ESTUDIO DE LOS TEMPLATES DE UML 2.0 PARA EL MANEJO DE ASPECTOS Y SU IMPLEMENTACIÓN SOBRE LA HERRAMIENTA CASE AR2CA

DIANA CAROLINA NASPIRAN BENAVIDES

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2008

ESTUDIO DE LOS TEMPLATES DE UML 2.0 PARA EL MANEJO DE ASPECTOS Y SU IMPLEMENTACIÓN SOBRE LA HERRAMIENTA CASE AR2CA

DIANA CAROLINA NASPIRAN BENAVIDES

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

DR. RAQUEL ANAYA DE PÁEZ
DIRECTOR

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2008

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

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

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

_____

Asesor: Dr. Raquel Anaya

_____

_____

_____

_____

_____

Jurado: Alexander Barón

_____

_____

_____

_____

_____

Jurado: Jaime Dávila

San Juan de Pasto, 25 de agosto de 2008

AGRADECIMIENTOS

A Dios por iluminar mi camino y darme las salidas apropiadas en los momentos difíciles de toda mi vida.

A la doctora Raquel Anaya mi asesora quien me dio la oportunidad de integrar el grupo de investigación de ingeniería de software de la Universidad EAFIT y me guió en este camino nuevo de conocimiento.

A todos los integrantes del grupo de investigación de ingeniería de software de la Universidad EAFIT en quienes siempre encontré una gran fuente de conocimiento y compañerismo, en especial Alexander y Gustavo quienes se convirtieron no solo en mis compañeros de trabajo sino también en amigos invaluables para toda la vida.

A mis Padres y a mis tíos Jesús y Mónica quienes me brindaron su apoyo en el momento de tomar decisiones y de quienes he aprendido que los sueños se cumplen con esfuerzo y dedicación.

A Julián quien en todo este tiempo siempre me ayudo a levantarme en los momentos en los que las fuerzas me abandonaban y quien ha celebrado mis logros y me ha ayudado a cargar mis dificultades.

RESUMEN

La adopción del proceso de desarrollo orientado a objetos por el entorno empresarial tuvo muchas dificultades y retrasos debido a la falta de estandarización y a la creación de modelos y metodologías de gran capacidad pero que no se ligaban a los lineamientos reales seguidos en los entornos empresariales. Esto ha servido como una buena experiencia para los grupos investigadores de nuevas tecnologías en ingeniería de software quienes en los últimos años han realizado grandes esfuerzos por ofrecer lineamientos y herramientas que permitan a los desarrolladores no solo mejorar su trabajo sino también actualizarse y poner en práctica nuevas tecnologías de forma más rápida haciendo que la evolución el desarrollo de software no se detenga y siga un rumbo especifico.

El proceso de desarrollo orientado por aspectos es una evolución en el desarrollo de software sobre la cual se están haciendo grandes esfuerzos para lograr que su adopción sea más rápida en el ambiente empresarial, para esto se están desarrollando estudios y creando metodologías de desarrollo teniendo en cuenta la de las empresas. Este es el caso del proyecto MEDUSA “Marco Metodológico Para El Desarrollo Orientado a Aspectos” el cual en el momento está siendo presentado y discutido por el entorno empresarial. Dentro del objetivo de ofrecer una más rápida adopción de la metodología de desarrollo por aspectos por parte de los desarrolladores, existe la necesidad de establecer modificaciones sobre las herramientas CASE orientadas al desarrollo de software para que ofrezcan un apoyo en las etapas de análisis y diseño. De esta idea surge este proyecto de tesis el cual retoma la herramienta CASE AR2CA, que inicialmente fue concebida para apoyar en el desarrollo de software orientado a objetos, y realizar sobre esta la implementación de los Templates como una de las alternativas más reconocidas para realizar un modelado parametrizado, siguiendo la propuesta de una aproximación de desarrollo orientada por aspectos denominada Theme/UML.

ABTSRACT

The adoption of the Object Oriented Software Development by the business environment had many difficulties and delays due to lack of standardization and the creation of models and methodologies of great ability but that is not linked to the lines followed in real business environments. This has served as a good experience for groups of researchers of new technologies in software engineering who in recent years have made great efforts to provide guidelines and tools to allow developers not only improve their work but also upgrade and implement new technologies faster making the evolution software development does not stop and follow a specific course.

The process of development-oriented aspects in an evolution in software development on which are making great efforts to ensure that its adoption will be faster in the business environment, are being developed for this study and creating development methodologies taking into account the view of enterprises. This is true of the project Medusa "methodological framework for the Development Oriented Aspects," which offers a methodology as a result of software development targeted areas at the moment is in business assessment. Within the objective of providing a more rapid adoption of the methodology development aspects by developers, there is the need for modifications on the tools CAS-oriented software development to provide support in the stages of analysis and design. This idea came this thesis project which takes up the case AR2CA tool, which was initially designed to support the development of object-oriented software, and perform on the implementation of this notation itself an approximation of development aspects heavily targeted used so-called Theme / UML.

TABLA DE CONTENIDO

LISTA DE TABLAS

Página

LISTA DE FIGURAS

GLOSARIO

ADVICE: Acciones que se ejecutan en cada JoinPoint incluido en un corte. Los advice definen el comportamiento que entrecruza toda la funcionalidad, están definidos en función de los pointcuts. La forma en que el código del advice es ejecutado depende del tipo del advice. (before, after, around). El cuerpo de un advice se parece mucho al cuerpo de un método, ya que encapsula la lógica que debe ser ejecutada cuando se alcanza cierto JoinPoint en la ejecución. Los puntos de corte junto con los advice, son las herramientas para implementar entrelazado dinámico. Los PointCuts identifican dónde y los advice lo completan indicando qué hacer.

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

ASPECTO: Un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos tienden a ser propiedades que afectan el rendimiento y la semántica de los componentes en forma sistemática (Ejemplo: sincronización, manejo de memoria, distribución, etc.) El aspecto es la unidad que encapsula JoinPoint, PointCuts, y advice, además de poder definir sus propios métodos y atributos. La diferencia con una clase es que un aspecto puede entrecruzar otras clases o aspectos, y que no son directamente instanciados con una expresión new, o procesos de clonado o serialización. Los aspectos pueden incluir la definición de un constructor pero dicha definición no debe contener argumentos y no debe señalar excepciones chequeadas.

CASE: Computer Aided Software Engineering

CONCERN: Son los diferentes temas o asuntos de los que es necesario ocuparse para resolver el problema. Por ejemplo una función específica que debe realizar una aplicación, pero también surgen otras como por ejemplo distribución, persistencia, replicación, sincronización, etc.

CROSSCUTING CONCERNS: Conceptos encapsulados que originalmente estaban repartidos por todo o parte de la aplicación.

CROSSCUTTING: Comportamiento trasversal.

JOINPOINTS: Brindan la interfaz entre aspectos y componentes; son lugares dentro del código donde es posible agregar el comportamiento adicional que destaca a la POA. Los joinPoints son puntos bien definidos en la ejecución de un

programa, entre ellos podemos citar llamadas a métodos, o a constructores, accesos, referencias e inicializaciones de atributos, etc.

OCL: Object Constraint Language (OCL) es un lenguaje de texto para especificar limitaciones, expresiones de navegación, expresiones booleanas, y otras consultas. No está destinado para escribir acciones o código ejecutable.

POA: "Programación orientada por aspectos" paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a la POA se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables.

POINTCUTS: Agrupan JoinPoints y permiten exponer el contexto en ejecución de dichos puntos. Una afirmación que es cumplida por un conjunto de puntos de unión. Los pointcuts principalmente son usados por los advice y pueden ser compuestos con operadores booleanos para crear otros cortes.

PUNTOS DE EXTENSIÓN: Puntos desde donde se extiende un comportamiento en un sistema.

RUP: El Proceso Unificado Racional (Rational Unified Process en inglés, habitualmente resumido como RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

SCATERING: Código diseminado.

TANGLING: Código Mesclado.

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

THEME: Constructores conceptuales y de diseño propios de la aproximación Theme/UML.

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

WEAVER: Encargado principal de tejer los diferentes mecanismos de abstracción y composición que aparecen tanto en los lenguajes de aspectos como en los lenguajes de componentes. Guiado por los puntos de enlace. (Se encargará de combinar los lenguajes)

WEAVING DINÁMICO: Los aspectos y componentes están relacionados débilmente. Los aspectos están unidos a los componentes en run time. Ineficiente (se debe usar código extra en runtime para acoplar y desacoplar los aspectos y clases).

WEAVING ESTÁTICO: Los aspectos están relacionados estrechamente con los componentes. El código del aspecto está mezclado dentro del código fuente antes de la compilación. Código altamente optimizado (la velocidad es comparable con código sin aspectos).

INTRODUCCIÓN

El desarrollo de software siempre se ha mantenido en constante avance, apuntando a brindar mayor escalabilidad, robustez y facilidad de mantenimiento, aparecen nuevas tecnologías que para ser adoptadas pasan por un estado de estudio y aceptación en el mundo de los desarrolladores de software. El desarrollo de software orientado a aspectos (sus siglas en ingles AOSD) es una de las respuestas que surgen con el fin de continuar con esta evolución y en este momento cuenta con un gran apoyo en grupos de investigación a nivel mundial.

La adopción de una notación adecuada para el análisis y diseño orientado a aspectos es fundamental para que esta nueva tecnología tome fuerza para el desarrollo de software a escala industrial. Teniendo en cuenta la adopción y difusión que ha tenido UML como estándar para modelado, resulta natural tratar de extender este lenguaje para que apoye a los aspectos en el desarrollo de software. Los Templates que se presentan como constructores auxiliares de la especificación de superestructura de UML 2.0 ofrecen una gran posibilidad de extensión que se pretende aprovechar y dirigir hacia el apoyo del modelado de aspectos y se plantean como objeto principal de esta investigación.

Este estudio permitirá extender la funcionalidad de la herramienta CASE AR2CA con la implementación del uso de los Templates, para que además del desarrollo de software orientado a objetos también soporte la orientación a aspectos y de esta forma los desarrolladores cuenten con una herramienta que soporte en la labor del modelado de software orientado a aspectos.

Este estudio hace parte del proyecto de investigación “Marco Metodológico Para El Desarrollo Orientado a Aspectos – MMEDUSA” desarrollado por el grupo de investigación en ingeniería de software de la universidad EAFIT y de la empresa de desarrollo de software AVANSOFT .SA.

En este trabajo se presenta la propuesta para usar los Templates de UML 2.0 como una notación que soporta aspectos en el modelado de software y su implementación en una herramienta CASE y se organiza así: en el primer capítulo se plantea el problema y su sistematización, en el segundo capítulo encontramos los objetivos tanto general como específico , el tercer capítulo se describe el marco teórico estudiado en con fines de la realización de proyecto, el cuarto capítulo presenta el uso de los templetes en las aproximaciones de desarrollo orientadas a aspectos haciendo un énfasis en las aproximaciones Theme y AOSD/UC, en el quinto capítulo se encuentra un breve estudio del soporte que ofrecen las herramientas CASE sobre los Templates, en el capítulo seis se presentan los cambios a los que fue sometida la herramienta CASE AR2CA para

el realizar la implementación de aspectos, por último se presentan las conclusiones del trabajo y trabajos futuros.

1. DESCRIPCIÓN DEL PROBLEMA

1.1. PLANTEAMIENTO DEL PROBLEMA

El fortalecimiento del desarrollo de software en Colombia en los últimos años ha comenzado a ser una preocupación principal en las empresas que quieren llegar a adquirir un alto nivel de competitividad ya que el manejo adecuado y rápido de su información y organización de sus procesos ofrece rentabilidad y calidad a la hora de ofrecer productos o servicios. Con el propósito de apoyar el auge del desarrollo de software la comunidad investigadora propone aproximaciones metodológicas que proveen estrategias y tecnologías que permiten la generación de productos estandarizados y generalizados que favorecen la comercialización y mantenibilidad de los productos de software.

La programación orientada a objetos (POO), la adopción del lenguaje de modelado unificado UML como un estándar de modelado, los lenguajes de programación maduros como Java y las diferentes herramientas CASE que apoyan el proceso de modelado e implementación con la generación automática de código han posicionado la orientación a objetos como la aproximación por defecto en los proyectos empresariales reales. Sin embargo, lograr este nivel en el desarrollo de software no fue fácil, mientras que el enfoque de objetos ya era utilizado con propiedad en industrias de talla mundial a mediados de la década de los 90, nuestras industrias de software tienen poco tiempo de aplicación de esta tecnología.

El desarrollo de software orientado a aspectos es una nueva aproximación que ofrece la ingeniería de software, la cual está siendo estudiada desde hace algunos años por diferentes grupos de investigación a nivel mundial y ha venido tomando fuerza como respuesta a algunas debilidades de la orientación a objetos. Al igual que la orientación a objetos, la orientación a aspectos ha nacido desde una propuesta a nivel de lenguaje de programación y luego ha comenzado a tomar fuerza como aproximación en todas las fases del desarrollo, conocida como AOSD (*Aspect Oriented Software Development*). Estos estudios han generado diferentes aproximaciones orientadas a brindar una solución para las etapas del desarrollo de software, tal es el caso de THEME[11], AOSD/UC[10], UFA[14], AODM[15], entre otras, quienes conservando UML como base del modelado tratan de extender sus artefactos para la representación de aspectos en el ciclo de vida del desarrollo de software.

Pensando en disminuir la brecha de adopción tardía de tecnología a la que estamos acostumbrados se torna de vital importancia incorporar en nuestras estrategias de desarrollo de software la orientación por aspectos. Para ello se

requiere desarrollar un marco metodológico para el apoyo de todo el ciclo de vida de desarrollo de software orientado a aspectos y en este sentido se hace necesario determinar los artefactos UML más apropiados para el modelado de aspectos. Para hacer efectivo su uso por parte de los equipos de desarrollo de software, las herramientas CASE deben soportar el manejo de aspectos, hecho que justifica la implementación de los artefactos identificados como apropiados para el modelado aspectual.

1.2. FORMULACIÓN DEL PROBLEMA

El interrogante que se formula en este trabajo es:

¿De qué manera los Templates de UML 2.0 soportan el modelado de aspectos y cuál sería la mejor forma de adaptar la herramienta CASE AR2CA para que admita el modelado de aspectos haciendo uso de los Templates de UML 2.0?

1.3. SISTEMATIZACIÓN DEL PROBLEMA

¿Cómo se podría determinar el rol de los Templates de UML 2.0 en el modelado de aspectos?

¿Qué uso se le está dando hoy en día a los Templates de UML 2.0?

¿De qué manera se podría integrar los constructores Templates de UML 2.0 como un artefacto para modelar aspectos en una herramienta CASE?

1.4. ALCANCE

El alcance de este proyecto es implementar en la herramienta CASE “Arquitectura de Refinamiento y Reutilización de Componentes de Análisis AR2CA” las modificaciones necesarias para darle a la herramienta la capacidad de soportar diseños con Themes, obedeciendo la arquitectura de la herramienta. Para esto se realizara el estudio de la especificación de UML correspondiente a los constructores Templates y se tendrán en cuenta las extensiones que la aproximación Theme realiza sobre la aproximación UML.

1.5. JUSTIFICACIÓN

El desarrollo de software orientado a aspectos (*Aspect Oriented Software Developmen AOSD*) es una disciplina muy prometedora que constituye una alternativa válida para mejorar el proceso de desarrollo de software, en un intento

de superar la creciente complejidad de los sistemas de software. Las técnicas orientadas a aspectos extienden técnicas tradicionales como la orientación por objetos, permitiendo a los desarrolladores de software encapsular en módulos separados varios conceptos que normalmente se dispersan en toda la estructura de otras unidades funcionales, a estos módulos se los denomina Aspectos. El concepto de Aspecto es adoptado en todas las fases del ciclo de vida del desarrollo de software. Así, los aspectos aparecen en ingeniería de requisitos, el análisis, en el diseño y en la implementación de las aplicaciones software. El beneficio principal de esta tecnología es una mejora de modularizacion de los sistemas obteniéndose un código menos enmarañado, facilitando el mantenimiento.

Sin embargo, los aspectos constituyen una disciplina emergente y por lo tanto no exenta de cuestiones y problemas aún no resueltos completamente. Por ejemplo, aun se necesitan notaciones para expresar aspectos en cada uno de los diferentes niveles de abstracción que componen el ciclo de desarrollo de software y otro problema por resolver es que todavía no hay herramientas CASE que soporten los artefactos necesarios para modelar aspectos. Como respuesta al primer problema los trabajos actuales sobre aspectos buscan maneras convenientes de extender UML ("*Unified Modeling Language*"), para que este lenguaje de diseño sea capaz también de expresarlos.

UML es el lenguaje de modelado con mayor respaldo de la ingeniería de software y está consolidado como el lenguaje estándar en el análisis y diseño de sistemas de cómputo. Mediante UML es posible establecer una serie de requerimientos y estructuras necesarias para plasmar un sistema de software, facilita a integrantes de un equipo multidisciplinario participar e intercomunicarse fácilmente durante todo el proceso de desarrollo de software. Los mecanismos de extensibilidad y nuevos constructores incorporados en la versión 2.0, permiten extender la notación y semántica de UML, de esta forma convertirse en una especie de especificación abierta que puede cubrir aspectos de modelado no especificados en su documentación propia y personalizar el modelado de procesos según las necesidades del sistema y los desarrolladores. Estas nuevas características son caso de estudio para establecer la forma en que UML puede llegar a modelar aspectos.

Un Template es un constructor de la especificación UML 2.0, el cual se describe como un elemento de modelado el cual es parametrizado por otros elementos del modelo. Para especificar su parametrización un elemento Template posee una firma o *Signature*. En un Template también describe e identifica un patrón para un grupo de elementos particular. Los Templates se pueden usar para diseñar un único elemento de modelado que puede funcionar con diferentes tipos de datos.

Usando Template de clase como un patrón general, se puede crear un grupo de clases que usen los parámetros de Template para definir un comportamiento más

específico. También se pueden adicionar Templates de parámetros a las clases para crear Templates de clase y para colaborar en la creación de patrones. Un parámetro de Template es un espacio dentro del *Signature* que otro elemento de modelo puede llenar cuando un nuevo elemento de modelo es generado desde el Template. Por estas características los Templetes pueden ayudar a modelar algunos elementos del AOSD y es necesario explorarlos afondo para descubrir todas sus potencialidades y hasta donde pueden apoyar en el modelado de aspectos.

De la mano de la especificación de una notación para aspectos está el desarrollo de un herramienta CASE que soporte esta nueva tecnología ya que la principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados esto debido a que permiten a los analistas examinar el modelado del sistema con mayor facilidad para detectar errores antes de la codificación además permite solucionar sustancialmente problemas de análisis y diseño como lo son la lógica del diseño y la coherencia de los módulos del sistema. En un segundo término las herramientas CASE aumentan la productividad de los desarrolladores ya que se pueden completar las mismas actividades de desarrollo en un tiempo menor que el que se utiliza cuando no se recurre a estas herramientas, esto se consigue a través de la automatización de determinadas tareas, como la generación de código y la reutilización de módulos. Por tal motivo AOSD requiere del soporte que brindan las herramientas CASE y también requiere que estas herramientas evolucionen para brindar los artefactos requeridos para el modelado de aspectos.

El grupo de investigación en ingeniería de software de la universidad EAFIT en conjunto con la Universidad Nacional, sede Medellín, desarrollaron un proyecto cofinanciado por Colciencias alrededor de las herramientas CASE que lleva por título “Extensiones en Herramientas CASE con Énfasis en Formalismos y Refinamiento AR2CA”, esta herramienta ya está siendo probada en empresas de desarrollo de software como AVANSOFT y PSL de la ciudad de Medellín.

La presente investigación busca extender la funcionalidad de la herramienta CASE AR2CA con el uso de los Templates de UML 2.0 para el soporte del modelado orientado a aspectos.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

Implementar el uso de los Templates de UML 2.0 para el manejo de aspectos en la herramienta CASE AR2CA, a partir del estudio correspondiente sobre los Templates de UML 2.0.

2.2. OBJETIVO ESPECIFICO

- Determinar el rol de los Templates en el modelado de aspectos a partir del estudio del metamodelo UML 2.0.

- Determinar el nivel de uso de los Templates en herramientas CASE existentes para determinar la manera de incorporar su uso en el modelado de aspectos.

- Realizar el análisis y diseño de una propuesta del uso de los tempates de UML 2.0 en la herramienta AR2CA para el manejo de aspectos.

- Realizar la codificación de las nuevas funcionalidades de la herramienta CASE AR2CA para que soporte el manejo de aspectos a partir del uso de los Templates de UML 2.0.

3. MARCO TEÓRICO

3.1. LENGUAJE UNIFICADO DE MODELADO UML

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

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

UML es solo un lenguaje y por lo tanto es tan solo una parte de un método de desarrollo de software. UML es independiente del proceso, aunque su origen está estrechamente relacionado con RUP, el cual es un marco metodológico dirigido por los casos de uso, centrado es arquitectura, interactivo e incremental. Un proceso bien definido guiará a los usuarios al decidir los artefactos a producir, que actividades y que personal que se emplea para crearlos y gestionarlos, y como usar estos artefactos para medir y controlar el proyecto de forma global. Sin embargo UML es algo más que un conjunto de símbolos gráficos. Detrás de cada símbolo hay una notación UML bien definida. De esa manera un desarrollador puede escribir un modelo, y otro desarrollador o incluso una herramienta puede interpretar el modelo sin ambigüedad e implementar la solución. [1]

3.1.1. Breve historia de UML. Los lenguajes de modelado orientados por objetos aparecieron en algún momento entre la mitad de los setenta y finales de los ochenta cuando los metodologistas, enfrentados a los nuevos lenguajes de programación orientados a objetos y a unas aplicaciones cada vez más complejas, empezaron a experimentar con enfoques alternativos al análisis y al diseño. El número de métodos orientados por objetos se incremento de menos de 10 a más de 50 durante un periodo entre 1989 y 1994. Muchos usuarios de estos métodos tenían problemas al intentar encontrar un lenguaje que cubriera las necesidades completamente, alimentando de esta forma la llamada guerra de métodos. Aprendiendo de esta experiencia, comenzaron a aparecerse nuevas generaciones de métodos entre los que destacaron de manera muy clara unos pocos métodos en especial el método de Booch, el método OOSE (*Object-Oriented Software*

Engineering, Ingeniería de Software Orientada por Objetos) de Jacobson y el método OMT (*Object Modeling Technique*, Técnica de Modelado de Objetos) de Rumbaugh. Otros métodos importantes fueron *Fusion, Shlaer-Mellor y Coad-Yourdon*. Cada uno de estos era un método completo aunque todos tenían sus puntos fuertes y sus debilidades. En pocas palabras el método de Boche era bastante expresivo durante las fases de diseño y construcción de proyectos, OOSE proporcionaba un soporte excelente para los casos uso como forma de dirigir la captura de requisitos, el análisis y diseño de alto nivel y MOT-2 era principalmente útil para el análisis y los sistemas de información con gran cantidad de datos.

Una primera masa crítica de ideas comenzó a formarse en la primera mitad de los noventa cuando Grandy Booch (*Rational Software Corporation*), Ivar Jacobson (*Objetory*) y James Rumbaugh (*General Electric*) empezaron a adoptar ideas de cada uno de los otros métodos, los cuales habían sido reconocidos en conjunto como los tres principales métodos orientados a objetos a nivel mundial. Los creadores de los principales métodos de Booch, OOSE, OMT, se sintieron motivados a crear un lenguaje unificado de modelado por tres razones. En primer lugar cada uno de los métodos ya estaba evolucionando independientemente hacia los otros dos. Tenía sentido hacer continua esa evolución de forma conjunta en vez de hacerlo por separado, eliminando la posibilidad de cualquier diferencia gratuita e innecesaria que confundiría aun más a los usuarios. En segundo lugar al unificar los métodos, se podría proporcionar cierta estabilidad en el mercado orientado a objetos, permitiendo que los proyectos se pusieran de acuerdo en un lenguaje de modelado maduro. En tercer, con la colaboración mutua se generaba mejoras a los tres métodos anteriores cubriendo problemas que los 3 métodos que ninguno de los métodos había manejado bien anteriormente. [1]

3.1.2. Versiones UML

- **Versión 0.8.** El objetivo fue hacer la unificación de los Métodos Booch y OMT. El borrador se publico en octubre de 1995 y fue denominado el Método Unificado.

- **Versión 0.9.** El alcance de proyecto se amplia para incorporar OOSE. El documento se publica en 1996. Y de solicita retroalimentación de la comunidad internacional. Con este fin se establece un consorcio con varias organizaciones que dedican recursos para trabajar en una definición fuerte y completa de UML.

- **Versión 1.0.** Las organizaciones *Digital Equipment Corporation*, *Hewlett-Packard, I-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational, Texas Instrument, y Unisys* colaboran en la producción de esta nueva versión donde se presenta un lenguaje de modelado bien definido, expresivo, potente, y aplicable a un amplio espectro de dominios de problemas.

El *Object Management Group's* (OMG) emite una RFP (*Request for Proposal*) para un lenguaje de modelado estándar. UML 1.0 de ofrece como respuesta a esta petición de estandarización de parte del OMG en enero de 1997.

- **Versión 1.1.** Entre enero y julio de 1997 el grupo de colaboradores se amplia. Se forma un grupo de trabajo para la semántica, para formalizar la especificación UML y para integrar UMLM con otros esfuerzos de estandarización, de este esfuerzo se ofrece la versión revisada 1.1 al OMG para su estandarización en julio de 1997. En Septiembre de1997, esta versión fue aceptada por la *OMG Analysis and Desing Task Force (ADTK)* y el *OMG Architecture Board* y se somete a votación de todos los miembros de OMG. Como resultado el 14 de noviembre de 1997 UML 1.1 fue adoptado por el OMG.

- **Versiones siguientes.** La *OMG Revision Task Force* asume el mantenimiento de UML creando las versiones 1.3, 1.4, 1.5. Del año 2000 al 2003, un conjunto nuevo y ampliado de colaboradores crea una versión actualizada UML 2.0, esta versión fue revisada durante un año por una *Finalization Task Force* (FTF) y el OMG adopta UML 2.0 a principios del 2005.

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

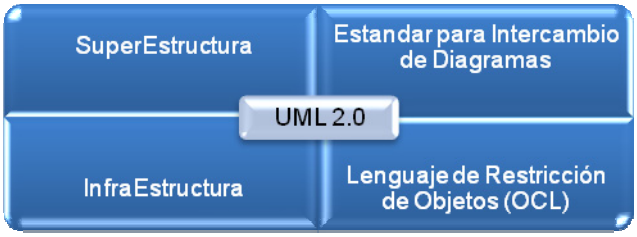

Figura 1: Paquete completo UML 2.0

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

tiempo de ejecución (comparar modelos ejecutables) incluyendo la especificación de estructuras jerárquicas y comportamientos dinámicos.

Para esto se refina la semántica de las relaciones, incluyendo generalización, asociación y dependencia, también se mejorar el encapsulamiento y la escalabilidad en los modelos de comportamiento, especialmente en los diagramas de estado y en los diagramas de interacción, y se refina la semántica gráfica de las actividades, centrándose en la gestión de eventos y el flujo de control y de objetos.

3.1.3.2. UML 2.0 infraestructura. Contiene los constructores básicos para la definición y adaptación de UML, y proporciona los mecanismos de extensión de UML. Surge como respuesta a las propuestas para mejorar las bases arquitectónicas de UML, incluyendo su núcleo y sus mecanismos de extensión. Para esto mejorar la alineación arquitectónica con otros estándares de modelado del OMG, como MOF (*Meta Object Facility*) y XMI (*XML Metadata Interchang*e), reestructurar la arquitectura del lenguaje, para que sea más sencillo de entender, implementar y extender, manteniendo la semántica que ya había sido contrastada, y proporciona perfiles y mecanismos de extensión de primera clase (metaclases) que fueran consistentes con la arquitectura del metamodelo.

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

Surge como estándar XMI que usa el esquema XML para contener información sobre el modelo y como se ha de representar dicho modelo. La alineación del metamodelo UML 2.0 con el metamodelo MOF simplificará el intercambio de modelos vía XMI y la interoperabilidad cruzada entre herramientas.

3.1.3.4. Lenguaje de restricción de objetos (OCL). Contiene la definición de un metamodelo de Lenguaje de Restricciones de Objetos (OCL) acorde al metamodelo de UML, para definir en un diagrama invariantes, precondiciones, poscondiciones y restricciones. Con esto de incrementa la precisión y consistencia de las implementaciones OCL y facilita el intercambio de constructores OCL entre distintas herramientas.

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

Dentro de la especificación de Superestructura de UML los constructores Templates usan 6 clases, estas determinan sus características, a continuación se hará una descripción de estas clases y se presentan ejemplos gráficos que se toman de [2].

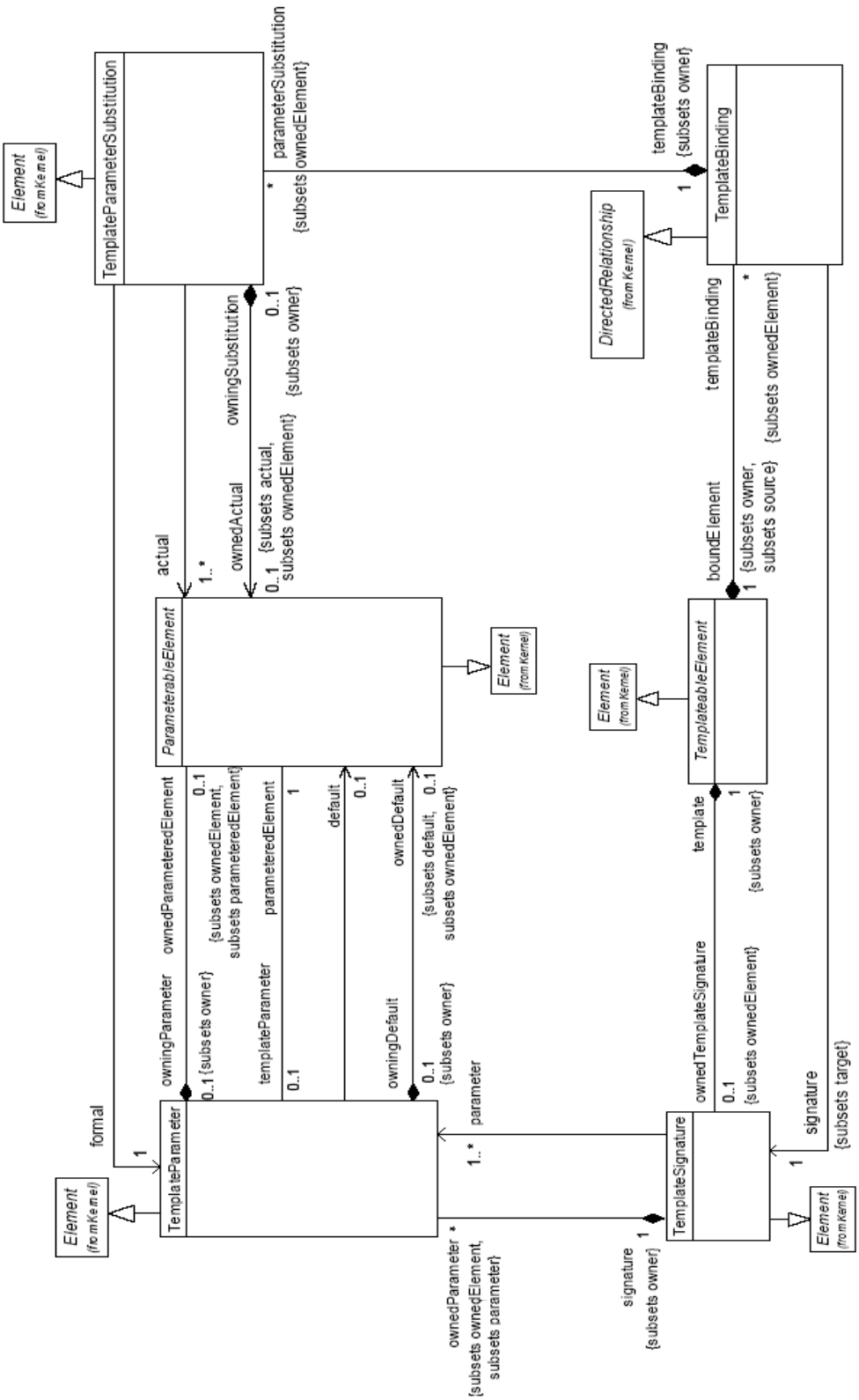

Figura 2: Clases que brindan las características de los constructores Template

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

Un *ParameterableElement* puede ser referenciado por la clase *TemplateParameter* (ver 3.1.4.4) cuando este va a ser definido como un parámetro formal de un *Template*, en este caso este *ParameterableElement* es usado como restricción para los argumentos actuales que se especificarán en la relación *TemplateBinding* de modo que los parámetros actuales que esta relación contenga deberán respetar el tipo de elemento que representa el *ParameterableElement*.

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

Un *ParameterableElement* también puede ser referenciado por la clase *TemplateParameterSubstitution* (ver 3.1.4.5) donde será usado como un parámetro actual dentro de una relación *TemplateBinding*.

3.1.4.2. TemplateableElement. Se describe como un elemento que puede ser definido opcionalmente como una plantilla (*Template*) o en un elemento vinculado (*Bound Element*). Un *TemplateableElement* puede contener un *Templatesignature* (ver 3.1.4.6), el cual especifica los parámetros formales de un *Template*. Un *TemplateableElement* que contiene un *Templatesignature* después es referenciado como plantilla o *Template*.

Un *TemplateableElement* puede contener un *TemplateBinding* (ver 3.1.4.3), el cual describe como los parámetros formales son remplazados por los parámetros actuales. Un *TemplateableElement* que contenga un *TemplateBinding* es después referenciado como un elemento vinculado o *Bound Element*.

Si un *TemplateableElement* tiene parámetros de *Template* (*TemplateParameter*), un pequeño rectángulo de línea punteada se superpone en el símbolo del *TemplateableElement*, normalmente en la esquina superior derecha. El rectángulo punteado contiene una lista de parámetros de *Template* denominados formales. La lista de parámetros no puede estar vacía, aunque podría ser suprimida en la presentación. La lista de parámetros formales del *Template* puede ser mostrada como una lista separada por comas, o se puede presentar un parámetro formal de *Template* por línea. Un elemento vinculado tiene la misma notación gráfica que otros elementos del tipo al que corresponda por ejemplo una clase, un paquete etc. (ver Figura 3)

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

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

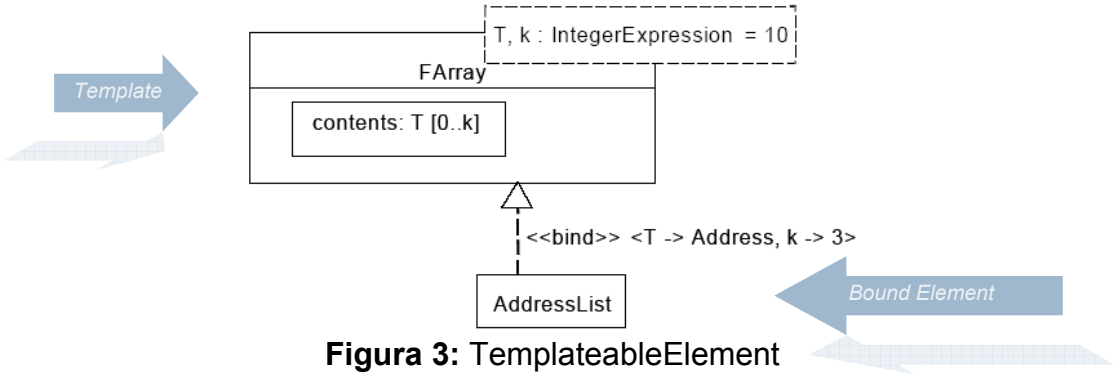

Figura 3: TemplateableElement

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

La semántica de una relación *TemplateBinding* es equivalente al modelo de elementos que se derivan de copiar el contenido del *Template* dentro de un elemento vinculado, remplazando todos los elementos expuestos como *TemplatesParameter* o parámetros formales con los elementos correspondientes especificados como parámetros actuales en el *TemplateBinding*. Si no se especifica el parámetro actual en el *TemplateBinding* para un parámetro formal, entonces el valor por defecto definido en el parámetro formal del *Template* es usado automáticamente. (Ver Figura 4)

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

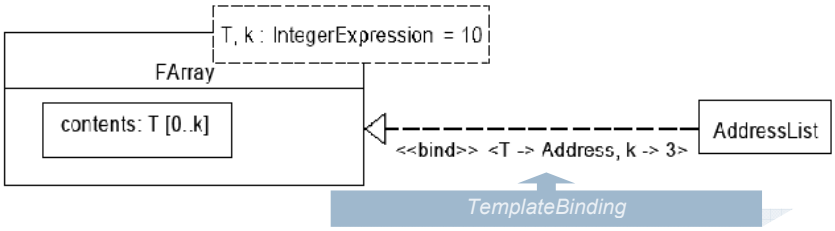

Figura 4: TemplateBinding

3.1.4.4. TemplateParameter. Un *TemplateParameter* expone un *ParameterableElement* como un parámetro formal de *Templates*. Este *ParameterableElement* sólo tiene sentido dentro del *Template* o de otras *Templates* que pueden tener acceso a su funcionamiento interno (por ejemplo, si el *Template* soporta especialización). Los *ParameterableElement* expuestos como *TemplateParameter* no pueden ser utilizados en otras partes del modelo. (Ver Figura 5)

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

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

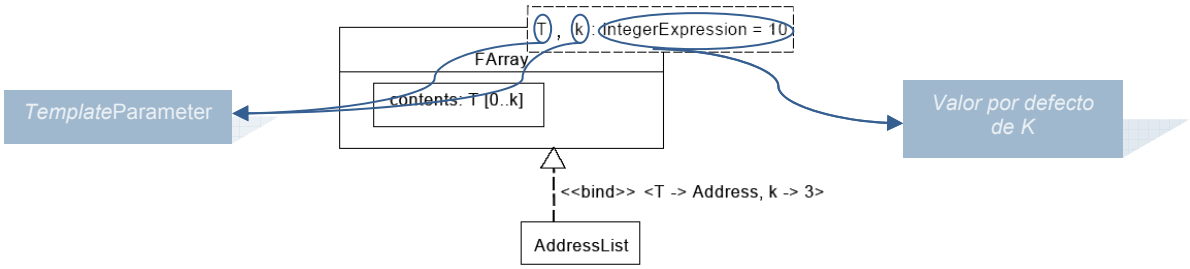

Figura 5: TemplateParameter

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

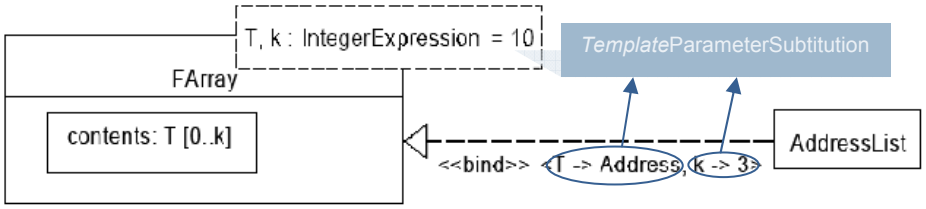

Figura 6: TempleParameterSubstitution

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

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

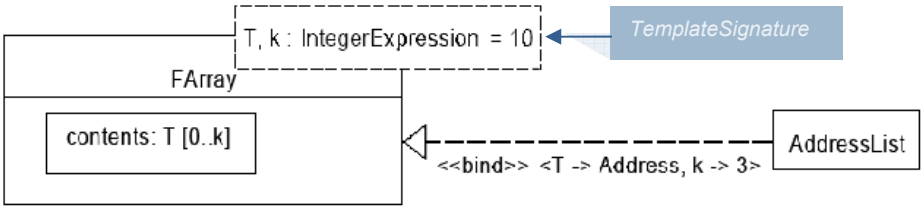

Figura 7: Templatesignature

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

Un elemento vinculado también es un *TemplateableElement*. Al relacionar un elemento vinculado con un Template, se genera una relación de tipo *TemplateBinding*. Esta relación contiene los valores específicos que remplazaran a cada uno de los parámetros formales que se encuentran en el *Templatesignature* del Template. Estos valores recibe el nombre de *TemplateParameterSubstitution*.

3.2. DESARROLLO DE SOFTWARE ORIENTADO POR ASPECTOS AOSD

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

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

3.2.1. Separación de concerns. AOSD es un paradigma de ingeniería del software que conduce a reducir la complejidad de los sistemas software a través de la separación de *concerns* o intereses. Los *concerns* son los diferentes temas o asuntos de los que es necesario ocuparse para resolver el problema. Por ejemplo una función específica que debe realizar una aplicación, pero también surgen otras como por ejemplo distribución, persistencia, replicación, sincronización, etc. La separación de *concerns* se refiere a la habilidad para identificar, encapsular y manipular de forma aislada aquellas partes del software que son relevantes para un concepto dado, objetivo o propósito [3], centrada en un principio en la etapa de implementación y posteriormente aplicada a todas las etapas del ciclo de vida del software.

Una de las corrientes más relevantes dentro de AOSD es MDSOC [4] (*Multidimensional Separation of Concerns*) que promueve la encapsulación de diferentes tipos de *concerns* organizados en dimensiones y la integración de las mismas para formar el sistema completo. Cada una de estas dimensiones está compuesta por *concerns* del mismo tipo encapsulados de forma independiente. Estas piezas separadas (artefactos que encapsulan *concerns*) serán integradas a través de una serie de reglas de composición para formar el sistema completo. Todas las dimensiones son tratadas de forma arbitraria, posibilitando que un desarrollador se centre en determinadas características del software sin tener que

conocer las demás. Por *concerns* del mismo tipo se pueden entender propiedades que descomponen el software atendiendo a un determinado criterio.

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

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

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

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

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

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

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

de otros módulos. Consecuentemente, los desarrolladores de estos otros módulos pueden ser inconscientes del *crosscutting concern*.

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

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

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

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

3.2.6. Tejedor (weaver). La interacción entre clases y aspectos se hace posible a través de un tercer componente, el cual conocemos como tejedor. El tejedor es el encargado de realizar la mezcla de la funcionalidad base con el comportamiento

aspectual. Las clases y los aspectos se pueden mezclar de dos formas distintas: de manera estática y de manera dinámica.

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

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

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

- Clases, las cuales modelan la funcionalidad base
- Aspectos
- Tejedor

En el esquema tradicional, la construcción de la aplicación se realiza a través de un compilador o intérprete, el cual simplemente toma el código de la funcionalidad base y lo traduce a un lenguaje entendible por la maquina. La construcción de aplicaciones bajo este esquema se muestra en la Figura 8.

Figura 8: Construcción de aplicaciones enfoque tradicional

La construcción de una aplicación bajo el enfoque orientado a aspectos requiere la combinación del código de la funcionalidad base con los distintos módulos que implementan los aspectos. Esta combinación es realizada por el tejedor. Cada aspecto codificado con un lenguaje distinto. La construcción de aplicaciones bajo este esquema se muestra en la Figura 9.

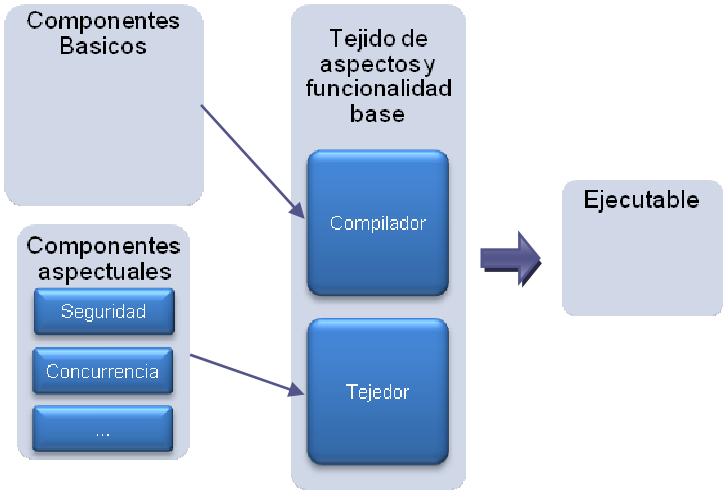

Figura 9: Construcción de aplicaciones Orientado por aspectos

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

3.3. APROXIMACIONES DE DISEÑO ORIENTADAS POR ASPECTOS

La actividad de diseño de un proceso de desarrollo de software da a un diseñador la oportunidad de razonar sobre un sistema de software requerido, tal como se define por un conjunto de requisitos. Este proceso de razonamiento sobre el sistema implica un estudio sobre el comportamiento necesario para que el sistema logre sus objetivos y satisfaga las restricciones, de tal manera se defina una solución vista desde diferentes perspectivas (estática, dinámica) y diferentes niveles de detalle (diseño de la arquitectura y diseño detallado). Por ejemplo, en un sistema de software orientado por objetos, el diseñador probablemente considerara el comportamiento en términos de por lo menos diagramas de interacción y diagramas de estado, y la estructura en términos de por lo menos diagramas de clase y objetos. El diseñador puede iterar sobre las diferentes vistas, teniendo en cuenta cada vez más niveles de detalle con el tiempo. La salida resultante de la actividad de diseño es un conjunto de modelos que caracterizan y especificar el comportamiento y la estructura del sistema requerido. Estos modelos pueden ser a diferentes niveles de abstracción dependiendo del nivel de detalle del razonamiento del diseñador. Los estándares de Ingeniería de software en cuanto a medidas de calidad de la producción incluyen criterios como la cohesión

y acoplamiento de los módulos descritos. Asimismo el producto software debe satisfacer criterios de calidad establecidos por todos los involucrados (amigabilidad, seguridad, desempeño, etc.), los cuáles se debe tener en cuenta a la hora de definir la arquitectura y establecer la alternativa de solución más adecuada.

El diseño orientado por aspectos (AOD) tiene los mismos objetivos que cualquier actividad de diseño de software; caracterizar y especificar el comportamiento y la estructura del sistema de software. Su contribución al diseño de software se refiere a extensiones de capacidad de modularidad. Los *concerns* de un sistema de software que son necesariamente dispersos y enmarañados en una aproximación no-AOD puede ser modularizados. Una aproximación de diseño AOD proporcionará los constructores del lenguaje para apoyar la modularización de *concerns*, independientemente de si el *concern* tiene un impacto o entrecruza otros *concerns*. AOD también prestará el soporte para la especificación de composición de *concerns*, teniendo en cuenta los conflictos o co-operaciones. Más allá de eso, el diseño de cada uno de los módulos de *concerns* es probable espejo de un estándar de diseño de software.

Una aproximación AOD probablemente incluya un proceso y un modelo. Un proceso AOD es aquel que articula todas las actividades tanto de ingeniería como de gestión y apoyo para producir el producto software. Desde el punto de vista de ingeniería, lo ideal es que se aplique un enfoque de levantamiento y análisis de requisitos teniendo en cuenta los intereses (concerns) de todos los involucrados y a partir de estos requisitos producir un modelo de diseño que pueda realizar parte de una arquitectura. El modelo AOD producido durante el proceso de AOD representa por separado los *concerns* y las relaciones entre *concerns*. Este modelo es una especificación resumida para la aplicación que puede ocurrir en una plataforma AOD o de otro tipo.

Un lenguaje AOD es aquel que incluye los constructores que pueden describir los elementos para ser representado en el diseño y las relaciones que pueden existir entre esos elementos. En particular para los lenguajes AOD, los constructores se proporcionan para apoyar la modularización de *concerns*, y la especificación de composición de *concerns*. Esto incluye un medio para captar los conflictos y especificaciones de cooperación. [8]

En el momento existen muchas aproximaciones de desarrollo de software orientado a aspectos las cuales se orientan al uso de UML para especificar una notación de lenguaje para modelar aspectos, de estas Theme, y AOSD/UC se distinguen por tener soporte en las etapas primarias del desarrollo de software como lo son análisis de requisitos, análisis y diseño del sistema. A continuación se presenta el modelo y el proceso que usan aproximaciones mencionadas [9]

3.3.1. AOSD/UC. Las tecnologías orientadas por aspectos proveen mecanismos de composición, que permiten componer partes, clases o componentes que realizan diferentes *crosscutting concern*, los que integrados forman clases o componentes completos.

Además del mecanismo de composición se requiere de una técnica de separación de *crosscutting concern* para capturarlos y estructurarlos durante los requisitos y mantener esa separación a lo largo del análisis, diseño y codificación, AOSD/UC propone alcanzar este objetivo expresando los *crosscutting concern* con el empleo de casos de uso.

Se puede modelar y estructurar casi cualquier *crosscutting concern* con casos de uso, de hecho, no se conoce un *cosscutting concern* funcional por naturaleza que no pueda ser representado como caso de uso. En el proceso de desarrollo de software se analiza, se diseña, se implementa y se prueba, básicamente de la misma manera, aplicando casos de uso. [10]

El enfoque basado en casos de uso propone modelar las funcionalidades propias del sistema como casos de uso base, que pueden poseer puntos de extensión, es decir, puntos concretos dentro de un determinado caso de uso, donde es posible añadir comportamiento, este comportamiento se especificaría mediante casos de uso denominados extensiones. De este modo se puede realizar la captura de requisitos en dos fases: una primera donde se modela la funcionalidad básica y propia del sistema mediante los casos de uso base, que posteriormente, en la segunda fase, se adornan con los requisitos representando aspectos. Dado que las extensiones son comunes a diversas aplicaciones, podrían reutilizarse, tomándose de un repositorio genérico [10].

3.3.1.1. Método AOSD/UC. AOSD con casos de uso [11] amplía el tradicional enfoque de casos de uso [16] con dos elementos principales: pointcuts para casos de uso y la agrupación de artefactos de desarrollo en caso de uso slice y caso de uso módulo:

- Los pointcuts son agrupaciones de joinpoints de caso de uso representados por puntos extensión y elementos tales como clases, operaciones, etc dentro de un caso de uso. Considerando que un punto de extensión (punto en el cuerpo de la especificación del caso de uso) es un punto solo usado para referenciar un caso de uso *extension* particular.
- caso de uso slice: contiene los detalles de un caso de uso para una determinada fase de desarrollo, por ejemplo, los requisitos o diseño.
- caso de uso del módulo: por otra parte un caso de uso módulo, contiene todos los detalles relacionados con un caso de uso, a través de todas las fases de desarrollo.

El enfoque distingue dos tipos de casos de uso: caso de uso *Peer* y caso de usos *extension*. Los casos de uso *Peer* son distintos e independientes entre sí, cada uno puede utilizarse por separado sin referencia alguna a los demás, son los requisitos base. Cuando los casos de uso *Peer* se componen, todas las operaciones sin intervención en su ejecución se componen. Sin embargo, la composición de los casos de uso *Peer* tendrá en cuenta la superposición de comportamiento y los conflictos entre las clases utilizadas para la realización de diferentes casos de uso.

Los *extensions* son funciones adicionales en la parte superior de los casos de uso base. A pesar de las extensiones puede definirse independientemente los casos de uso base, normalmente, se utilizan junto a la base. Cuando las extensiones están compuestas con casos de uso base, sus operaciones normalmente interfieren con la ejecución de las operaciones de los caso de uso base.

El AOSD/UC también promueve la captura de las requisitos no-funcionales como casos de uso, utilizando un concepto denominado caso de uso infraestructura que se refiere a las actividades que la infraestructura del sistema necesita para llevar a cabo y satisfacer las necesidades de los usuarios.

3.3.1.2. Proceso AOSD/UC. El proceso de aplicación de la aproximación orientada por aspectos AOSD/UC involucra el siguiente proceso [17]:

- **Entender los asuntos de los participantes.** En este paso se realiza el proceso de recolectar los requisitos del sistema con el fin de determinar el alcance de la solución, para esto se determina el dominio del problema, los módulos del sistema y se establecen las características del los módulos, como por ejemplo las reglas de negocio, asuntos trasversales, y se identifican los requisitos funcionales y no funcionales.

- **Capturar casos de uso de aplicación.** Los casos de uso de aplicación se establecen a partir de los requisitos funcionales. En cada modulo ya sea aspectual o básico se identifican los casos de uso de aplicación. En la descripción de los casos de uso se establecen los flujos básicos, los flujos alternativos y flujos secundarios o subflujos. Los flujos básicos son el caso general; los flujos alternativos son contingencias para cuando el caso general no se aplica; los subflujos son las secuencias del acontecimiento que aparecen en forma repetitiva, están descritos en un lugar específico y referido más adelante. Los casos de uso pueden ser extendidos o se pueden utilizar para extender otros casos de uso, los extension points o puntos de extensión describen un evento donde el comportamiento del caso de uso es crosscutting o trasversal a la aplicación, los flujos de extensión describen el flujo que cruza ese evento, el punto de extensión y

los flujos de extensión se especifican también en la especificación de casos de uso. Una vez establecidos los casos de uso se pasa a establecer las clases y operaciones básicas en el sistema y su respectivo rol.

Durante esta etapa un diagrama de casos de uso se crea para representar el sistema completo, de allí, la base y el comportamiento crosscutting es especificada en las especificaciones de los casos de uso, los diagramas de casos de uso y las especificaciones asociadas se definen y se refinan en un proceso iterativo.

- **Capturar casos de uso de infraestructura.** Los casos de uso de infraestructura se establecen a partir de los requisitos no funcionales. Al igual que en los acasos de uso de aplicación en la descripción de estos casos de uso se establecen los flujos básicos, alternos y se describen los puntos de extensión.

- **Definir las realizaciones de los casos de uso de aplicación.** Los diagramas de secuencia y de colaboración de alto nivel, se utilizan para modelar la secuencia de la interacción entre los conceptos del dominio, estos diagramas del comportamiento son basados en las especificaciones de casos de uso.

Una vez, los casos de uso, las especificaciones de los casos de uso, los conceptos del dominio y la secuencia de iteraciones de alto nivel se identifican en el análisis, entonces inicia el diseño, el diseño se basa en el análisis, los casos de uso se representan con slices que se pueden diseñar por separado, esta aproximación es simétrica para AOD, para cada caso de uso o slice, se construye un modelo de diseño separado, la fase de análisis en AOSD/UC produce un diseño de alto nivel y una vez que es completado se continua con plataforma especifica de diseño.

Para diseñar casos de uso base, el diseñador comienza identificando componentes e interfaces de componentes, los componentes corresponden a los conceptos del dominio identificados en el análisis, en el diseño, los conceptos del dominio se pueden diseñar como una clase o puede requerir un número de clases específicas para proporcionar un diseño que pueda ser implementado.

Cada componente expone interfaces requeridas y proporcionadas, los conceptos de diseño de alto nivel se expresan con un diagrama de componentes, y las relaciones entre estos conceptos de diseño se pueden capturar como relaciones entre componentes.

Una vez que se proporcione esta visión de alto nivel, el caso de uso puede ser representado como un paquete que se referencia explícitamente por el nombre del caso de uso que es realizado, los paquetes de casos de uso contienen los diagramas que proporcionan los detalles de la especificación de componentes.

Los diagramas de clase proporcionan un diseño estructural de los componentes UC-slice, cada clase representa un componente o una parte de un componente, los diagramas de secuencia modelan la interacción entre las clases, los diagramas de secuencia y comunicación (antes diagramas de colaboración) se utilizan para describir el flujo del comportamiento entre clases y objetos, el flujo, en estos diagramas, se basa en los flujos descritos en las especificaciones de los casos de uso y en los diagramas de interacciones de alto nivel creados durante análisis.

Diseñar casos de uso crosscutting es muy similar a diseñar casos de uso base, salvo que los aspectos también necesitan ser modelados, los aspectos se representan como clasificadores, marcados con un estereotipo <<aspect >>, los aspectos están contenidos en el paquete de casos de uso, este clasificador describe el comportamiento y la estructura crosscutting asociados al caso de uso.

Un aspecto es un clasificador nombrado que contiene la declaración del pointcut y las clases de extensión, las cuales son descritas en los diagramas de clases y pueden ser de comportamiento o de estructura crosscutting, para diseñar un aspecto se identifican las clases que especifican comportamiento y estructura crosscutting, así mismo, se identifican los Pointcuts que aseguran la correcta extensión de los casos de uso relacionados y se crean los diagramas de secuencia que ilustran cómo el comportamiento es cruzado, para ilustrar comportamiento crosscutting.

- **Definir las realizaciones de los casos de uso de infraestructura.** Los casos de uso no funcionales se diseñan por medio de la especialización de utilidades de los casos de uso, una utilidad de un caso de uso representa un interés no funcional que es crosscutting, para soportar la reutilización de intereses crosscutting no funcionales, las utilidades de los casos de uso se representan por medio de paquetes plantilla parametrizados.

Diseñar casos de uso no funcionales o casos de uso de utilidad es similar a diseñar un aspecto.

Para crear un aspecto concreto los parámetros de la plantilla son remplazados por los argumentos que son especificados para la aplicación. Para reutilizar casos de uso de utilidad, el diseñador debe elegir un caso de uso de utilidad que resuelva los requisitos no funcionales y utilice argumentos para crear un aspecto concreto.

La composición se modela o se especifica en un alto nivel en los diagramas de caso de uso, por medio de relaciones include, extend y generalizaciones y los modelos de diseño se especifican por separado durante la fase de diseño.

Cuando el diseño de un caso del uso es completado, los diagramas estructurales y de comportamiento creados se utilizan para implementar el interés representado por el caso del uso.

- **Desarrollar la Arquitectura.** La arquitectura del sistema se define a partir de los casos de uso Slice o de los módulos establecidos en el sistema y las relaciones que presentan entre ellos.

3.3.2. THEME/UML. Theme/UML es una extensión de UML que enfatiza en apoyar al diseñador en la especificación de Themes que se traslapan unos con otros, y en las capacidades de composición, que están definidas.

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

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

3.3.2.1. Método THEME/UML. Theme es una aproximación de análisis y diseño que soporta la separación de concerns para las fases de análisis y diseño del ciclo de vida del software. La aproximación Theme también provee un lenguaje de diseño orientado por aspectos basado en UML, llamado Theme/UML que extiende el metamodelo UML.

La aproximación Theme expresa los *concerns* en constructores conceptuales y de diseño llamados Themes, estos son más que aspectos generales, y enfatiza en los *concerns*, cualquier *concern* que entrecruce o no otros *concerns* se puede encapsular en un Theme [11].

3.3.2.2. Proceso THEME/UML. El proceso del Theme se describe en [8] como un proceso de tres fases, análisis, diseño y composición. En la fase de análisis, se identifican y se caracterizan los Themes. En la fase de diseño los Themes identificados y caracterizados se especifican en modelos técnicos del diseño. Finalmente, en la fase de la composición, se especifica la composición de Themes. El proceso completo se ilustra en la Figura 10.

- **Fase de análisis.** La parte del análisis es abordada por Theme/DOC, que soporta la caracterización de dos tipos de Themes, Themes base y Themes aspectuales, las vistas de los Themes base presentan al diseñador los requisitos que son necesarios para producir un modelo de diseño que integre los requisitos; las vistas de los Themes aspectuales presentan a diseñador los requisitos para el Theme aspectual, así mismo, la vista del Theme aspectual también presenta los Themes que son cruzados por el aspecto.

- **Fase de diseño.** La fase de diseño, mostrada como el segundo paso en el proceso del Theme en la Figura 10, toma cada Theme y produce modelos separados y posiblemente traslapados, los requisitos relacionados con la vista del Theme se utilizan para construir los diseños de los Themes, cada uno de los cuales es construido como un diagrama de clases en una construcción separada del Theme, esta construcción es un paquete especializado, en el caso de los Themes aspectuales requieren un diagrama de secuencia adicional que indique dónde y cómo corta el comportamiento de otro Theme.

- **Fase de composición.** La fase de composición inicia una vez que los Themes son capturados por separado, en esta fase se especifica la composición y se diseña la composición.

Los modelos del dominio de Theme/UML se identifican cuando el mismo concepto del dominio se representa como clase en más de un Theme, un traslapo ocurre cuando los requisitos de diversos Themes describen parcialmente un mismo concepto del dominio, este concepto se describe completamente a través de todos los Themes en los cuales es incluido.

En Theme/UML, para especificar la integración o la composición de los Themes aspectuales, es necesario especificar el criterio de selección del *join point*, es decir, los puntos donde los Themes aspectuales cruzan otros Themes, estos puntos se utilizan como argumentos para el paquete parametrizado o Template por medio del cual es expresado el Theme. La composición puede ser realizada una vez que los argumentos están asociados a la plantilla.

El resultado de la composición de conceptos traslapados es un modelo en el cual todos los conceptos del dominio se representan completamente, a sí mismos, el

resultado de componer Themes aspectuales es la generación de modelos donde los *joinPoints* identificados por el criterio de selección son usados como argumentos para los parámetros en las plantillas de los Themes aspectuales. [9]

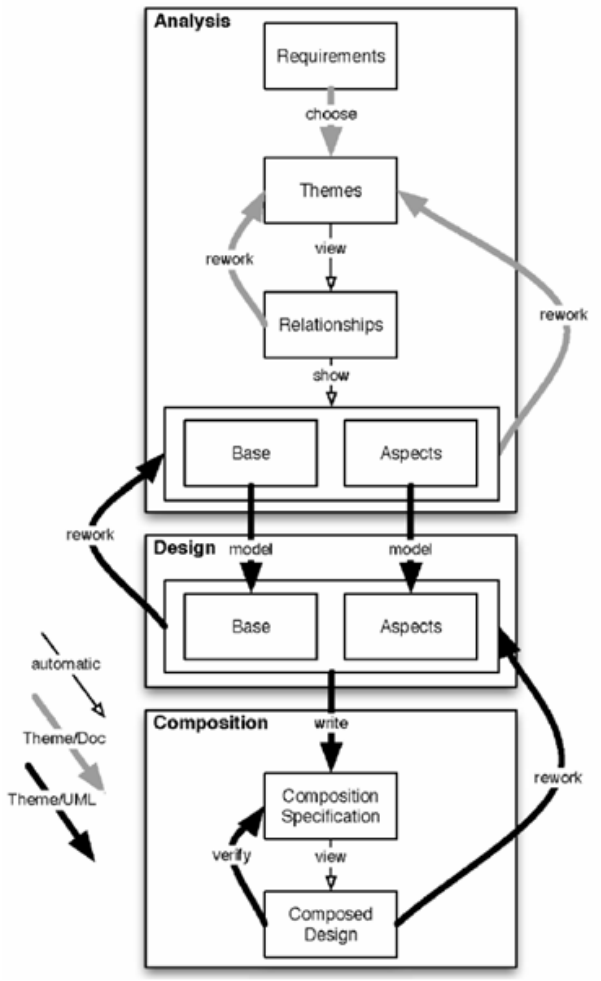

Figura 10: Proceso completo Theme

3.4. HERRAMIENTAS CASE

El desarrollo de herramientas CASE (*Computer Aided Software Engineering*) para el soporte del proceso de desarrollo de software es uno de los aportes más significativos de la ingeniería del software. [16] Su uso, permite automatizar las actividades clave de todo el proceso de desarrollo de un sistema de software, incluida la gestión de requisitos, la representación y transformación de artefactos de arquitectura, el diseño y la generación automática de código. En muchos casos las utilidades de las Herramientas CASE van mucho más allá, algunas proveen facilidades para el soporte de tareas protectoras como la gestión de configuración del software, gestión de riesgos, control de calidad, estimación, entre otras. De

esta manera su uso se convierte en un factor crítico de éxito a la hora de evaluar los proyectos de desarrollo de software y los productos generados. Esta es la razón por la cual el mercado de las herramientas CASE está en constante evolución. Sin embargo, la adopción de las herramientas CASE no ha cubierto totalmente las expectativas; su selección, adquisición, inserción y uso no es una actividad sencilla.

3.4.1. La tecnología CASE. La ingeniería de software asistida por computador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de software. [7]

CASE es una combinación de herramientas software (aplicaciones) y de metodologías de desarrollo: Las herramientas permiten automatizar el proceso de desarrollo del software y las metodologías definen los procesos a automatizar.

La tecnología CASE busca la automatización del desarrollo del software, contribuye a mejorar la calidad y la productividad. EL uso de herramientas CASE en un proyecto de desarrollo de software presupone los siguientes objetivos:

- Facilitar la aplicación práctica de metodologías de desarrollo que al ser realizadas con una herramienta permite agilizar el trabajo.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento del software.
- Mejorar y estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes software.
- Permitir el desarrollo y el refinamiento visual de las aplicaciones.
- Facilitar la trazabilidad de los artefactos de software a lo largo de todo el ciclo de vida del software.
- Agilizar el proceso de producción de código.
- Automatizar el chequeo de errores
- Facilitar la gestión del proyecto

3.4.2. Componentes de una herramienta CASE. En términos generales una herramienta CASE se compone de los siguientes elementos:

- **Repositorio:** Donde se almacenan los elementos definidos o creados por la herramienta, y cuya gestión se realiza mediante el apoyo de un Sistema de Gestión de Base de Datos (SGBD) o de un sistema de gestión de archivos.

- **Meta modelo:** Constituye el marco para la definición de las técnicas y metodologías soportadas por la herramienta.

- **Carga o descarga de datos:** Son facilidades que permiten cargar el repositorio de la herramienta CASE con datos provenientes de otros sistemas, o bien generar a partir de la propia herramienta esquemas de base de datos, programas, etc. que pueden, a su vez, alimentar otros sistemas. Este elemento proporciona así un medio de comunicación con otras herramientas.

- **Comprobación de errores:** facilidades que permiten llevar a cabo un análisis de la exactitud, integridad y consistencia de los esquemas generados por la herramienta.

- **Interfaz de usuario:** que constará de editores de texto y herramientas de diseño gráfico que permitan, mediante la utilización de un sistema de ventanas, iconos y menús, con la ayuda del ratón, definir los diagramas, matrices, etc. que incluyen las distintas metodologías.

3.4.3. Clasificación de las herramientas CASE. No existe una única clasificación de herramientas CASE y, en ocasiones, es difícil incluirlas en una clase determinada. Podrían clasificarse atendiendo las plataformas que soportan teniendo en cuenta:

- Las fases del ciclo de vida del desarrollo de sistemas que cubren.
- La arquitectura de las aplicaciones que producen.
- Su funcionalidad.

También podrían clasificarse considerando la amplitud de la herramienta CASE de la siguiente forma:

TOOLKIT: es una colección de herramientas integradas que permiten automatizar un conjunto de tareas de algunas de las fases del ciclo de vida del sistema informático: Planificación estratégica, Análisis, Diseño, Generación de programas.

WORKBENCH: Son conjuntos integrados de herramientas que dan soporte a la automatización del proceso completo de desarrollo del sistema informático. Permiten cubrir el ciclo de vida completo. El producto final aportado por ellas es un sistema en código ejecutable y su documentación.

Una segunda clasificación es teniendo en cuenta las fases (y/o tareas) del ciclo de vida que automatizan:

UPPER CASE: Planificación estratégica, Requerimientos de Desarrollo Funcional de Planes Corporativos.

MIDDLE CASE: Análisis y Diseño.

LOWER CASE: Generación de código, test e implantación.

3.4.4. AR2CA. AR2CA (Arquitectura de Refinamiento y Reutilización de Componentes de Análisis) es una herramienta CASE que tiene como principal objetivo poner al alcance de los desarrolladores de software una herramienta que apoye el desarrollo de aplicaciones bajo el enfoque Orientado a Objetos, utilizando la notación UML. [12] Esta herramienta CASE ha formado parte de 2 proyectos cofinanciados por Colciencias y desarrollados por el grupo de investigación de Ingeniería de software de la Universidad EAFIT. El primer proyecto se titulo "*Extensiones en herramientas CASE con énfasis en formalismos y reutilización*" hasta este proyecto la herramienta CASE se enfoco en el apoyo sobre el enfoque orientado por objetos. El segundo titulado "*Marco Metodológico Para El Desarrollo Orientado a Aspectos – MMEDUSA"* del cual hace parte este proyecto de grado pretende darle a la herramienta la capacidad de apoyar a los desarrolladores en el diseño de aplicaciones orientadas por aspectos como parte del marco teórico se expone una explicación de la arquitectura de la herramienta y los módulos que la componen.

La herramienta está construida en Java (versión 1.5), como una aplicación multiusuario que trabaja como cliente desconectado (Java Swing) y que permite realizar conciliación de modelos de manera asincrónica soportándose en los servicios provistos por un servidor de CVS (CVSNT 2.0 51d). Maneja información para administración de usuarios en una base de datos (MySQL). La herramienta integró la especificación UML 2.0 desarrollada por el grupo Eclipse y la adoptó para que funcione como un módulo articulado a la capa de presentación y control propia de AR2CA. [13]

3.4.4.1. Vista general de la arquitectura de AR2CA. La arquitectura de AR2CA está fundamentada en tres patrones de arquitectura: el patrón layer, el patrón Modelo Vista Controlador MVC y el patrón DAO. El patrón layer permite estructurar las clases en paquetes con un rol claro dependiendo de la capa en la cual está ubicado; el patrón MVC permite definir el esquema de notificación de cambios en el elemento de UML que se esté trabajando a la capa de presentación; el patrón DAO permite desacoplar la funcionalidad de los objetos del modelo de los servicios de persistencia. La Figura 11, muestra una vista general de paquetes de la herramienta. En el área central se tiene la funcionalidad básica de manejos de modelos en sus capas de presentación, control, modelo y acceso a datos. En la franja de la izquierda se tienen los paquetes que contienen la funcionalidad para

configurar la herramienta a través de factorías y las clases que representan la información meta de los elementos y los servicios relacionados con la administración de usuarios y dominios y la clase encargada de manejar los datos de la sesión de usuario. En la franja de la derecha se tienen los paquetes que agrupan servicios de utilidad específica tales como las utilidades de dibujo, el catálogo de elementos activos, los servicios de CVS, el editor de documentos XML, el editor de expresiones OCL y los servicios de transformación y generación de modelos. La Tabla 1, presenta una breve descripción de cada uno de los paquetes. [13]

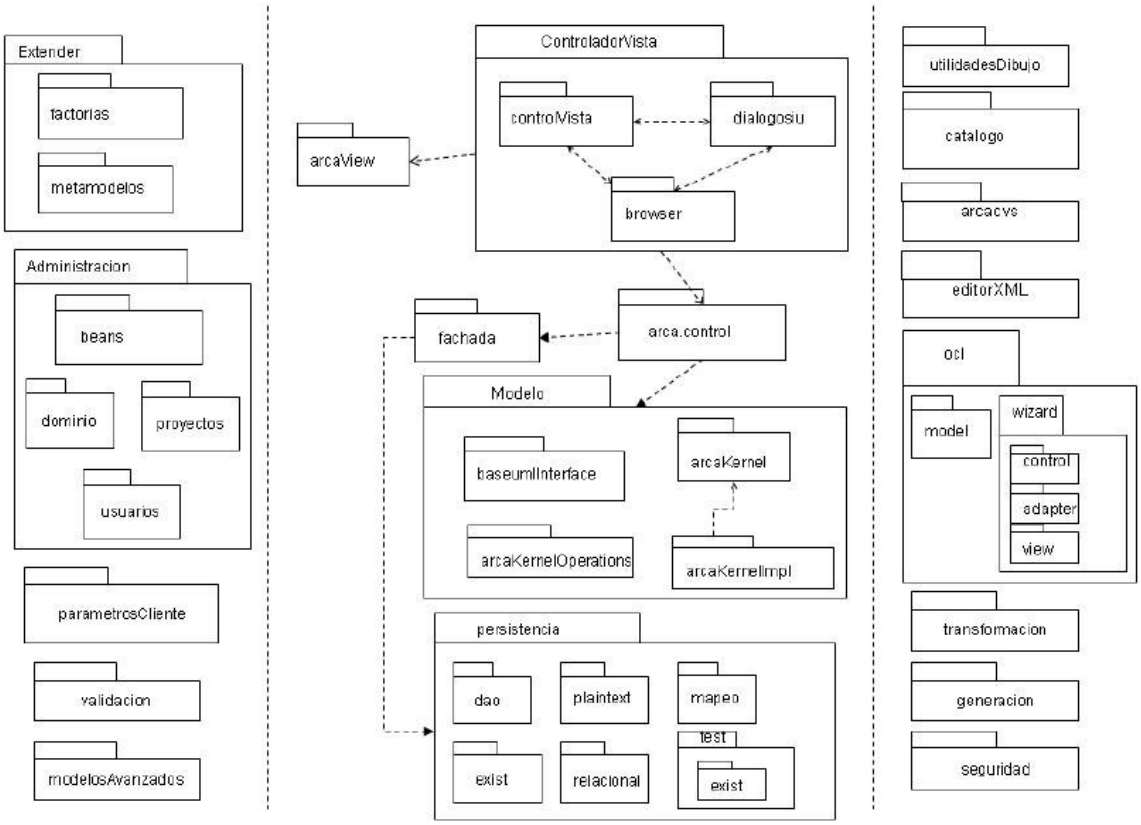

Figura 11: Diagrama general de paquetes de AR2CA

Paquete	Descripción
arcaView:	Contiene las clases que construyen los elementos gráficos que pueden ser usados por los modelos UML disponibles en AR2CA además de las clases que construyen las barras de herramientas para el manejo de estos elementos al interior de la aplicación. Rol: Vista.
arcaKernel:	Contiene la declaración de interfaces del modelo UML2.0. Rol: Modelo.
arcaKernelImp:	Contiene las implementaciones del modelo UML2.0. desarrolladas por el grupo Eclipse Rol: Modelo.
arcaKernelOperation:	Contiene las clases que ofrecen servicios generales al interior de los modelos UML2.0. Rol: Modelo(servicios generales)
baseUMLInterfaces:	Este paquete contiene declaración de las interfaces de los elementos del modelo que son conocidas por los objetos de vista y control. Sirven como interfaces adaptadoras para ocultar a los niveles superiores de la aplicación la complejidad de servicios de las implementaciones particulares de UML. Rol: Adaptadores entre el modelo y las capas de presentación y control.
Administración:	Módulo que contiene los servicios de control de operaciones sobre los modelos y el manejo de proyectos y dominios.
browser:	Contiene las clases que contribuyen a la creación del Browser de AR2CA desde donde se pueden visualizar los nodos que hacen alusión a un elemento determinado creado dentro de un modelo de AR2CA así como su nivel de abstracción. Rol: Control de Vista.
Catalogo:	Contiene las clases que almacenan en memoria los elementos que se encuentran activos en una sesión de usuario. Los elementos que suben a memoria son almacenados en una estructura indexada (java.util.TreeMap) que facilita la ubicación de los elementos por medio de su oid2. Rol: Servicios
Control:	Contiene las clases que concentran todos los servicios de la interfaz hacia el modelo. Rol: Control.
controlVista:	Contiene los elementos gráficos por medio de los cuales el usuario interactúa para ejecutar acciones sobre los modelos. Contiene la clase que representa el marco principal de la aplicación (AR2CAFrame) y los elementos de dibujo que representan un modelo desde el punto de vista gráfico (DibujoCanvas, ConexionDibujo, ElementoDibujo). Rol: Control Vista.

arcacvs:	Paquete que contiene los servicios subir (Check-in) y bajar (check-out) de elementos desde el servidor CVS
persistencia:	Contiene los servicios de persistencia de los diferentes elementos. Tiene dos paquetes anidados: plainText y relacional para la persistencia tanto en documentos XML para los modelos como en la base de datos para la información administrativa. Rol: Servicios de infraestructura (persistencia)
dialogosIU:	Contiene los diálogos por medio de los cuales el usuario puede ingresar o editar elementos del modelo. Rol: Control de Vista.
editorXML:	Clases que configuran un editor desde el cual puede generarse documentación para los modelos creados en AR2CA a partir del lenguaje XML.
extender:	Contiene las clases que representan la información meta de los elementos para permitir su parametrización externa.
fachada:	Contiene los servicios que se realizan contra el almacenamiento persistente. Sirve para separar la lógica de acceso a datos del resto de clases del aplicativo.
manejoModelos:	Contiene las clases que son responsables de controlar los modelos que se encuentran activos en la vista.
modelo:	Módulo que agrupa la estructura de modelos propia de UML. Está conformado por el paquete que describe las interfaces definidas en la especificación de UML; las implementaciones de dichas interfaces (arcaKernelImpl) y las interfaces que sirven como estructura adaptadora hacia las capas de presentación y control de la herramienta (baseumlInterfaces)
ocl:	Paquete que contiene los servicios de construcción de restricciones OCL (precondiciones, poscondiciones, invariantes y derivaciones) asociadas a los modelos del usuario.
parametrosCliente:	Contiene las clases encargadas de leer y actualizar las variables de ambiente en el entorno de un cliente.
proyecto:	Contiene la clase que hace un manejo de los proyectos definidos dentro de un dominio.
transformación:	Paquete que contiene las clase para transformación a XMI, tanto importar como exportar (GeneradorXML), la generación a plantillas de documentación en html (GeneradorDocumentacionHTML) y la clase que maneja las reglas de transformación de los elementos del modelo (UMLTransformer)

usuario:	Paquete creado con la finalidad de manejar la información asociada a los usuarios de la herramienta con sus respectivos privilegios y roles
utilidadesDibujo:	Contiene las clases que ofrecen servicios generales de dibujo.
seguridad:	Provee servicios de seguridad tales como encriptación de cadenas de caracteres.

Tabla 3: Descripción de paquetes de AR2CA

4. TEMPLATES EN LAS APROXIMACIONES AOSD

En el núcleo de todos los enfoques AOD, es un requisito representar las relaciones transversales entre los modelos. Algunos enfoques basados en UML logran esto a través de la utilización de Templates. Los Templates permiten la creación de modelos de diseño abstractos. Un modelo es abstracto en el sentido en que razona sobre elementos abstractos que luego serán reemplazados por los elementos concretos dentro del modelo. El elemento abstracto es un parámetro que puede ser sustituido por un elemento concreto. Este reemplazo genera un nuevo diagrama donde el elemento concreto gana las relaciones, los atributos y las restricciones asociadas al parámetro que es substituido. Este es un mecanismo para representar *crosscutting*. Una relación *crosscutting* es en la cual un modelo altera una serie de modelos diferentes. Como tal los Templates soportan la especificación de las relaciones *crosscutting*. [8]

A continuación se presenta la representación de AOSD/UC y THEME con énfasis en el uso de los constructores Templates.

4.1. TEMPLATES EN LA APROXIMACIÓN AOSD/UC

En el diseño de alto nivel de AOSD/UC, los aspectos se especifican como casos de uso que extienden otros casos de uso, en un diseño más detallado los casos de uso se especifican como paquetes, dentro de estos paquetes se representa el diseño.

Los casos de uso Crosscutting contienen aspectos, un aspecto es un clasificador que se identifica con el estereotipo <<aspect>>, el cual se representa gráficamente como una caja que tiene dos compartimientos internos, uno contiene las declaraciones del pointcut y el otro contiene las clases de extensión o aspectos Hijos.

Los aspectos reutilizables o casos de uso de utilidad se especifican como paquetes parametrizables o Paquetes Templates, los parámetros del Template se nombran y se describen dentro de un cuadro con un borde punteado en el lado superior derecho de la plantilla del paquete. [9]

Una de las claves de aplicar AOSD está en su facilidad de parametrizar la definición de pointcuts con comodines y expresiones regulares. Los parámetros son usados para localizar y atar el aspecto en clases existentes y operaciones para ampliar el tema. Los Templates son frecuentemente usados como medio de generar clases y elementos.

- **Casos de uso slice parametrizandos.** Atreves del uso de Templates se aplican aspectos sobre operaciones múltiples mediante la parametrizacion de expresiones del pointcut. En UML, un elemento parametrizado es anotado por un cuadro punteado sobre su esquina superior derecha. Este cuadro punteado denominado *Signature* o firma contiene los parámetros para aquel elemento.

Para definir parámetros en un caso de uso Slice hay tres caminos [10]:

- Identificar el parámetro como un poincut y definir expresiones para el pointcut.
- Identificar el parámetro como un poincut y posponer la definición de la expresión para el pointcut a un aspecto hijo.
- Identificar el parámetro como un parámetro de Template.

En general, AOSD/UC usa una combinación de las tres técnicas. Esto se muestra en la Figura 12, donde se presenta el caso de uso slice Logging de forma parametrizada. El siguiente ejemplo es tomado de [10].

En la Figura 12, se muestran varios parámetros, cada uno delimitado por <>: roomAcessOperation, roomCall, RoomAccessor, logger y Resource. Los parámetros roomAcessOperation y roomCall son definidos en un poincut en la Figura 12. El parámetro RoomAccesor es definido por un poincut, Pero en este caso, se pospone su definición de la expresión pointcut a un aspecto hijo. El pointcut roomCall usa un parámetro de caso de uso slice Resource en la expresión, call(< Resource >. * (..)). Resource puede ser una serie de parámetros que serán substituidos cuando se aplique el Template. Logger es también otro parámetro de caso de uso slice. En este ejemplo, se quiere modelar el registro de petición de un cuarto en un hotel, entonces el parámetro Resouce sera sustituido por cualquier clase Cuarto. También se pude substituir el parámetro Logger con clases que permitan registrar a los usuarios en el momento de aplicar el caso de uso slice.

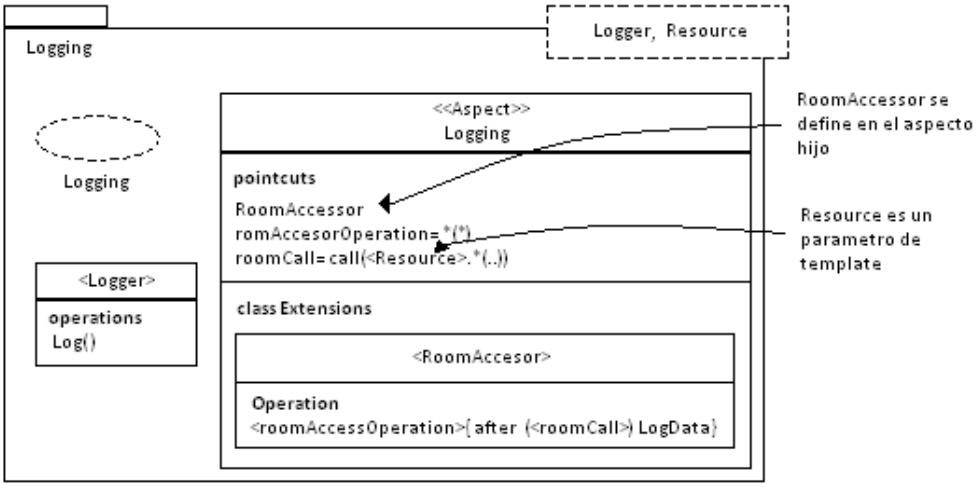

Figura 12: Representación de aspectos en de diseño detallado de AOSD/UC

4.2. TEMPLATES EN LA APROXIMACIÓN THEME

Siendo un lenguaje simétrico, Theme/UML soporta la representación de Themes base y Themes aspectuales, los Themes base y aspectuales se especifican en paquetes, los cuales se identifican como Themes por medio del estereotipo <<Theme>>, los Themes base y los Themes aspectuales contienen diagramas de clase que representan la estructura del aspecto, en estos diagramas de clases, la clase dentro del paquete Theme representa los conceptos de diseño que se requieren para realizar los requisitos relacionados con el Theme [9]. El siguiente ejemplo es tomado de [8].

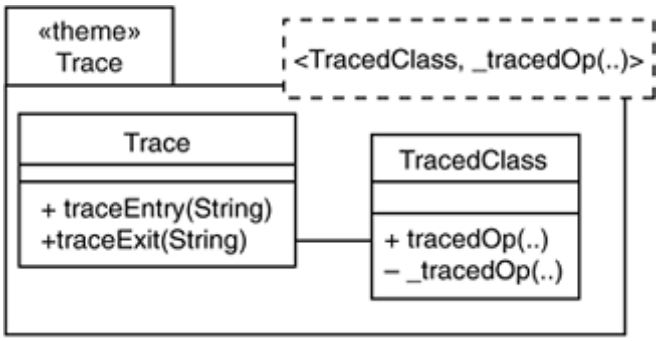

Figura 13: Theme Aspectual

Theme/UML soporta modelos de diseño como vistas independientes llamadas temas de diseño los cuales en el modelado se representan como paquetes UML estereotipados como <<Theme>>. Theme/UML soporta la representación de Themes base, que contienen el comportamiento normal del sistema y Themes aspectuales, que especifican el comportamiento transversal. Un Theme aspectual

difiere de un Theme base en que el primero es un paquete el cual tiene las características de un *TempleteableElement* y por lo tanto puede ser parametrizado para convertirse en un *Template*, la Figura 13, se muestra un ejemplo de un Theme aspectual, el cuadro punteado en la parte superior derecha del diagrama denominado firma corresponde al *Templatesignature* que aparece en el momento en que el paquete se parametriza, estos parámetros o *TemplateParameters*, representan los puntos de enlace que el Theme aspectual entrecruza. En un Theme aspectual se puede potencialmente tener múltiples clases patrón. Los servicios que contienen estas clases patrón se pueden usar como *TempletesParameter*. Las clases patrón son sustituidas por elementos que vienen del paquete Vinvulado de la forma como se determinan en la relación *Binding* expresada en la clase *Templat Binding*.

La Figura 13, ilustra un Theme aspectual con una clase patrón, TracedClass, la cual es especificada como *TemplateParameter* un servicio que le corresponde denotado _tracedOp() dentro del *Templatesignature* y denota que cualquier servicio puede ser sustituido por esta siempre y cuando cumpla con la restricción que se encuentra en el bind. En el diseño del Theme la clase patrón se encuentra como una clase estándar dentro del paquete parametrizado o *Template*.

Theme UML define una relación de composición para soportar la especificación de como *TemplateParameters* que se encuentran en los Themes aspectuales. UML define que un elemento que puede ser parametrizado en este caso el paquete puede contener una o varias relaciones de tipo Bind o *TemplateBinding* en tal caso este se convierte en un paquete vinculado, este *TemplateBinding* muestra la coneccion entre los parámetros de un *Template* o T*emplateParameters* y los elementos que será remplazados, dentro del paquete vinculado denominados TemplateParameterSubstitution. Sin embargo UML restringe el Bind en los TemplateParameters haciendo que la instanciación entre estos y los TemplateParameteSubstitution sea uno a uno, Theme UML extiende esta restricción de la notación UML incluyendo un accesorio bind[] y permitiendo que varios TemplateParameteSubstitution sustituyan un solo TemplateParameter. El orden de los parámetros contenidos en el bind[] marca el orden correspondiente en el cual se harán los remplazos en la clase patrón correspondiente. En caso de que un conjunto de elementos remplace un mismo parámetro dentro de un Templatesignature, este conjunto de parámetros serán enmarcan con llaves { }.

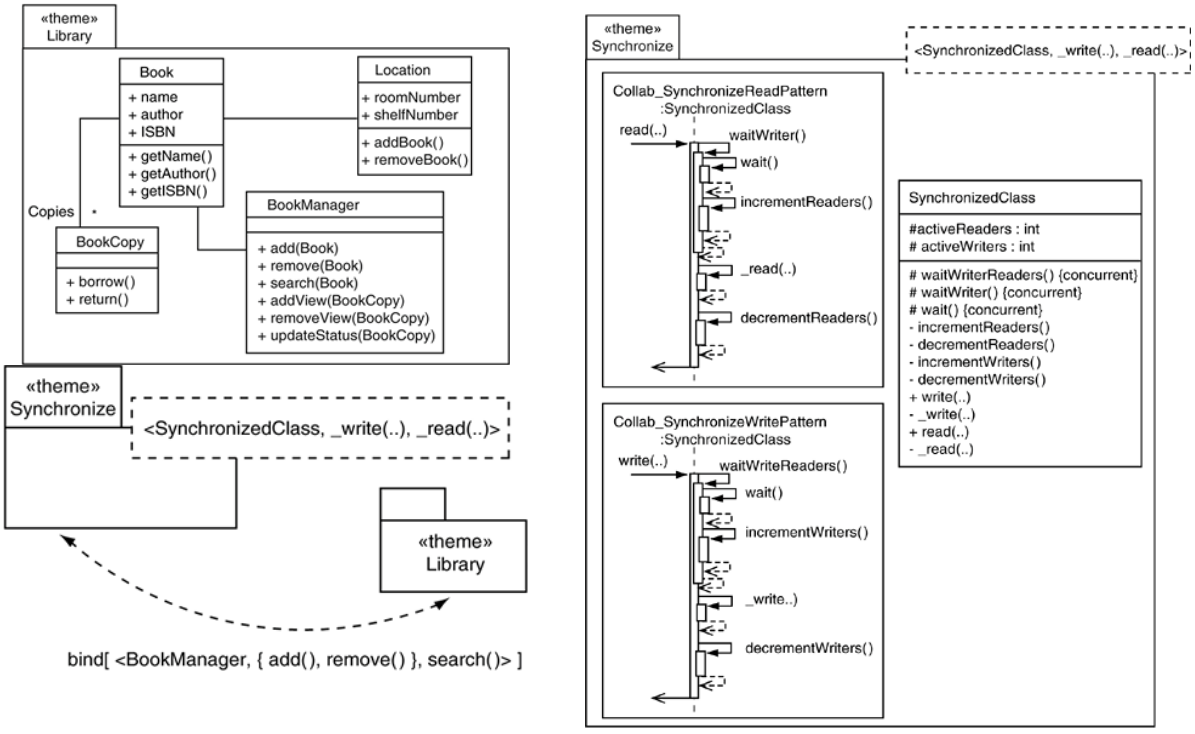

Figura 14: Modelado de aspectos con Theme

En el caso de la Figura 14, la clase BookManager que se encuentra en el paquete vinculado library quien al realizar la sustitución de parámetros determinada en el bind remplaza la clase patrón SynchronizedClass contenida dentro del *Template* Synchronize que representa el Theme aspectual de diseño, los métodos add() y remove() de la clase BookMananger son *TemplateParameterSubstitution* entrecruzados por el método write() que es un T*emplateParameter* de la clase Synchronize. De la misma forma la operación search() de BookManager se entrecruza con el método read() de Synchronize.

5. EL SOPORTE DE LAS HERRAMIENTAS CASE DE MERCADO PARA LA REPRESENTACIÓN DE TEMPLATES

Para establecer el soporte de las herramientas CASE en la implementación de Templates, se realizo un breve estudio de algunas herramientas CASE que son conocidas en el ambiente empresarial y las cuales soportan UML 2.0 o UML 2.1. El estudio únicamente se concentro en la representación de Templates en las herramientas con énfasis en la creación de paquetes parametrizables para la representación de aspectos. Las herramientas que se sometieron al estudio fueron Enterprise Architect versión 7.0, Magic Draw versión 15.5 beta 1, Microsoft office Visio 2007, y Visual paradigm from UML versión 6.2.

5.1. ENTERPRISE ARCHITECT 7

Es una herramienta CASE que cubre el desarrollo de software desde las recolección de requisitos pasando por análisis y modelado del sistema hasta llegar a la etapa de pruebas y mantenimiento. Esta versión de la herramienta toma como base la especificación UML 2.1.1 para definir la notación y semántica del dominio a modelar.

Para la representación de los modelos en las diferentes etapas de desarrollo de software Enterprise Architect 7 ha realizado la implementación de 13 diagramas definidos en UML 2.0 clasificados en 2 tipos:

- Diagramas de modelado estructural
  - Diagrama de paquete
  - Diagrama de clases
  - Diagrama de objetos
  - Diagrama de componentes
  - Diagrama de despliegue

- Diagramas de modelado de comportamiento
  - Diagrama de casos de uso
  - Diagrama de actividades
  - Diagrama de maquinas de estado
  - Diagrama de comunicación
  - Diagrama de secuencia
  - Diagrama de tiempos
  - Diagrama de interacción

5.1.1. Representación de templates en Enterprise Architect. Enterprise Architect soporta Templates únicamente como clases parametrizadas. Importa y genera clases parametrizadas para C + +. Los parámetros del Template se muestran en un cuadro punteado en la esquina superior derecha de la clase. Enterprise Architect no realiza la implementación de la relación de substitución de parámetros establecida en la especificación UML como bind.

Para crear una clase parametrizada, se siguen los siguientes pasos:

1. Abrá el cuadro de diálogo Propiedades de la clase haciendo doble clic sobre la clase.
2. Seleccione la pestaña de Detail.
3. En el campo Type, haga clic en la flecha hacia abajo y seleccione parameterised.

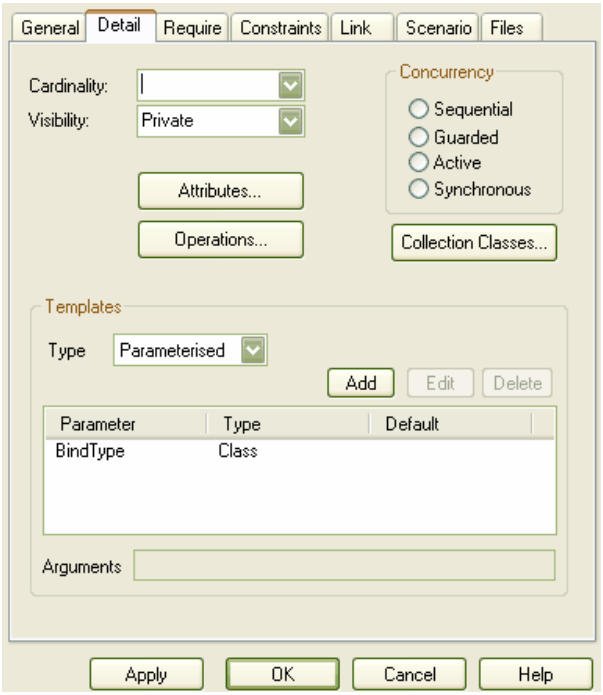

Figura 15: Dialogo de especificación de clases de Enterprise Architect

4. Haga clic en el botón add y luego clic en el botón ok.

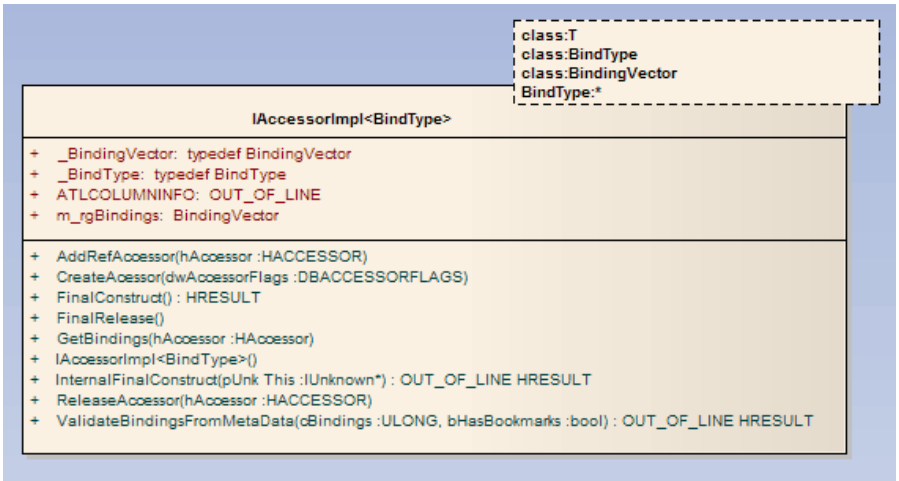

Figura 16: Clase parametrizada realizada en Enterprise Architect

5.2. MAGICDRAW 15.5

MagicDraw es una herramienta CASE de modelado UML y en esta versión implementa la especificación UML 2.1.1. Facilita el análisis y diseño de Orientado a Objetos. Los perfiles UML y diagramas personalizados le permiten ampliar el estándar UML para adaptarlo a su problema específico de dominio.

MagicDraw realiza una representación de los diagramas UML de la siguiente forma:

Diagramas UML	Representación en Magic Draw
Diagrama de casos de Uso	Diagrama de casos de uso
Diagrama de clases	Diagrama de clases
Diagrama de objetos	
Diagrama de máquina de estado	Diagrama de máquina de estado
Diagrama de actividades	Diagrama de actividades
Diagrama de secuencia	Diagrama de secuencia
Diagrama de comunicación	Diagrama de comunicación
Diagrama de componentes	Diagrama de implementación
Diagrama de despliegue	

Tabla 4: Diagramas UML implementados en MagicDrawn

5.2.1. Representación de templates en MagicDraw:

Magic Drawn soporta la implementación de Templates en los elementos que se especifican en UML 2.0 por ejemplo clases, operaciones, etc, sin embargo, únicamente los elementos de tipo clase realizan la representación gráfica de la

signatura y los parámetros de témplate. Para hacer la representación gráfica de los Templates en una clase se siguen los siguientes pasos:

1. Abra el cuadro de dialogo de especificación de la clase haciendo doble clic sobre ella, seleccione la opción Template Parameter y haga clic en el botón Create.

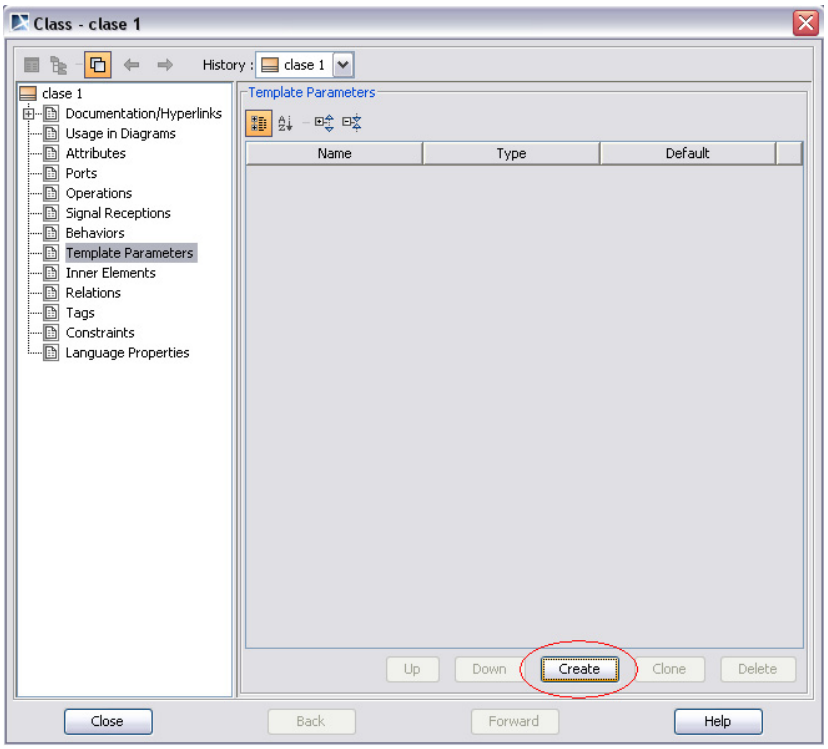

Figura 17: Dialogo de especificación de clases de MagicDraw

2. Seleccione el tipo que tendrá el parámetro de plantilla y se hace clic en ok.

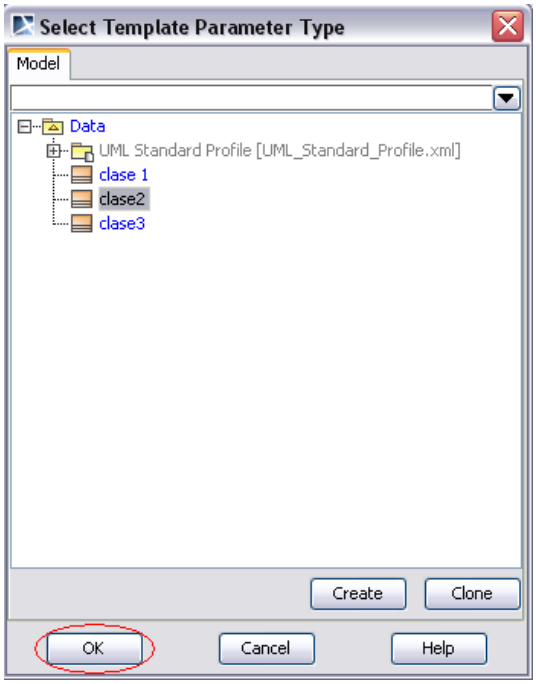

Figura 18: Dialogo de especificación de tipo de parámetro de MagicDraw

3. Escriba el nombre del parámetro y haga clic en el botón Close.

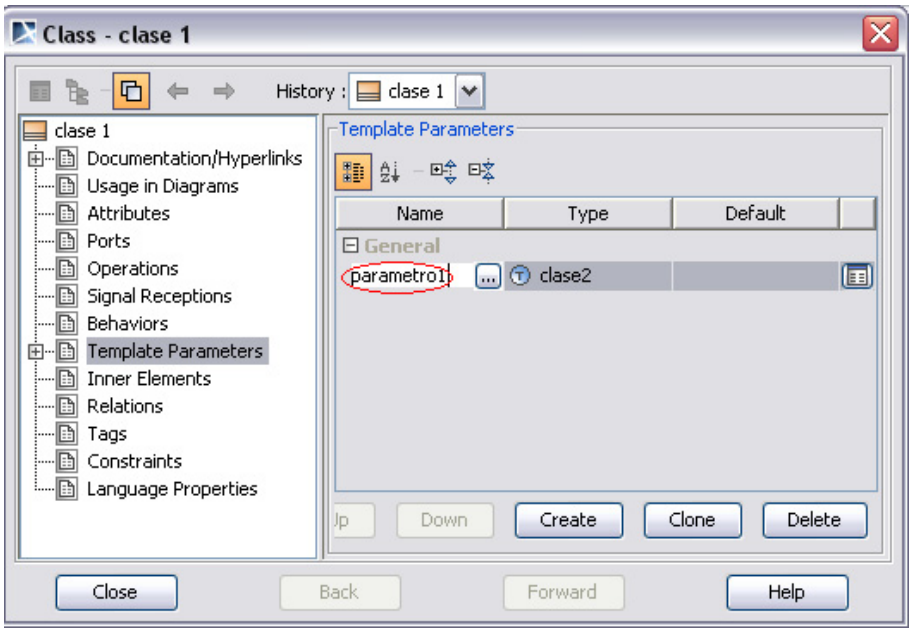

Figura 19: Dialogo de edición de parámetros de MagicDraw

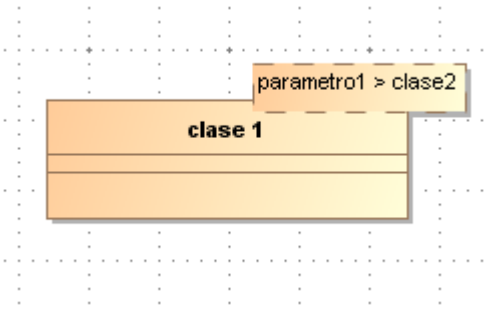

Figura 20: Clase parametrizada realizada en MagicDraw

Magic drawn implementa una relación Template binding pero no presenta la notación grafica de la relación, para generara una relación Template binding se siguen los siguientes pasos:

1. En el browser se hace clic derecho sobre la clase vinculada, luego se escoge la opción new Relation->Outgoing y se escoge la opción Template binding.

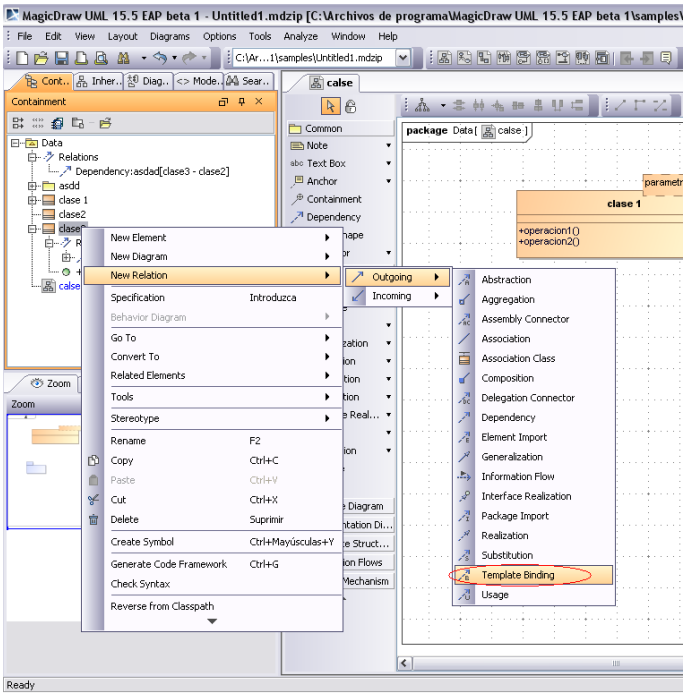

Figura 21: Menú de relaciones de MagicDraw

2. Aparece una ventana de selección de tipo, en al cual solo permite escoger elementos que estén contenidos en un signature.

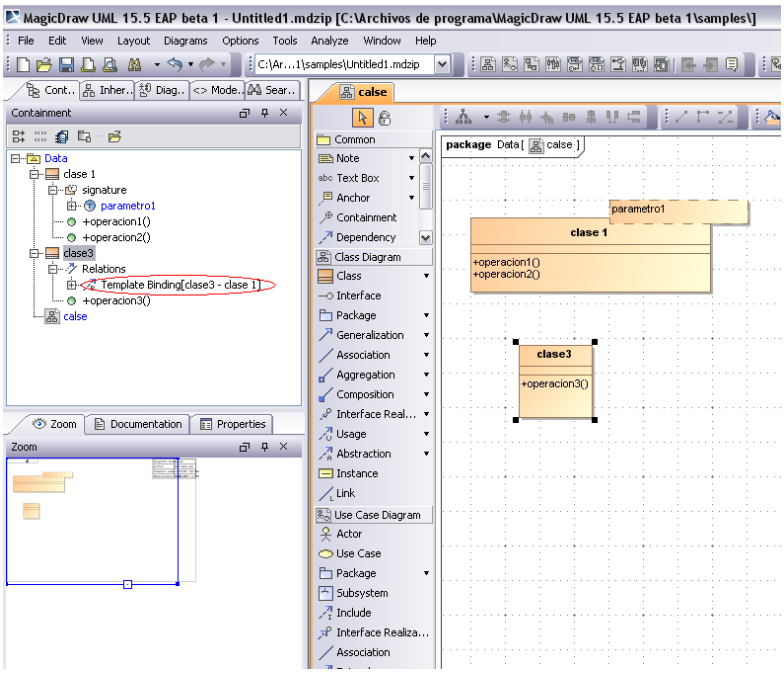

Figura 22: Presentación de Template Binding en browser de MagicDraw

5.3. MICROSOFT OFFICE VISIO 2007

Microsoft Office Visio 2007 es software de creación de dibujos y diagramas que facilita a los profesionales empresariales y de TI la visualización, el análisis y la comunicación de información compleja. No es una herramienta case especializada en desarrollo de software pero si contiene una base de diagramas UML de la versión 2.0 entre los que se encuentran:

- Diagramas de casos de uso
- Diagramas de estructura estática o diagrama de clases
- Diagramas de paquetes
- Diagramas de actividad
- Diagramas de estados
- Diagramas de secuencia
- Diagramas de colaboración
- Diagramas de componentes
- Diagramas de implementación

5.3.1. Representación de templates en Visio. Visio realiza la implementación de Templates en clases parametrizadas al igual que Enterprise Architect. Como parte de los elementos del diagrama de clases Visio especifica un elemento que

denomina clase parametrizada y para implementarlo únicamente requiere ser arrastrado a la hoja de dibujo.

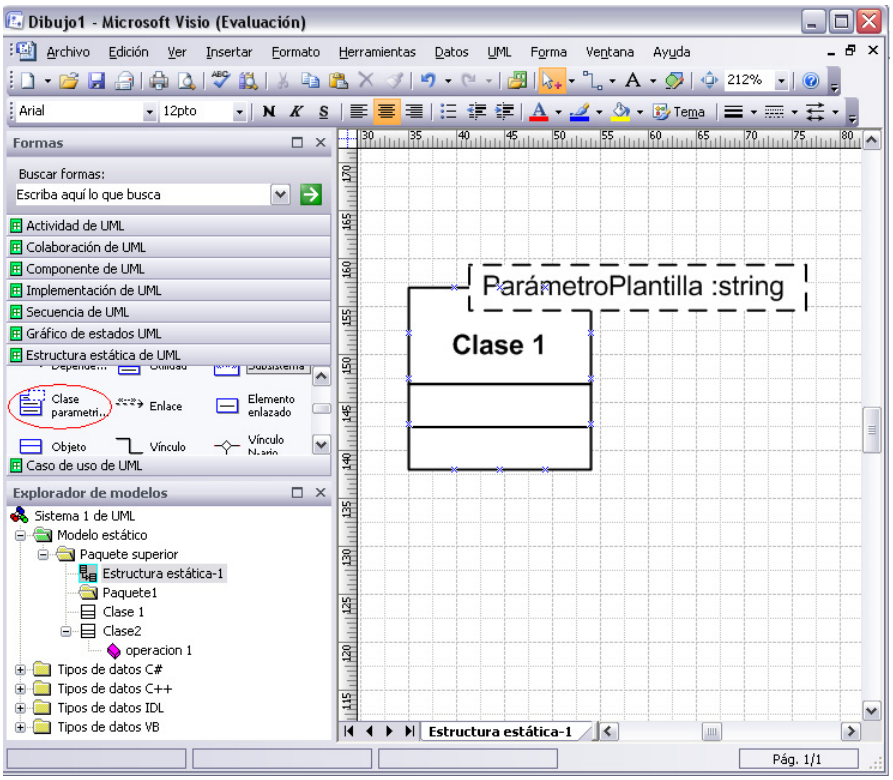

Figura 23: Clase parametrizada realizada en Microsoft Visio 2007

Después de esto se hace doble clic en la clase y se escoge la opción parámetros de plantilla donde se específica el nombre y el tipo del parámetro.

Visio realiza la implementación de la relación bind entre una clase parametrizada y una clase normal. Para realizar esta relación se realizan los siguientes pasos:

1. Se crean 2 clases de las cuales una debe tener parámetros de plantilla para realizar la especificación de estos parámetros a través de la relación bind.
2. Se arrastra la relación titulada enlace hasta la hoja de dibujo, uniéndola a las 2 clases.

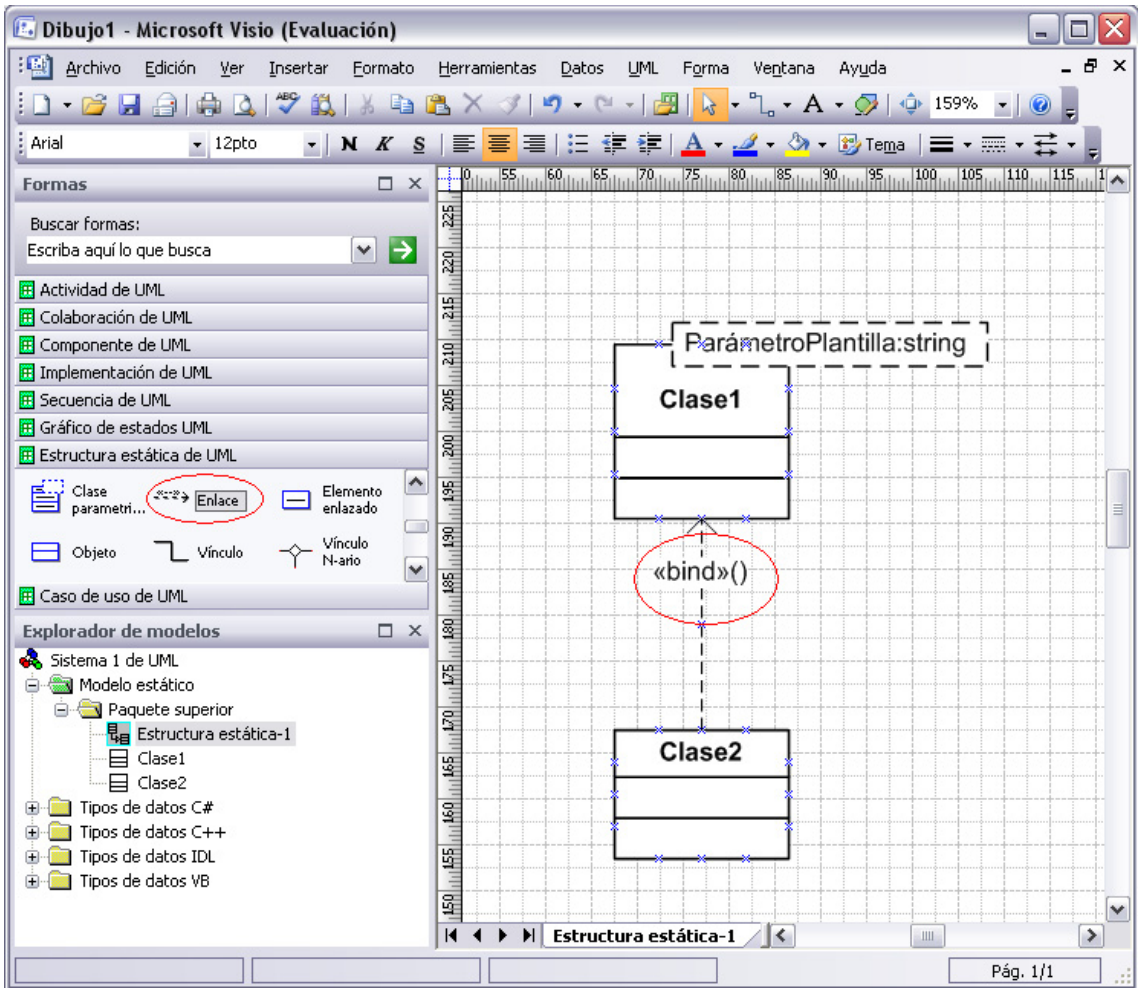

Figura 24: Representación de la relación Template Binding entre una clase parametrizada y una clase normal en Microsoft Visio 2007

Para establecer los parámetros de sustitución se hace doble clic sobre la relación y se escoge la opción argumentos limitados. En esta ventana parecen los parámetros de plantilla especificados en la clase parametrizada. En esta ventana se escoge el parámetro que se va a sustituir y se hace clic en propiedades.

3. Aparece una pantalla donde se escribe el valor por el cual el parámetro será sustituido.
4. Por último se hace clic en el botón aceptar.

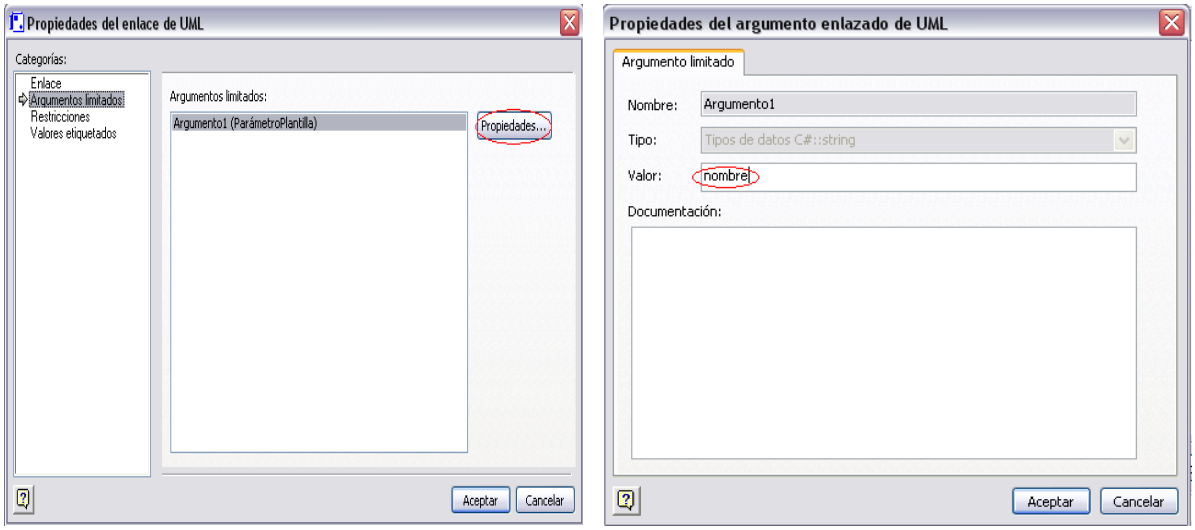

Figura 25: Ventanas de representación de parámetros de sustitución en la relación Template Binding entre una clase parametrizada y una clase normal en Microsoft Visio 2007

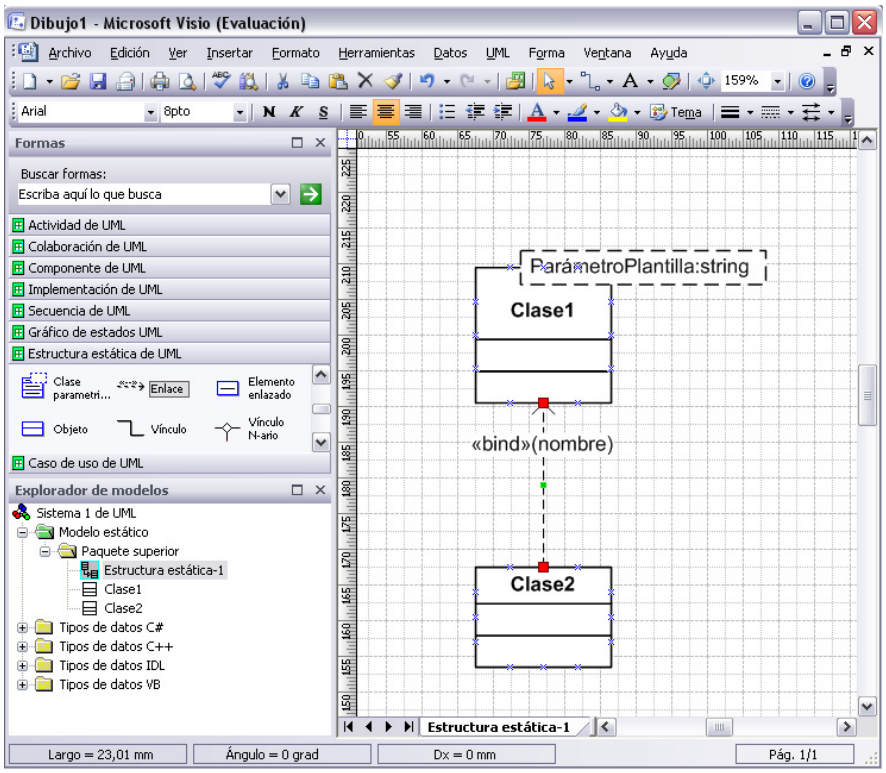

Figura 26: Representación de clases parametrizadas y relación bind en Microsoft Visio 2007

5.4. VISUAL PARADIGM FROM UML 6.2

Visual paradigm es una herramienta case que apoya todo el proceso de desarrollo de software orientado a objetos, para esto cumple el estándar UML realizando la implementación de sus diagramas acorde a la especificación 2.1. Visual paradigm realiza la implementación de los siguientes diagramas.

- Diagrama de paquete
- Diagrama de clases
- Diagrama de objetos
- Diagrama de componentes
- Diagrama de despliegue
- Diagrama de casos de uso
- Diagrama de actividades
- Diagrama de maquinas de estado
- Diagrama de comunicación
- Diagrama de secuencia
- Diagrama de tiempos
- Diagrama de interacción

5.4.1. Representación de templates en Visual Paradigm. Visual paradigm permite la implementación de Templates en todos los elementos descritos en la especificación UML 2.1, además de la relación bind descrita en la misma especificación. A continuación se muestran los pasos a seguir en la herramienta para realizar un paquete parametrizado y una relación bind.

1. En el diagrama de clases se crea un paquete sobre el cual doble clic para abrir la pantalla de especificación, en esta pantalla se escoge la pestaña Template Parameters en la cual se permite ingresar los parámetros de Template con su respectivo nombre, tipo de dato y valor por defecto.

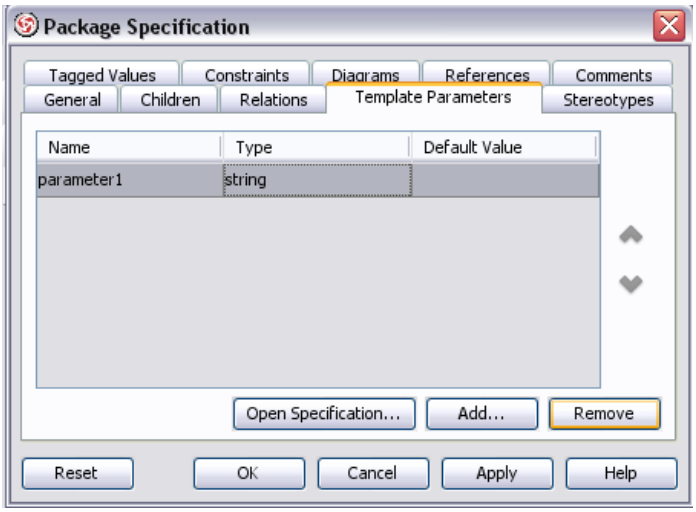

Figura 27: Dialogo de edición clase de Visual Paradigm

2. Al especificar el parámetro de forma manual se hace clic en el botón ok.

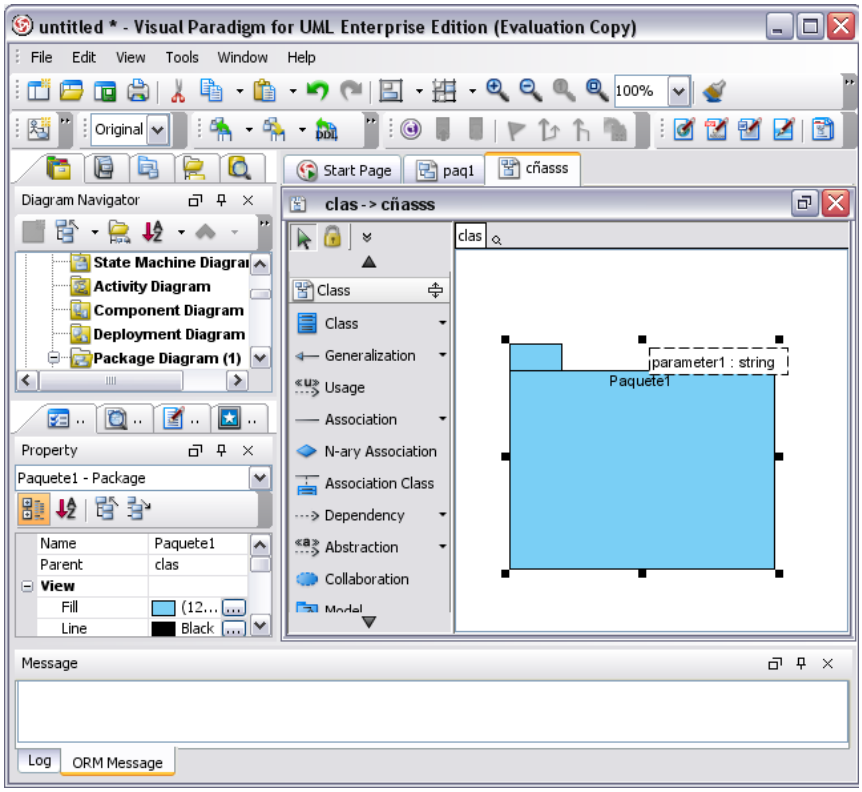

Figura 28: Paquete parametrizado realizado en visual Paradig

3. Se crea en el diagrama un nuevo paquete el cual se va a vincular al paquete parametrizado, después de esto se traza una línea entre los paquetes usando el botón derecho del mouse.

4. Aparece a continuación un menú desplegable en el cual se escoge la opción Binding Dependency

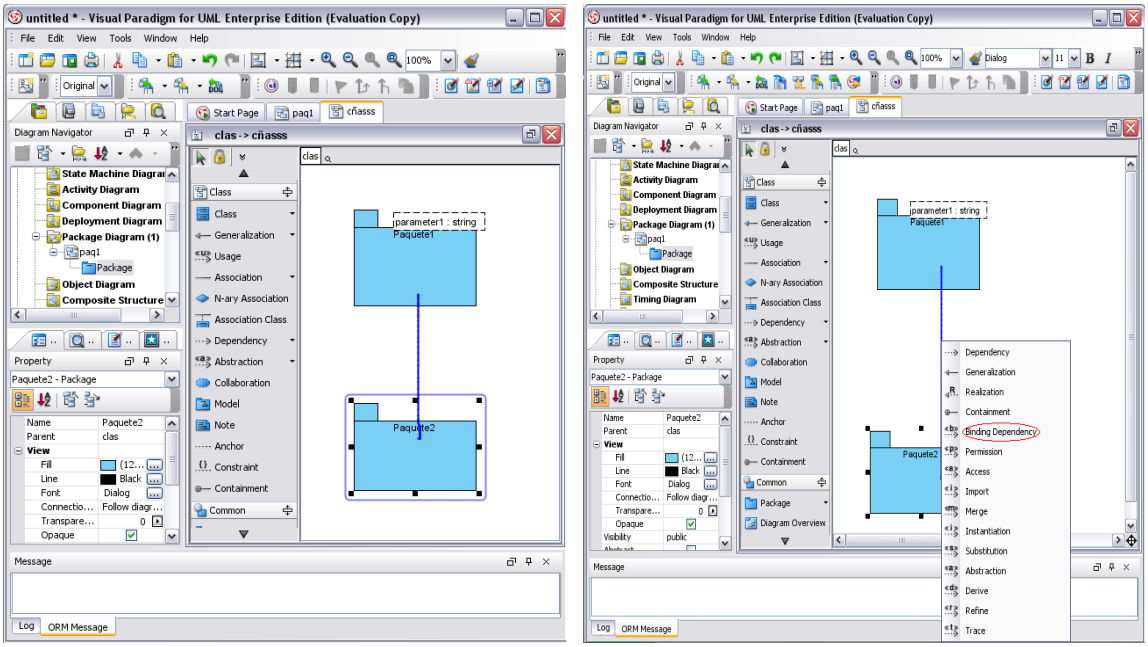

Figura 29: Creación de relación bind entre un paquete parametrizado y un paquete normal en Visual Paradigm

5. Aparece la relación bind y un campo de texto en el cual se pueden escribir los parámetros de sustitución.

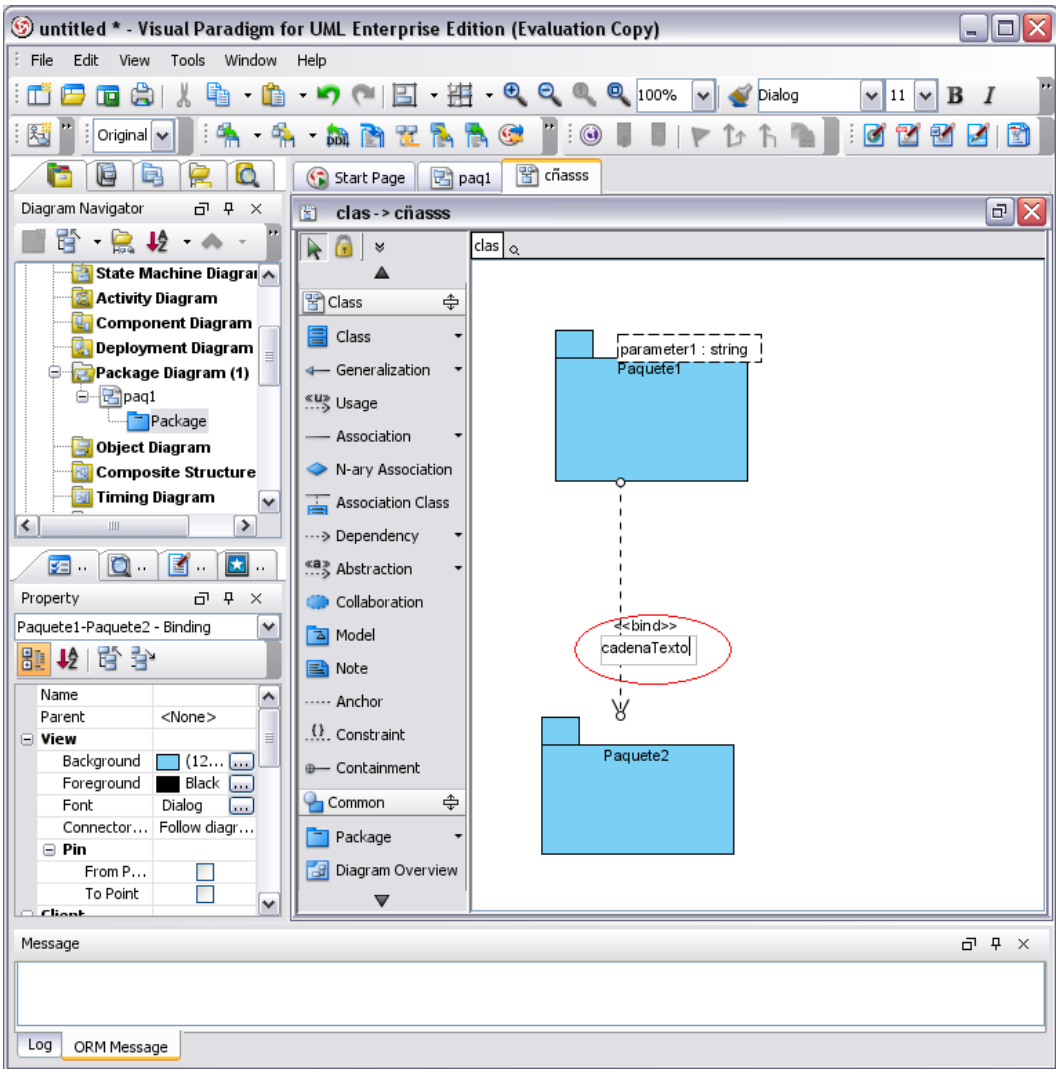

Figura 30: Representación de paquetes parametrizados y relación bind en Visual Paradigm

5.5. Resultado del estudio las herramientas:

Las herramientas se evaluaron con el fin de determinar hasta qué punto las herramientas poseen una implementación los Templates de la especificación de UML, y cuán fácil es para los diseñadores usar estas herramientas.

5.5.1. Implementación de la especificación UML

- **Implementación de los constructores Templates.** Las herramientas CASE evaluadas ofrecen el soporte de UML dirigido al desarrollo de software orientado a objetos, por esta razón la implementación de los constructores Templates se hace

en la mayoría de los casos a nivel de las clases. Solo en una de las herramientas evaluadas la parametrización se aplico a los paquetes.

- **Implementación de la relación bind.** La relación bind aunque se encuentra establecida en la especificación como una relación que realiza la sustitución de los parámetros no se encuentra de manera efectiva en ninguna de las herramientas.

5.5.2. Facilidad de uso de las herramientas. La creación de los Templates ya sea de clases o paquetes se hace estableciendo los parámetros de plantilla, en todos los casos los parámetros se adicionan como propiedades del elemento, sin embargo a pesar de encontrarse estas propiedades de una forma intuitiva la creación de los parámetros no tiene restricciones y se torna confusa para los usuarios.

La relación bind no posee buen soporte en las herramientas ya que es muy difícil encontrar la relación para aplicarla en los diagramas y no posee un soporte para la sustitución de parámetros que colabore al desarrollador en la sustitución de parámetros.

6. IMPLEMENTACIÓN DE TEMPLATES EN AR2CA

Con la necesidad de representar el comportamiento transversal, el uso de los elementos parametrizables se ha hecho visible en muchas aproximaciones AOSD, de estas como se ha mostrado en este documento la aproximación Theme hace un uso muy detallado y ligado a las características de la especificación de Templates aplicadas sobre los paquetes o contenedores también especificados por UML, con estos paquetes Theme hacer su representación de los modelos de Themes base y Themes aspectuales, por esta razón la implementación de Templates en AR2CA se vasa en la representación de Templates de Theme/UML.

Los elementos de tipo clasificadores, paquetes, u operaciones definidos en UML forman parte de los elementos que generalizan de la clase *TemplateableElement*, esta clase hereda a su descendencia características que les permiten contener una signatura para tornarse parametrizable y contener relaciones bind que permiten realizar el remplazo de los de los parámetros de la signatura.

Para dar a AR2CA la capacidad representar Themes se adicionaron algunas funcionalidades a las capas que componen su arquitectura.

6.1 CAMBIOS EN LA LÓGICA DEL MODELO DE AR2CA

La capa de modelo está representada por la implementación de un paquete que contiene las clases especificadas en UML 2.0, este paquete fue adaptado de una implementación de grupo desarrollador de Eclipse. [13] Esta capa contiene en el paquete arcaKernelImpl la implementación de las clases necesarias para el modelo UML, para la implementación de los Themes en AR2CA se usaron 5 de las 6 clases que se detallan en la especificación de UML sobre los constructores Template, además de la clase que contiene la implementación de los paquetes de la misma especificación, las clases se listan a continuación:

- PackageImpl
- TempletableElementImpl
- TemplateBindingImpl
- TemplateParameterSubstitutionImpl
- TemplateParameterImpl
- TemplatesignaturImpl

En las 5 clases correspondientes a la especificación de UML sobre los constructores Template se implemento la funcionalidad detallada en la

especificación con el fin de adaptar el estándar establecido. En el caso de los Themes la parametrizacion solo se realiza sobre los paquetes, para cumplir con este requerimiento en AR2CA, se realiza la implementación de Templates a nivel de los paquetes, sobre estos se realizaron las extensiones necesarias para la representación de Themes. La clase PackageImpl que contiene el comportamiento de los paquetes o contenedores generaliza de la clase ParameterableElement y de esta hereda las características de ser elemento parametrizable.

6.2 CAMBIOS EN LA LÓGICA DE ADAPTACIÓN DE AR2CA

La capa adaptadora del modelo ha sido definida con el propósito de facilitar la migración hacia la versión UML 2.0. Esta capa contiene un conjunto de interfaces relacionadas con la definición de los servicios que son utilizados por las capas superiores. Tiene como finalidad desacoplar los servicios requeridos por las capas superiores de la arquitectura de la especificación concreta de UML que se esté utilizando. [5]

En el paquete baseUMLInterfaces se encuentran las interfaces que sirven como adaptadoras entre la capa de modelo y las capas de control y presentación, para tener acceso a las funcionalidades que permiten a AR2CA modelar Themes, se crearon en este paquete las siguientes interfaces.

- ElementoParametrizable
- ParametroSustitucion
- RelacionEnlace
- Paquete

Estas in interfaces contienen los servicios necesarios para la representación de Themes en AR2CA, su implementación se realiza en algunas clases del paquete de modelado arcaKernelImpl de la siguiente forma:

INTERFACE	CLASE IMPLEMENTACION
ElementoParametrizable	*Template*ParameterImpl
ParametroSubstitucion	*TemplateParameterSubstitutionImpl*
RelacionEnlace	*TemplateBinding*Impl
Paquete	*PackageImpl*

Tabla 5: Implementación de Interfaces en AR2CA

En la interfaz Paquete se adicionaron métodos que permiten al paquete tener las características para representar un Theme, estos métodos se encargan de crear la signatura y adicionar o eliminar los parámetros dentro de la signatura, además

para la creación de Themes base se crea un método que le permite al paquete contener la relación bind.

En la interfaz ElementoParametrizable se adicionan los métodos que permiten a un elemento como una clase o un servicio ser parámetro para luego contenerse en una signatura.

La interfaz RelaciónEnlace brinda los servicios que permiten crear la relación bind. Esta interfaz solicita la declaración del paquete que la va a contener el cual se conoce como elemento vinculado y también el conjunto de parámetros de sustitución que contendrá la relación bind para realizar los remplazos de los parámetros en el Template.

Y por último la interfaz parámetro sustitución contiene los servicios que se ofrecen para controlar el parámetro formal de la sustitución y los parámetros actuales con los cuales se va a realizar la sustitución de el parámetro formal.

A través de los servicios que prestan estas interfaces se puede comunicar las diferentes capas para realizar el modelo del diagrama.

Todas estas características se aplican sobre los paquetes o contenedores propios del diagrama de clases, por esta razón no se requiere generar un nuevo diagrama. De esta forma el diagrama de clases de AR2CA se extiende y adque las características necesarias para realizar el modelado de aspectos con Themes.

A continuación se presenta en la Figura 31 un diagrama de clase que establece las relaciones entre las clases y las interfaces realizadas en AR2CA para el modelado de Themes.

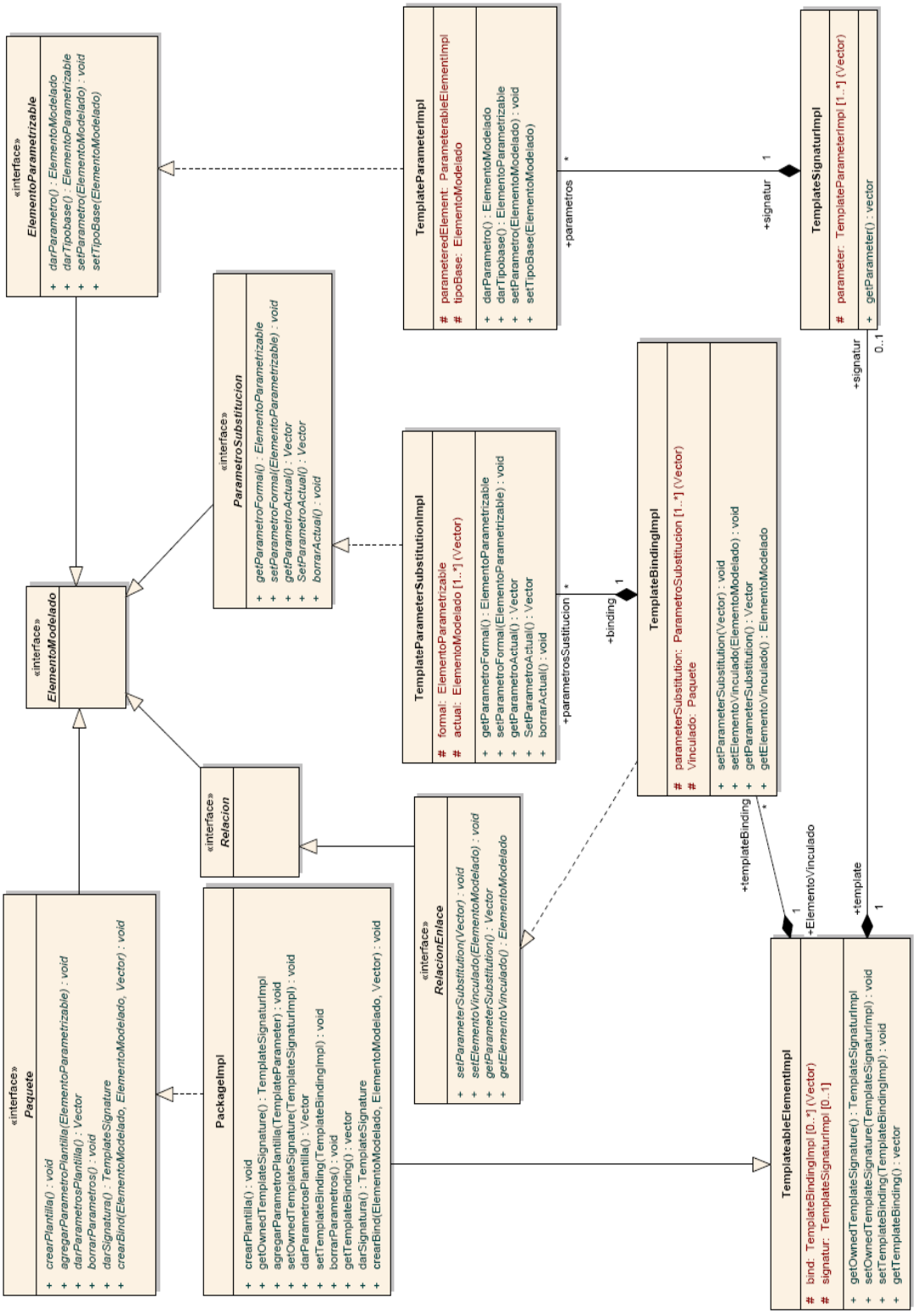

Figura 31: Clases e interfaces usadas en AR2CA para la representación de Themes

6.3 CAMBIOS EN LA LÓGICA DE PRESENTACIÓN DE AR2CA

En la capa de presentación se encuentra la clase DibujoCanvas, la cual es la encargada de representar el modelo UML en la capa de presentación. Esta clase representa un grafo conformado por los elementos nodales (ElementoDibujo) y lineales (ConexionDibujo). Cada elemento nodal o lineal tiene asociada su respectiva clase de tipo Figura que es la encargada de implementar los servicios gráficos para dibujar el elemento. [5] En la Figura 32 podemos observar el diagrama de clases que se usan para la representación de Themes en AR2CA.

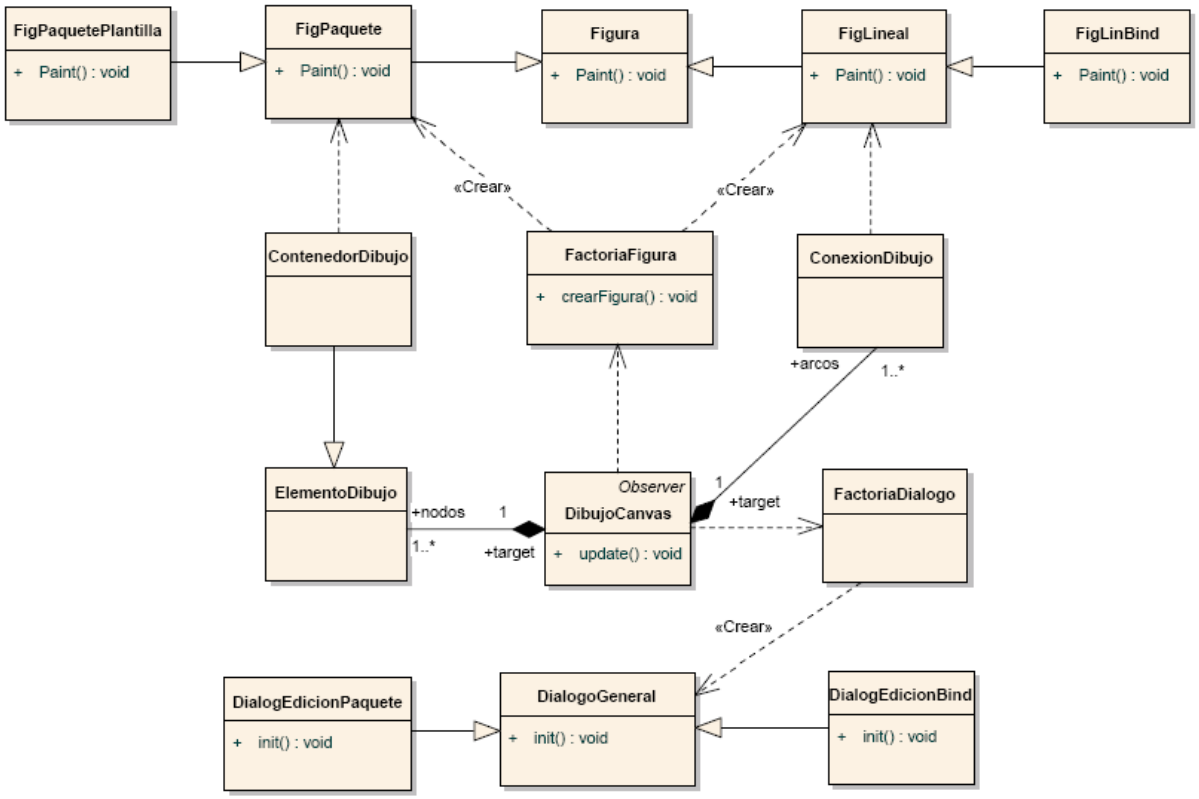

Figura 32: Clases de la capa de presentación que se encargan de la representación de Themes en AR2CA

Para la representación de Themes en AR2CA se extendieron los servicios de la clase ContenedorDibujo, esta clase extiende de la clase ElementoDibujo y contiene los servicios que permiten controlar la interacción de los usuarios al ejecutar acciones con los paquetes desde la clase DibujoCanvas.

La clase ContenedorDibujo está asociada a la Clase FigPaquete la cual se encarga de implementar los servicios gráficos de un paquete, en caso de que el paquete se convierta en Template los servicios gráficos se implementan desde la

clase FigPaquetePlantilla la cual tiene la capacidad de dibujar la signatura del paquete. Para introducir los parámetros del paquete la clase DibujoCanvas llama a la clase DialogEdicionPaquete la cual genera una Ventana que permite introducir los parámetros de la signatura. (Ver figura 33)

La clase ConexionDibujo controla la interacción de los usuarios con las relaciones que se pueden dar entre los elementos del diagrama, en el caso de la relación Bind se asocia a esta clase la clase FigLinBind que contiene los servicios gráficos que corresponden a las características de la relación, para ingresar los parámetros de sustitución en esta relación se hace uso de la clase DialogEdicionBind la cual genera una ventana que permite ingresar parámetros tanto por elementos que se encuentren el el diagrama como por una exprecion sensilla parámetros. (Ver figura 34 y 35)

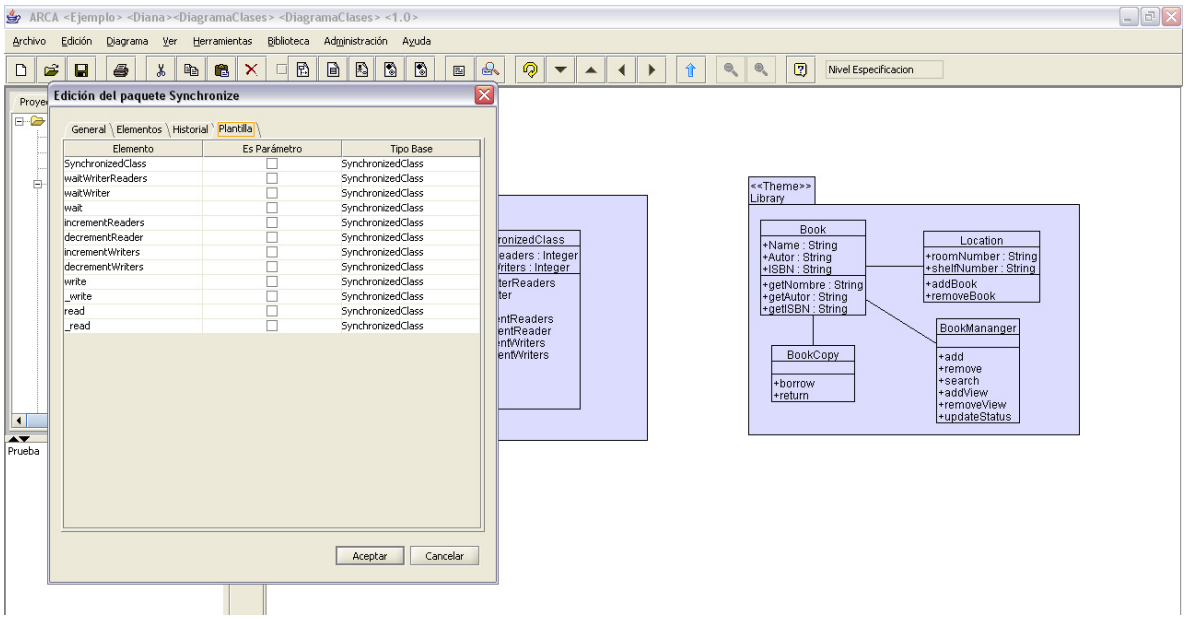

Figura 33: Ventana de creación de parámetros de paquete en AR2CA

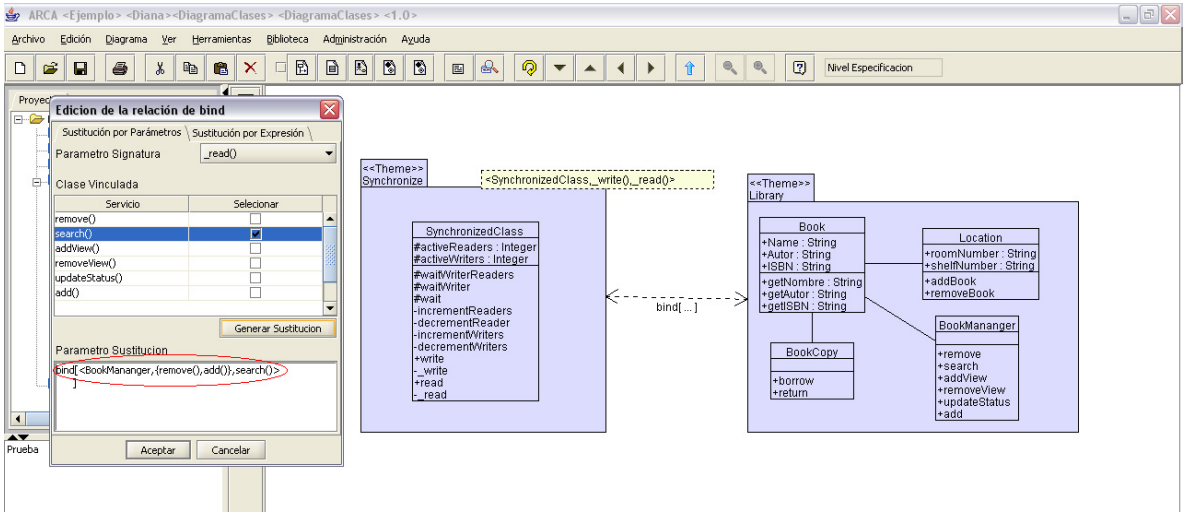

Figura 34: Sustitución de parámetros por elementos del diagrama en AR2CA

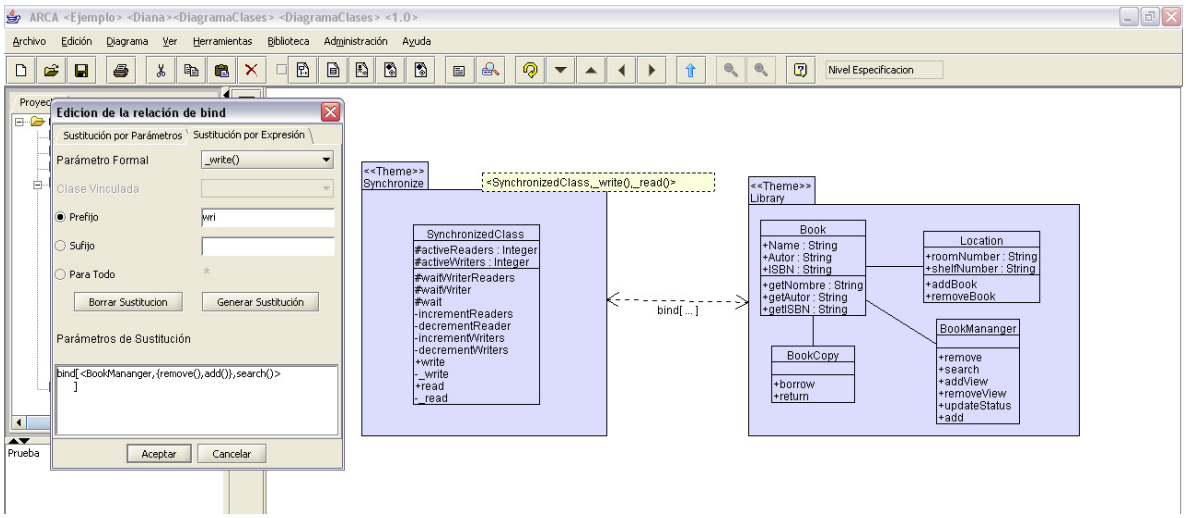

Figura 35: Sustitución de parámetros por expresión en AR2CA

6.4 CAMBIOS EN LA LÓGICA DE PERSISTENCIA AR2CA

La lógica de persistencia no presenta cambios en estructura, la clase *ArcaFachadaImplementacion*, es la encargada de manejar los servicios que van a ser ejecutados contra el almacenamiento persistente. Existe un servicio de factoría el cual se encarga de definir por medio de parámetros configurables cual es la clase concreta que se encarga de la persistencia. Las clases encargadas de la persistencia forman una estructura de herencia en la que la superclase (*BaseDaoXML*) realiza operaciones comunes para inicializar el documento XML con las propiedades comunes de cualquier elemento del modelo (*agregarPropiedadesComunes*) o para leer del documento XML las propiedades comunes (*subirPropiedadesComunes*). Esta clase a su vez invoca por medio de reflexión la clase específica encargada de generar el documento XML *getJDOMElement)* y de subir las propiedades particulares de un elemento concreto (*deserializarElementos*) [5].

Para realizar la persistencia de los Themes a la clase PaqueteDaoXML la cual generaliza de BaseDaoXML se le adicionaron servicios que le permiten escribir en los archivos XML tanto los parámetros del Template como la relación Bind y sus componentes.

CONCLUSIONES

Este trabajo cumplió con los objetivos inicialmente establecidos. El objetivo específico 1 se cumplió al realizar los capítulos 3 y 4 de este documento, el objetivo específico 2 se cumplió en el desarrollo del capítulo 5 de este documento, y los objetivos específicos 3 y 4 se cumplen en el apartado 6 de este documento y se presenta como resultado la herramienta CASE AR2CA con las modificaciones necesarias para el diseño de Themes.

El desarrollo de software orientado por aspectos es una evolución del desarrollo de software orientado por objetos que evita los fenómenos de diseminación y mesclado de software. Para que este enfoque funcione se requiere que los diseñadores de software asuman una cultura de modelado para que puedan determinar el comportamiento trasversal del sistema.

Uno de las grandes dificultades para establecer la orientación por aspectos en las empresas desarrolladoras de software es resistencia al cambio y la falta de actualización de los desarrolladores en cuanto al conocimiento de nuevas tecnologías.

El lenguaje de desarrollo unificado UML está en constante cambio debido a los nuevos requerimientos de los desarrolladores de software, sin embargo de nada sirven estos avances si las herramientas CASE que ofrecen ayuda en el proceso de desarrollo de software no realizan las nuevas implementaciones de la especificación.

Debido a la gran robustez de lenguaje de modelado unificado UML es muy difícil para los desarrolladores de herramientas CASE orientadas al apoyo en el desarrollo de software implementar todas las características que la implementación establece, sin embargo también debido a su gran facilidad de extensión y modularizacion UML permite implementar las características necesarias de esta forma las herramientas CASE pueden especializarse para lograr un objetivo especifico.

El estudio e implementación de los Templates representó la parte central del trabajo. Las principales conclusiones alrededor de este tema fueron las siguientes:

- Por sus características, entre las cuales está la facilidad de remplazar algunos parámetros en el momento de su especificación, los Templates de la especificación 2.0 de UML, son de gran ayuda para establecer una notación de diseño orientada por aspectos.

- Aunque los Templates se encuentran dentro de la especificación de UML por ser constructores auxiliares las Herramientas CASE no realizan una implementación completa de sus características y en la mayoría de los casos se limitan a hacer una implementación de clases parametrizables las cuales son de uso más común para el desarrollo de software orientado por objetos.

- Para la representación de aspectos usando la notación de la aproximación de Theme fue necesario realizar la implementación de Templates sobre los paquetes también descritos en la especificación de UML que se encuentra en el paquete UML de Eclipse y que forma parte del núcleo de esta herramienta CASE.

- Asimismo fue necesario extender la funcionalidad gráfica de la herramienta para cumplir con los requisitos gráficos de los Themes.

- Los Templates ofrecen un gran apoyo al diseño de aspectos debido a que los parámetros determinan de forma fácil los puntos de intervención de los elementos aspectuales y permiten separar de forma visible las operaciones básicas del sistema de las operaciones trasverzales.

La facilidad de extender una herramienta CASE orientada al desarrollo de software depende mucho de la comprensión de la arquitectura de la herramienta, del uso de estándares de desarrollo y del entendimiento de la especificación UML. Por esta razón fue necesario un estudio profundo de las características de la herramienta, y de la especificación de UML.

Fue una experiencia enriquecedora en mi formación profesional el realizar adaptaciones en la herramienta CASE AR2CA. Aspectos como el entendimiento de su arquitectura, la experimentación con patrones ya implementados, el manejo de herramientas de apoyo al desarrollo, como la facilidad de debug y los servicios de gestión de configuraciones (CVS) fortalecieron mis competencias como desarrolladora de software aplicando buenas prácticas.

Con la realización de este trabajo se comprendió la dinámica de trabajo de un grupo de investigación, lo cual contribuye en el mejoramiento de la formación académica y profesional. Se reconoció la importancia del trabajo en equipo, y de compartir el conocimiento adquirido, además se contribuyo de manera efectiva en la dinamización de las relaciones del grupo y en la creación de un clima ameno de trabajo.

TRABAJOS FUTUROS

Dado que la herramienta CASE AR2CA contiene un módulo de generación de documentación en formato HTML un trabajo futuro seria modificar este módulo para que emita la documentación HTML de los Themes de diseño.

Otro trabajo futuro seria continuar la implementación de Templates para otros constructores como Clases y Operaciones de forma que se cumpla por completo la especificación UML sobre los Templates

REFERENCIAS BIBLIOGRÁFICAS

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

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

[3] González, C. Murillo, J. Amaya, P. Un modelo de propiedades y dependencias para el análisis orientado a aspectos en MDA. Quercus Software Engineering Group. Departamento de Informática. Universidad de Extremadura. 2004

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

[5] Filman, R E. Elrad, T. Clarke, S. Aksit, M. Aspect-Oriented Software Development. Addison-Wesley Professional. 2004

[6] Pressman, R. S. Ingeniería del Software un enfoque práctico. McGraw-Hill Interamericana. 2005.

[7] Brichau, J. Theo D'Hondt, "Introduction to Aspect-Oriented Software Development", AOSD-EUROPE.

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

[9] Barón, A. Análisis comparativo AOSD/UC y THEME. Reporte técnico, Grupo de Investigación de Ingeniería de software universidad EAFIT, 2007.

[10] Jacobson, I. Ng, P-W. Aspect-Oriented Software Development with Use Cases. Addison Wesley Professional, 2005.

[11] Baniassad, E., Clarke, S.: Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley (2005)

[12] Blanco, G. Vásquez, J. Anaya, R. Manual de Usuario de AR2CA 3.0, Manual de Referencia, Versión 0.1. Universidad EAFIT. Grupo de Investigación de Ingeniería de Software. 2007

[13] Anaya, R. Vásquez, J. Documento de Arquitectura de AR2CA 3.0, Manual de Referencia Técnico, Versión 0.8. Universidad EAFIT. Grupo de Investigación de Ingeniería de Software. 2006

[14] Herrmann, S. Composable Designs with UFA.Technical University Berlin.Germany. 2002

[15] Gray, J. Bapty, T. Neema,S. Gokhale, A. Aspect-Oriented Domain-Specific Modeling.Institute for Software Integrated Systems (ISIS).Vanderbilt University, Nashville TN. 2002

[16] Jacobson, I. Chirsterson, M. Jonsson, P. Overgaard, G. Object-Oriented Software Engineering: A Use Case Driven Approach, 4 ed: Addison-Wesley, 1992.

[17] Baron, A. Desarrollo de Software Orientado por Aspectos con Casos de Uso –AOSD/UC. Reporte técnico, Grupo de investigación de ingeniería de software, Universidad EAFIT, 2008.