: A practical guide to feature-driven development.*

2. **González-Baixauli, Miguel A. Laguna y Bruno.** Variabilidad, Trazabilidad y Líneas de Productos: una Propuesta basada en UML y Clases Parciales. [En línea] [Citado el: 02 de Noviembre de 2012.] http://www.giro.infor.uva.es/Publications/2007/LG07/merge-jisbd07.pdf.

3. *Object-oriented programming with flavors, Conference proceedings on Object-oriented programming systems, languages and applications.* **Moon, David A.** 1986.

4. Eclipse Process Framework (EPF). [En línea] http://www.eclipse.org/epf/.

5. Feature Modeling Tool. [En línea] http://giro.infor.uva.es/FeatureTool.html.

6. **Fowler, Martin.** DomainSpecificLanguage. [En línea] [Citado el: 02 de Noviembre de 2012.] http://martinfowler.com/bliki/DomainSpecificLanguage.html.

7. **Rahien, Ayende.** *DSLs in Boo: Domain Specific Languages in .NET.* s.l. : Manning.

8. **Amodeo, Enrique.** ¿Qué son los DSL (Domain Specific Languages)? [En línea] [Citado el: 02 de Noviembre de 2012.] http://eamodeorubio.wordpress.com/2010/09/13/%C2%BFque-son-los-dsl-domain-specific-languages/.

9. **MARJAN MERNIK, JAN HEERING AND ANTHONY M. SLOANE.** When and How to Develop Domain-Specific Languages. [En línea] [Citado el: 02 de Noviembre de 2012.] http://repository.cwi.nl/search/searchrepository.php?auteurin=1078.

10. **Grupo GIRO. Departamento de Informática, Universidad de Valladolid.** Un Lenguaje Específico de Dominio para Líneas de Productos Software. [En línea] [Citado el: 02 de Noviembre de 2012.] http://www.giro.infor.uva.es/Publications/2009/FLRS09/demo%20FMT.pdf.

11. **Conference, Software Product Line.** Product Line Hall of Fame. [En línea] [Citado el: 02 de Noviembre de 2012.] http://www.splc.net/fame.html.

12. **Institute, Software Engineering.** Software Product Line Conference. [En línea] [Citado el: 02 de Noviembre de 2012.] http://www.splc.net/index.html.

13. **Bastarrica, María Cecilia.** Desarrollo de Líneas de Productos de Software. [En línea] http://www.eici.ucm.cl/Academicos/R_Villarroel/descargas/ing_sw_1/DesarrolloLineasProductosSoftware.pdf.

14. **Mario G. Piattini Velthuis, Javier Garzás Parra.** *Fábricas de software: Experiencias, tecnologías y organización.* s.l. : 2 Edición Actualizada.

15. **Charles W. Krueger, PhD, CEO.** BigLever Software. [En línea] http://www.softwareproductlines.com/benefits/benefits.html.

16. **Program, Software Engineering Process Management.** CMMI® para Desarrollo, Versión 1.3. [En línea] [Citado el: 02 de Noviembre de 2012.] http://www.sei.cmu.edu/library/assets/whitepapers/Spanish%20Technical%20Report%20CMMI%20V%201%203.pdf.

17. Agile Software Development. [En línea] http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software.

18. **Kang, K. C., S. Cohen, J. Hess, W. Nowak, and S. Peterson.** *Feature-Oriented Domain Análisis (FODA) Feasibility Study.* s.l. : Software Engineering Institute (Carnegie Mellon).

19. **Microsoft, Centro de Estudios de Justicia de las Américas (CEJA) y.** Centro de Estudios de Justicia de las Américas. [En línea] http://www.cejamericas.org/portal/index.php/es/gestion-e-informacion/justicia-y-tecnologia.

20. **Larman, Craig.** *UMLY PATRONES.* s.l. : Pearson Prentice Hall, 2003.

21. DSL Tools. [En línea] http://msdn.microsoft.com/en-us/vstudio/ff637759.

22. **Wikipedia.** UML: Lenguaje Unificado de Modelado. [En línea] [Citado el: 02 de Noviembre de 2012.] http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado.

23. **Booch, G., Rumbaugh, J, and Jacobson, I.** *The Unified Modeling Language User Guide.* s.l. : Addison Wesley, 1999.

24. **César de la Torre Llorente, Unai Zorrilla Castro, Miguel Angel Ramos y Javier Calvarro.** *Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0.* s.l. : Microsoft.

25. **Microsoft.** ASP.NET Ajax : Enhanced Interactivity and Responsiveness. [En línea] [Citado el: 02 de Noviembre de 2012.] http://www.asp.net/ajax.

26. **Wikipedia.** ORM. [En línea] [Citado el: 02 de Noviembre de 2012.] http://es.wikipedia.org/wiki/ORM.

27. —. Hibernate. [En línea] [Citado el: 02 de Noviembre de 2012.] http://es.wikipedia.org/wiki/Hibernate.

28. —. NHibernate. [En línea] [Citado el: 02 de Noviembre de 2012.] http://es.wikipedia.org/wiki/NHibernate.

29. **Klaus Pohl, Günter Böckle y Frank van der Linden.** *Software Product Line Engineering, Foundations, Principles, and Techniques.* s.l. : Springer.

30. **Laguna, Miguel A.** LACCEI. [En línea] www.laccei.org/LACCEI2009-Venezuela/p155.pdf.

31. **Microsoft.** MSDN. [En línea] [Citado el: 19 de Noviembre de 2012.] http://msdn.microsoft.com/en-US/library/dfb3cx8s.aspx.

32. NHibernate Forge. [En línea] [Citado el: 17 de Noviembre de 2012.] http://nhforge.org/.

33. Castle Project. [En línea] [Citado el: 17 de Noviembre de 2012.] http://docs.castleproject.org/Active%20Record.MainPage.ashx.

34. **Mario G. Piattini Velthuis, Javier Garzás Parra.** *Fábricas de software, Capítulo 8 Gestión de la Configuración Software: Experiencias, tecnologías y organización 2 Edición Actualizada.* s.l. : RA-MA.

35. **Wikipedia.** CVSNT. [En línea] [Citado el: 02 de Noviembre de 2012.] http://en.wikipedia.org/wiki/CVSNT.

36. **Project.net.** Project.net. [En línea] [Citado el: 02 de Noviembre de 2012.] http://www.project.net.

37. **Appleton, Steve Berczuk and Brad.** *Software Configuration Management Patterns: Effective Teamwork, Practical Integration.* s.l. : Addison-Wesley, Boston, MA.

38. **TortoiseCVS.** TortoiseCVS. [En línea] [Citado el: 02 de Noviembre de 2012.] http://www.tortoisecvs.org/.

39. *The Economic Impact of Product Line Adoption and Evolution.* **Schmid, K., Verlage, M.** July/August 2002 pp. 50-57, s.l. : IEEE Software .

40. **Svahnberg, M., van Gurp, J., and Bosch, J.** *On the Notion of Variability in Software Product Lines.* 2001.

41. **Group, Object Management.** *OMG Unified Modeling Language Specification.* 2001.

42. **Library, Rational Process.** http://www-01.ibm.com/software/awdtools/rmc/library/. [En línea] Octubre de 2012.

43. **Clements, P. & Northrop, L.** *Software Product Lines: Practices and Patterns.* s.l. : Addison-Wesley, 2002.

44. **Miguel A. Laguna, Bruno González-Baixauli, José M. Marqués, Rubén Fernández.** Feature Patterns and Multi-Paradigm Variability. [En línea] [Citado el: 02 de Noviembre de 2012.] http://giro.infor.uva.es/Publications/2008/LGM08/TR-featureP.pdf.

45. **Geoffrey Sparks, Sparx Systems, Australia.** Una Introducción al UML El Modelo de Casos de Uso. [En línea] [Citado el: 02 de Noviembre de 2012.] http://www.exa.unicen.edu.ar/catedras/modysim/teoria/casos_de_uso_a.pdf.

46. **G. Booch, J. Rumbaugh, and I. Jacobson.** *The Unified Modelling Language User Guide.* s.l. : Addison Wesley, 1999.

Anexo A: Herramientas

A continuación se presenta un breve resumen sobre las herramientas que fueron utilizadas durante la implementación del proyecto de tesis.

- **Eclipse Process Framework**

  El Eclipse Process Framework es una herramienta de desarrollo de procesos open source. Con ella es posible crear procesos consistentes, publicarlos y visualizarlos en un sitio web. Eclipse Process Framework facilita el desarrollo de los procesos porque cuenta con un editor potente de procesos que ayuda a crear procesos consistentes y provee mayor facilidad para su actualización.

  El resultado de la creación de procesos con EPF no es un documento monótono, por el contrario, es un sitio web que permite un despliegue rápido de los procesos y una navegación ágil para los usuarios de los procesos.

- **Feature Modeling Tool**

  Feature Modeling Tool (FMT) (5) es una herramienta que permite modelar gráficamente y configurar modelos de características (features) dentro del IDE Microsoft Visual Studio. Se añade como plugin y se puede comenzar a utilizar. Creado por el grupo GIRO, grupo de investigación y desarrollo sobre reutilización sistemática en el desarrollo de sistemas software de la Universidad de Valladolid.

- **DSL Tools**

  DSL Tools es una herramienta que se integra dentro del Visual Studio (21), y cuyo propósito es el de definir lenguajes específicos de dominio totalmente personalizados a las necesidades del usuario para poder construir una línea de producción de sistemas. Al mismo tiempo, DSL Tools permite asociar una metáfora gráfica a todos los artefactos que definen el lenguaje para disponer de un editor gráfico y así poder construir modelos visuales en base al lenguaje específico de dominio. Además, DSL Tools dispone de una utilidad para la generación de ficheros de texto, de forma que a partir de cualquier modelo especificado es posible emitir código fuente.

- **Microsoft Visual Studio 2008**

  Microsoft Visual Studio 2008 es un conjunto global de herramientas para crear proyectos de desarrollo pensados para Web (ASP.NET AJAX incluido), Windows Vista, Windows Server 2008, 2007 Microsoft Office System, SQL Server 2008 y dispositivos de Windows Mobile. Ofrece herramientas de desarrollo avanzadas, funciones de debugging, funciones para bases de datos que permiten crear rápidamente aplicaciones futuras para distintas plataformas.