Tesis de Licenciatura

AODW
Un Metamodelo Multidimensional Orientado a Aspectos para el Diseño de Datawarehouses

Alumnos : **Andrea Venegas, Janina Rubacha**
Director : **Eduardo Rodríguez**
Fecha : **Octubre de 2006**

Departamento de Computación
Facultad de Ciencias Exactas
Universidad de Buenos Aires

Resumen

A pesar de que el Datawarehousing ha recorrido una larga trayectoria desde sus comienzos a mediados de los 90, sigue adoleciendo de la falta de un modelo conceptual generalizado que se diferencie y distancie de las implementaciones lógicas y/o físicas dominantes hasta el momento así como de una clara posición con respecto al manejo y modelado de seguridad en bases de datos multidimensionales.

En este trabajo se presenta un nuevo meta modelo multidimensional completo, auto contenido y basado en estándares que permite modelar también la aplicación de la seguridad. Para ello se utilizaron técnicas de modelado orientado a objetos para el meta modelo conceptual de datawarehousing y orientación a aspectos para completar el modelado de la seguridad en dicho meta modelo.

Abstract

Despite the long way that Datawarehousing has thread since its beginnings in the nineties, it still lacks both a generalised conceptual model to isolate itself from the dominant logical and physical implementations and a definite position with regards to the handling and modelling of security in multidimensional database systems.

In this work a complete and self-contained multidimensional metamodel based on standards is introduced, which also allows modeling the security issues. Object oriented modelling techniques have been used for the conceptual datawarehousing metamodel and an aspect orientation approach to finalize the modelling of security within such metamodel.

Agradecimientos

Queremos dedicar esta tesis principalmente a las personas que vivieron con nosotras este largo camino que hemos recorrido, a las que no están y, en particular, a una pequeña personita que aportó, con su inocencia, la alegría necesaria en los interminables fines de semana de "Tesis" en este último y más arduo año: Tommy.

Nuestra idea comenzó allá a lo lejos, en un bar, junto a un grupo de alumnos que buscábamos delinear temas de tesis guiados por el Lic. Eduardo Rodríguez. De este grupo bastante heterogéneo, salimos nosotras dos. Una con varios años de experiencia en el mercado laboral, particularmente en el área de Datawarehousing, y la otra recién incursionando en sus primeros trabajos, con un amplio conocimiento en Orientación a Objetos.

Ambas, en nuestro paso por la Facultad, tuvimos cargos de ayudantes en cátedras como Arquitecturas y Sistemas Operativos e Ingeniería de Software I.

El trabajo de esta tesis comenzó dedicándole mucho tiempo a la investigación del estado del arte en lo que respecta a modelos multidimensionales en general y a los orientados a objetos en particular así como también todo lo referido al paradigma de orientación a aspectos, tema relativamente nuevo en ese momento.

Durante la evolución de esta tesis, presentamos dos trabajos: "Aspect Oriented Datawarehousing" en el "Taller de Trabajo en Desarrollo de Software Orientado a Aspectos, DSOA'03" en Alicante, España en Noviembre de 2003 y "OODW: Un Metamodelo Conceptual Orientado a Objetos para Bases de Datos Multidimensionales" en "IDEAS´06" en La Plata, Argentina en Abril de 2006. De ambas presentaciones obtuvimos recomendaciones importantes que hoy se encuentran plasmadas en este trabajo.

Lamentablemente la continuidad y la dedicación de esta tesis se vio interrumpida por responsabilidades laborales en el exterior de ambas, una vivió en Chile, la otra en Londres, y por la llegada de Tomás a este mundo. Si bien nos retrasó y bastante en nuestros planes de entrega y finalización de este trabajo, llegando a hacernos dudar si algún día íbamos a lograr terminarla, todas las experiencias fueron enriquecedoras y hoy podemos mirar hacia atrás con la sensación del deber cumplido.

Haciendo un balance de todo este tiempo podemos decir que resultó de gran aprendizaje para ambas que la realización de esta tesis no haya tenido resultados inmediatos sino que muy por el contrario nos tomara más tiempo del pensado ya que en ese tiempo entendimos que a pesar de tener que recorrer largos y sinuosos caminos de trabajo o de pasar momentos de no encontrar la llave mágica que abriera las puertas de las soluciones, la constancia y la perseverancia traen sus resultados. En fin: "costó pero se logró".

Desde ya, dirigimos nuestro especial agradecimiento a Eduardo Rodríguez, que a pesar de haberse ido a vivir al exterior nos siguió acompañando y guiando en todo este tiempo. También a Ariel Aizemberg, que nos revisó una de las versiones preliminares de la Tesis y nos alentó a presentarla en uno de los Congresos anteriormente mencionados, y no nos queremos olvidar de Adrián Anacleto que fue un crítico

excepcional y que nos ayudó a depurar la versión final de este trabajo. Por último, a la persona que nos ayudó a agilizar los trámites para esta defensa: la Dra. Irene Loiseau.

Por todo esto, queremos nuevamente dar las gracias a todo aquellos que nos han devuelto una sonrisa y que nos ayudaron a seguir adelante, a los que han puesto su parte para que le trajín de estos años fuera más llevadero. Gracias nuevamente a:

Eduardo, Ariel, Pablo, Adrián, Irene y a nuestros familiares Marta, Daniel, Exequiel, Carmen, Fernando, Claudio y Tommy.

Indice General

Resumen....2
Abstract....2
Agradecimientos....3
....4
Indice General....5
Indice de Figuras....6
Indice de tablas....8
SECCION I – Introducción....9
Motivación y objetivos....10
Organización de la tesis....13
Caso de estudio....16
SECCION II – Modelos multidimensionales....18
Marco teórico – Introducción a OLAP....19
Modelos multidimensionales tradicionales....21
Modelos multidimensionales OO....29
Beneficios de utilizar modelos multidimensionales OO....29
Propiedades deseables en un MMD conceptual....32
Evolución de los modelos multidimensionales OO....36
Rumbo al metamodelo....55
OODW - Un metamodelo multidimensional OO....60
El metamodelo de la dimensión....62
El metamodelo de las métricas....63
El metamodelo de los hechos....64
El metamodelo del cubo....65
Instanciación de OODW....67
Aplicaciones de OODW....71
Construcción de modelos multidimensionales....71
Consulta de modelos multidimensionales....72
SECCION III – Aplicación de AO a Seguridad en OLAP....75
Modelos de seguridad en datawarehousing....76
Control de acceso....76
Políticas de seguridad....78
Mecanismos de seguridad....78
Ejemplo de un modelo conceptual de seguridad multidimensional....79
Trabajos recientes....83
Orientación a aspectos – Una introducción....85
Motivación....85
Conceptos fundamentales....86
AOP aplicado al ciclo de vida del software....89
Comparación de las propuestas....101
AODW - Modelado de la seguridad en DW con aspectos....111
SECCION IV - Conclusiones y Trabajo Futuro....114
Conclusiones....115
Trabajo Futuro....116
Bibliografía....118
ANEXO A – Glosario....125
ANEXO B – Introducción a la POO....131
Conceptos de la POO....132
Acceso a los objetos....133

Indice de Figuras

Figura 1: Ejemplo de presentación de un esquema estrella.....23
.....24
Figura 2: Diagrama entidad-relación para un esquema estrella.....24
Figura 3: Ejemplo de diagrama de DFM de una cadena de VENTAS ([Gol98]).....32
Figura 4: Representación gráfica de caminos ARRP y ACP para representar relaciones entre atributos en el modelo GOLD ([Tru99]).....39
Figura 5: WarGen – Clases Temporales y Clases Archiving y su relación de composición con la Clase Genérica ([Rav99]).....40
Figura 6: WarGen – Clases Temporales con Filtros ([Rav99]).....41
Figura 7: WarGen – Clases Arachiving y la agregación de Clases Temporales ([Rav99]) 42
Figura 8: Ejemplo de celdas de un cubo: P(Ca) con Ca = {A,B,C,D} ([Alb02]).....45
Figura 9: Metamodelo presentado por Abelló en su tesis doctoral: YAM2 [Alb02].....46
Figura 10: Modelo UML para el hecho "Venta de Productos".....48
Figura 11: Herramienta CASE desarrollada por los autores del modelo GOLD [Tru99] para generar prototipos. ([Tru01b]).....49
Figura 12: Diagrama representando modelo de arquitectura de CWM (adaptado de [CWM02]).....52
Figura 13 : Modelo de Objetos de CWM (adaptado de [CWM02]).....54
Figura 14 : CWM – Metamodelo OLAP ([CWM02]).....56
Figura 15: OODW - Metamodelo multidimensional orientado a objetos propuesto.....61
Figura 16: Grafo de celdas en un Hecho con dos dimensiones.....62
Figura 17: Metamodelo de la dimensión.....62
Figura 18: Metamodelo de las métricas.....63
Figura 19: Metamodelo de Hechos.....64
Figura 20: Metamodelo del Cubo.....65
Figura 21: Ejemplo de instanciación de OODW con las dimensiones detalladas.....68
Figura 22: Ejemplo de instanciación de OODW con detalle de niveles base.....69
Figura 23: Ejemplo de instanciación del metamodelo de las métricas – Monto adeudado 69
Figura 24: Ejemplo de instanciación del metamodelo de las métricas - CantContribuyente.....69
Figura 25: Ejemplo de instanciación del metamodelo del cubo – Cubo AFIP.....70
Figura 27: Diagrama de clases - Modelo de Construcción.....72
Figura 28: Diagrama de interacción - Ejemplo de instanciación de un cubo.....73
Figura 29: Metamodelo de Consulta.....74
Figura 30: Jerarquía de Consulta.....74
Figura 31: Diferentes requerimientos de controles de acceso para OLAP ([Pri00]).....77
Figura 32: Metamodelo multidimensional propuesto por [PrPe01].....80
Figura 33: Modelo conceptual del escenario elegido a partir del metamodelo de [PrPe01].....81
Figura 34: Modelo conceptual de seguridad para el escenario elegido [PrPe01].....83
Figura 35: Metamodelo de seguridad de Trujillo y Piattini.....84
Figura 36: Descripción UML de un simple editor de figuras [Kic01].....87
Figuras 37a y 37b: Especificación de arquitecturas de software tradicional e incluyendo aspectos [Nav02].....91
Figura 38: Aspecto de memoria.....93
Figura 39: Diagrama de concerns – características principales.....93

Figura 40: Un modelo de requerimientos orientado a aspectos.....94
Figura 41: Especificación de templates en un patrón de composición Trace [Clar01].. 97
Figura 42: Especificación de vinculación para composición [Clar01].....97
Figura 43: Especificación de patrones de comportamiento transversal [Clar01].....98
Figura 44: Diagrama de clases de UML con estereotipos [Wai02].....99
Figura 45: Descomposición a nivel de paquete [Gro04].....100
Figura 46: Ejemplo de diseño del aspecto Tracing con AML [Gro04].....100
Figura 47: Modelo de diseño con Themes/UML [Clar01b].....105
Figura 48: Vista genérica del proceso de composición de temas [Clar01b].....106
Figura 49: Diagrama de objetos del "Bounded-Buffer" usando el perfil AOSD [Ald03] 109
Figura 50: AODW - Metamodelo de seguridad extendido con aspectos.....111

Indice de tablas

Tabla 1: Clasificación de modelos multidimensionales.....21
Tabla 2: Comparación de diferentes propuestas para diseño de aspectos.....102
Tabla 3: Mapeo de alto nivel de los elementos de programación de AspectJ a Theme/UML.....107

SECCION I – Introducción

En esta sección exponemos los principales motivos que nos llevaron a encarar esta tesis y en "Organización de la Tesis" damos una breve introducción a los temas a tratar en cada capítulo. Por último presentamos el caso de estudio que utilizaremos a lo largo del presente trabajo.

Motivación y objetivos

La evolución del *Datawarehousing* ha estado marcada por un problema concreto: la falta de modelos que logren abstraer la complejidad inherente al modelado multidimensional [Alb01]. Últimamente han surgido varias propuestas de modelos conceptuales que por deficiencias de cubrimiento de las propiedades deseables o por su complejidad de uso no han llegado a una aceptación global en el mercado. Algunas de ellas se pueden encontrar en: [AbSaSa01b], [Alb01b], [Tru01b], [MTSP05], [Mor05], [Mor04b], [Maz05] y [Med05]

De nuestra investigación y experiencia en el área, podemos afirmar que dentro del ámbito de la consultoría e implementación de soluciones de datawarehousing se ha popularizado la utilización de modelos lógicos como si fueran modelos conceptuales.

Particularmente, estamos hablando del Modelo Estrella (describiremos este modelo en detalle más adelante), o algunos de sus modelos derivados, que fueron y siguen siendo la opción más utilizada al momento de modelar bases de datos multidimensionales para implementar datawarehouses.

La razón por la cual se ha instaurado como estándar al momento de modelar bases de datos multidimensionales, no sólo se debe a la simpleza del modelo y a la popularidad adquirida por éxito del libro "The Data Warehouse Toolkit: The Complete Guide To Dimensional Modeling", donde se lo describe como "la" herramienta de modelado multidimensional [Kim96] sino a la falta de un modelo conceptual multidimensional. Esta tendencia que mantiene a un modelo lógico como el modelo de facto para el modelado multidimensional no ha sido revertida hasta el día de hoy a pesar de las limitaciones inherentes que presenta un modelo lógico por sobre un modelo conceptual y no ha sido por falta de propuestas. Creemos que todas ellas o se han quedado reducidas a ámbitos de investigación o no han logrado imponerse debido a su complejidad y por ende, nuestro objetivo es alcanzar un modelo conceptual que brinde el poder semántico necesario para estas soluciones pero que, por sobre todo, sea sencillo de entender y de usar.

Otro de los motivos que también sustentaron la falta de modelos conceptuales fue que se prestó mucha atención a la performance del datawarehouse, que es un punto muy importante que puede marcar el éxito o fracaso de estas soluciones y no siempre es fácil de lograr. Esto provocó que gran parte de las líneas de investigación de datawarehousing se focalizaran en la optimización a nivel lógico y físico de los modelos dejando así de lado el modelado conceptual.

De las distintas investigaciones y propuestas que surgieron relacionadas a la búsqueda de modelos multidimensionales, cada una con sus fortalezas y debilidades, se distinguen los modelos basados en el modelo relacional (ROLAP) [MStr95, MStr97, Info97, RedB97] o en bases de datos multidimensionales propietarias (MOLAP[3]) [Arbo96].

No obstante los años que este tipo de soluciones llevan en el mercado, la diversidad de herramientas propietarias disponibles y a pesar de la cantidad de proyectos implementados y en vías de implementación, gran cantidad de reportes sugieren que entre el 40% y el 50% de los proyectos de datawarehouse fracasan [Wix01], [Cut03]. Diversas son las razones para estos fracasos, pero sin dudas, tal magnitud de

implementaciones no exitosas tienen raíz en que la comunidad de usuarios y diseñadores carece de terminología estándar, homogeneidad en la definición de conceptos y lineamientos para lograr documentaciones completas que permitan a los diseñadores validar fácilmente los modelos con los usuarios y contrastarlos con los requerimientos de negocio. Modelos no validados correctamente llevan a implementaciones que en general distan bastante con las expectativas y necesidades de los usuarios. La utilización de modelos lógicos como base del modelado multidimensional lejos de ayudar a disminuir esta tendencia, la favorece.

Los proyectos de datawarehouse son caros: a menudo necesitan años para ser implementados correctamente y requieren millones de dólares en términos de software, hardware y servicios de consultoría. A pesar de esto la venta de productos de DW continúa creciendo año tras año. El mercado de herramientas de datawarehousing alcanzó los 7.9 billones de dólares en 2003 experimentando un crecimiento del 11%, el triple del año anterior.

A todo esto se suma el hecho de que la evolución de los datawarehouses ha sido paralela e independiente a la evolución de las necesidades de seguridad y privacidad de la información. Esto se debe a que, en los comienzos, un datawarehouse estaba pensado para ser consultado sólo por altos niveles jerárquicos de las organizaciones y, por lo tanto, no era necesario que las aplicaciones OLAP brindaran ningún nivel de seguridad. Sin embargo, a medida que éstas fueron ganando lugar en el proceso de gestión de la información de las organizaciones, la comunidad de usuarios fue en aumento, incluyendo a personas ajenas a la organización (como ser proveedores). En consecuencia, las necesidades de control de acceso se fueron transformando en un requerimiento indispensable. Este problema se encuentra todavía sin una propuesta formal. Sólo hay aproximaciones parciales [Pri00] [Pri02] [Ac04], no existe un modelo conceptual y las pocas herramientas de seguridad actuales adolecen de los mismos problemas que las herramientas OLAP en general: son soluciones propietarias con sintaxis no aptas para propósitos de diseño conceptual y documentación.

En cuanto al trabajo científico en el campo, en los últimos años se han propuesto una amplia variedad de modelos. Algunos de éstos son modelos lógicos que formalizan conceptos de la teoría del datawarehousing a través de álgebras que operan sobre cubos multidimensionales. Otros, en cambio, proponen alternativas de implementación a nivel físico para almacenar los datos en entornos relacionales, orientados a objetos o multidimensionales propietarios [Vas99].

A pesar de todos estos trabajos, pocos autores han puesto especial atención al modelado conceptual y de estos son menos aún los que proponen modelos orientados a objetos, entre los cuales podemos citar a [Ngu01], [Tru99], [Tru01] y [Tru01b]. Según nuestra investigación, ninguna de estas propuestas de modelos conceptuales ha logrado alcanzar la popularidad del *"Modelo Estrella"* –definido más adelante- ni han sido adoptados por alguna herramienta conocida en el mercado del datawarehousing. Adicionalmente, creemos que no han salido de la comunidad científica, al menos, luego de mucha investigación no hemos encontrado ejemplos de su utilización en la práctica. Esto nos lleva a concluir que no han logrado satisfacer las necesidades en este área y por ende lograr su aceptación masiva y/o estandarización. Particularmente, nosotras consideramos que no alcanzaron a cubrir las propiedades deseables para un modelo conceptual multidimensional como veremos en los capítulos subsiguientes.

Por todo lo enumerado anteriormente, el objetivo y desafío de este trabajo es el de definir un **Modelo Conceptual Multidimensional** que cubra las necesidades de

análisis multidimensionales y seguridad con flexibilidad, completitud y con el grado de reutilización y adaptabilidad a los cambios que exigen las actividades de análisis a lo largo de grandes períodos de tiempo en entornos altamente dinámicos.

El advenimiento del *"Paradigma de Orientación a Objetos"* (POO), su poder semántico entre otras virtudes [Eng00], [Wir90] y, más recientemente, el surgimiento del *"Paradigma de Orientación a Aspectos"* (AOP) [Elr01], [Kic01] resultan excelentes herramientas para la definición de un modelo conceptual multidimensional con tales características.

A continuación analizaremos las diferentes propuestas de modelos conceptuales *"Orientados a Objetos"* (OO), identificaremos sus aportes, ventajas y desventajas, basándonos en estudios teóricos y en necesidades reales de los datawarehouses en la práctica hoy en día. El objetivo de este análisis es el de identificar el conjunto de características deseables que tendremos en cuenta en la construcción de nuestro modelo multidimensional conceptual OO: OODW.

Una vez definido dicho modelo, nos dedicaremos al problema del modelado de la seguridad, perfiles y control de acceso y propondremos una extensión a nuestro modelo conceptual multidimensional OO basado en el paradigma de orientación a aspectos que modelen estas cuestiones transversales de seguridad: AODW.

Organización de la tesis

Este documento está organizado en dos secciones que conforman el cuerpo principal de la tesis, seguidos de un conjunto de anexos destinados a brindar un mayor nivel detalle sobre algunos temas específicos.

Si bien estos anexos pueden resultar de ayuda para el lector, entendemos que desviarían la atención hacia cuestiones accesorias si se incluyeran como parte del cuerpo principal.

A continuación se describe brevemente el objetivo de cada una de las secciones principales:

- Sección I
    - Motivación y objetivos
    - Organización de la tesis
    - Caso de estudio
- Sección II
    - Marco teórico – Introducción a OLAP

      Describe una introducción a la teoría subyacente de datawarehousing necesaria para comprender la investigación posterior sobre los modelos multidimensionales. En los diferentes capítulos de esta sección se describen los modelos multidimensionales tradicionales, como *"Estrella"* y *"Copo de Nieve"*, mostrando su limitado poder semántico para el modelado conceptual.
    - Modelos multidimensionales OO

      Describe los beneficios de utilizar el paradigma de objetos para construir modelos multidimensionales.

      Basándose en fundamentos teóricos y en necesidades surgidas de la experiencia práctica lista una serie de propiedades deseables en un modelo multidimensional y luego se adentra en el estudio de la evolución de los modelos multidimensionales orientados a objetos propuestos hasta el momento.
    - Rumbo al metamodelo

      Presenta las conclusiones de los modelos estudiados en el bloque anterior y los compara con respecto a las propiedades deseables enunciadas para llegar a concluir que los modelos existentes todavía tienen suficientes falencias como para dar lugar a la propuesta de un nuevo metamodelo multidimensional que cubra las desventajas encontradas en los anteriores.

- OODW - Un metamodelo multidimensional OO

  Presenta una innovadora propuesta de un metamodelo multidimensional orientado a objetos que pretende subsanar todas las desventajas de los modelos estudiados en el capítulo previo cubriendo, sino todas, la mayoría de las características deseables para un modelo multidimensional enunciadas en el capítulo anterior.

- Instanciación de OODW

  Muestra un ejemplo de instanciación de OODW – el metamodelo presentado - para que el lector pueda apreciar la factibilidad de utilización en la práctica de dicho metamodelo.

- Aplicaciones de OODW

  Presenta posibles aplicaciones para instanciación y consulta que permitirían la utilización de OODW para crear herramientas de útil e inmediata aplicación en el mercado.

- Sección III

  - Modelos de seguridad en datawarehousing

    Este capítulo está dedicado al estudio de políticas y mecanismos de seguridad en el ambiente de datawarehousing.

  - Orientación a aspectos - una introducción

    Describe los conceptos fundamentales del paradigma de orientación a aspectos y a continuación conceptualiza en términos de aspectos cada uno de los pasos del proceso de desarrollo de software haciendo hincapié en el estudio de los artefactos utilizados para modelar aspectos en la etapa de diseño.

  - AODW - Modelado de la seguridad en DW con aspectos

    Propone una extensión orientada a aspectos al modelo conceptual presentado en el capítulo anterior para agregar al modelado de seguridad y control de acceso.

- Sección IV

  - Conclusiones y trabajo futuro

    Presenta conclusiones, posibles extensiones y trabajo futuro sobre los modelos presentados. Una de las propuestas es la construcción de una herramienta que permita generar modelos multidimensionales a partir de la instanciación o prototipación del modelo conceptual presentado utilizando para ello los modelos presentados al final de la Sección II.

- Anexo A

  Consiste en un glosario en el que se definen aquellos términos que no fueron aclarados durante el desarrollo de la tesis o cuando se trate de palabras en otro idioma –principalmente inglés- que sean utilizadas a lo largo de este trabajo y que no puedan ser reemplazadas, o no sea conveniente, por sus equivalentes en español por cuestiones de simplicidad o porque en el ámbito computacional son más frecuentemente utilizadas en su idioma de origen.

- Anexo B

  - Introducción a la POO

    Presenta una introducción a la programación orientada a objetos para aquellos lectores que no estén familiarizados con el paradigma.

Caso de estudio

Para ilustrar las distintas características del metamodelo propuesto, se presenta su aplicación a un caso de estudio que muestra las necesidades particulares de gestión de una entidad gubernamental de recaudación de impuestos de Argentina.

- Empresa o Ente: Organismo responsable por la gestión de la recaudación de impuestos y obligaciones de los contribuyentes. El Objetivo del Organismo es tener una visión integrada y consolidada de las obligaciones de pago y recaudación de los contribuyentes de cada una de las regiones que componen la estructura de este Organismo Recaudador, y ser capaz de obtener respuestas a consultas tales como "¿Cuál es la recaudación para el período P para la región R abierto por sucursales y tipo de Impuesto?", o "Compare el índice de cumplimiento de pagos para el período P con los períodos anteriores abierto por Impuesto".

- Características del Negocio: Este organismo es responsable por la gestión de la recaudación de un conjunto de impuestos y obligaciones de los contribuyentes. Un contribuyente puede tener más de una obligación, de acuerdo con su actividad. El contribuyente debe no sólo pagar la obligación sino que debe presentar la declaración jurada de la obligación, o la rectificación de la misma en el caso en que en la primer presentación hubiera habido errores. Asimismo, dependiendo del tipo de contribuyente, pueden admitirse diferentes métodos de pago y posibilidades de moratorias. Por otro lado, este organismo se encuentra estructurado geográficamente en regiones, cada una bajo una dirección diferente, responsable por la recaudación de la propia región.

Para realizar un seguimiento del grado de cumplimiento de las metas de recaudación impuestas, es necesario identificar el volumen recaudado, el porcentaje de obligaciones presentadas versus esperadas, la cantidad de presentaciones originales versus rectificaciones, y cantidad y monto de morosidad.

Hemos simplificado el caso al dominio sin tener en cuenta casos particulares tales como la moratoria que sólo se aplica al manejo de algunos impuestos en Argentina así como tampoco el manejo de rectificativas. También dejamos de lado el "Indicador de Cumplimiento" ya que requiere de una complejidad que excede el alcance de este trabajo.

En este negocio particular, el manejo de la información, desde el punto de vista de seguridad es altamente sensible por lo que es imperiosa la aplicación de perfiles y políticas de seguridad estrictas reflejen la estructura organizativa en cuanto a acceso a la información.

- Características de las necesidades de Análisis y Gestión: Consiste en el seguimiento de la recaudación incluyendo monto pagado y monto adeudado discriminado por:

    - *Agencias* que se encuentran agrupadas en *Regiones*.

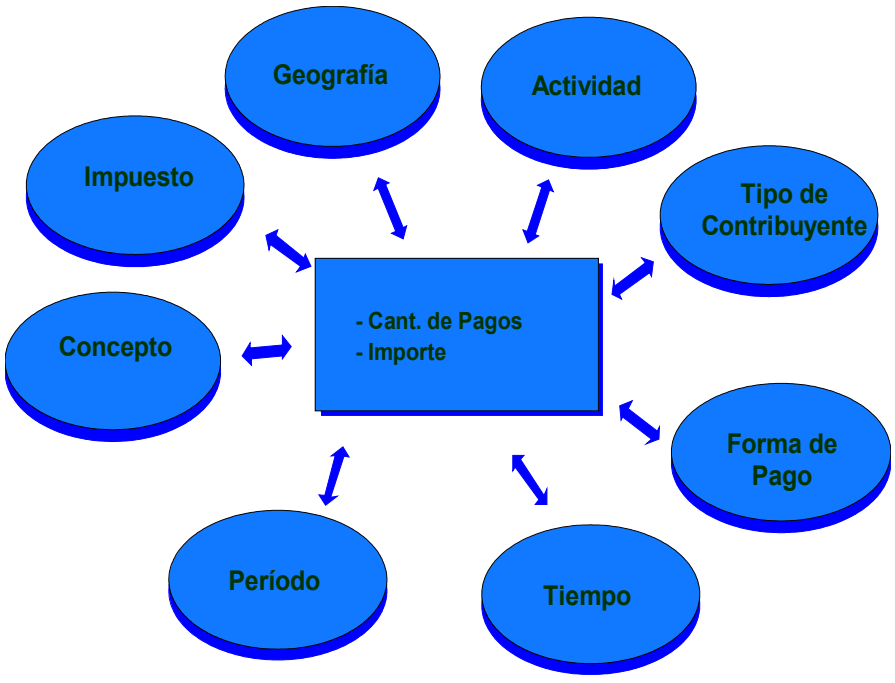
    - *Impuesto*s que se encuentran agrupados en *Tipos de Impuestos*.

    - *Actividades* que se encuentran agrupadas en *Tipos de Actividades*.

- *Contribuyentes* que se encuentran agrupados en *Tipos de Contribuyentes*.
- *Concepto de Pago*.
- *Forma de Pago*.
- *Período*.
- *Tiempo*.

Este organismo se encuentra estructurado geográficamente en regiones, cada una bajo una dirección diferente, responsable por la recaudación de la propia región. Los contribuyentes, de acuerdo con su actividad, pueden tener más de un impuesto asociado. Dependiendo también del tipo de contribuyente, pueden admitirse diferentes métodos de pago y posibilidades de moratorias. Para realizar un seguimiento del grado de cumplimiento de las metas de recaudación impuestas, es necesario identificar el volumen recaudado, el porcentaje de lo recaudado vs. lo esperado entre otras métricas.

Un esquema estrella básico será presentado como ejemplo basado en el siguiente escenario de negocios, dicho esquema se presentará en detalle en la Sección II:



SECCION II – Modelos multidimensionales

En este capítulo se presentan los conceptos básicos relacionados con los modelos multidimensionales y OLAP así como también una introducción a los modelos multidimensionales tradicionales y a los modelos multidimensionales orientados a objetos

Marco teórico – Introducción a OLAP

A mediados de los 90 E. F. Codd formalizó la frase "*On-Line Analytical Processing*" [COD93], de ahora en adelante **OLAP**, para caracterizar los requerimientos de análisis, sumarización, consolidación, visualización y síntesis de la información de acuerdo a múltiples dimensiones almacenadas en repositorios de información integrados. Una de las mayores ventajas de la multidimensionalidad de los datos radica en que se aproxima verdaderamente a la forma de pensar de un analista, facilitando las actividades de análisis de la información. Estas necesidades de análisis multidimensional distaban radicalmente de ser satisfechas con los reportes estáticos de los sistemas tradicionales de base de datos ya que éstos se encuentran optimizados para el procesamiento transaccional.

Se denomina "*On-Line Transactional Processing*" (**OLTP**) a sistemas donde predominan las transacciones concurrentes de inserción, actualización o eliminación que involucran pocos registros con tiempos de respuesta críticos. Por el contrario, las técnicas de análisis multidimensional se basan en la ejecución de pocas transacciones involucrando consultas complejas aplicadas sobre una gran cantidad de registros.

Esta técnica de análisis multidimensional, OLAP, junto a otras técnicas de acceso y visualización de la información estratégica como los reportes, *"Minería de Datos" ("Data Mining"*) y consultas ad-hoc, toman los datos de repositorios consolidados denominados *"Datawarehouses"* (**DWH**). Un Datawarehouse es una "colección de datos orientados a sujeto, integrados, no-volátiles y que varían en el tiempo para soportar la toma de decisiones estratégicas" según lo definió W.H. Inmon en "Building the DWH – 1992" [Inm92]. Analicemos este concepto en detalle:

- Orientado a sujeto: significa que el desarrollo del DW se lleva a cabo para satisfacer los requerimientos de análisis de los diferentes usuarios, que en general son los gerentes de las diferentes áreas y analistas.
- Integración: se refiere a la necesidad de unir información proveniente de diferentes sistemas operativos propios de la compañía o externos. En este proceso los típicos problemas que se deben resolver son, por ejemplo, diferencias en el formato de datos, codificación, sinónimos, multiplicidad de ocurrencia de los datos, etc.
- No volatilidad: implica durabilidad. Los datos no pueden ser removidos ni borrados.
- Variación en el tiempo: indica la posibilidad de contar con diferentes versiones del mismo objeto de acuerdo a sus variaciones en el tiempo.

Las estructuras de modelado y almacenamiento en DWH utilizadas por estas técnicas se conocen comúnmente bajo el nombre de *"Cubos Multidimensionales"*. Esta estructura fue elegida en los comienzos del DWH por tratarse del "modo más apropiado para describir las interdependencias de los datos" [Hac97]. Estrictamente hablando, el concepto de cubo se encuentra mal utilizado ya que como sabemos un cubo es un objeto tridimensional de lados iguales y justamente ambas características no suelen ser comunes en los modelos multidimensionales (**MMD**).

Contando con estas estructuras para el modelado multidimensional se intentó por mucho tiempo de equilibrar la necesidad de diseñar esquemas que puedan ser entendidos por los usuarios finales y que optimizaran la performance de las consultas.

Se podría decir que este equilibrio se alcanza gracias a la simplificación de los modelos multidimensionales lógicos/físicos, para que sólo contengan las entidades relevantes.

Así, podemos encontrar esquemas o técnicas de diseño que se acercan a la concepción que tiene el analista y sugieren la implementación de un tipo específico de consultas, como ser: la *"Relacional"* y la *"Multidimensional"*. A los modelos implementados sobre tecnologías relacionales se los denomina *"Relational OLAP"* (**ROLAP**) y a los que se encuentran sobre tecnologías multidimensionales propietarias se los denomina *"Multidimensional OLAP"* (**MOLAP**). ROLAP es una extensión al sistema relacional que mapea el modelo multidimensional sobre tablas relacionales accediendo a los datos a través de consultas SQL. Los sistemas MOLAP manejan y almacenan datos directamente en estructuras llamadas vectores multidimensionales. En [Buz98] se presentan las ventajas y desventajas de ambas tecnologías resaltando la propiedad de ROLAP de seguir estándares de la industria y la propiedad de MOLAP de tener un esquema físico similar a un cubo. (Se puede encontrar gran cantidad de información referente a esto en la bibliografía sobre investigación en DW mantenida por Daniel Lemire [Lem03]).

Como consecuencia de la divergencia natural de estas dos tecnologías de implementación de bases de datos multidimensionales, muchos de los denominados "modelos o técnicas de modelado multidimensionales" que surgieron se vieron estrechamente ligados con la implementación física (ROLAP o MOLAP) perdiendo la habilidad de conceptualizar adecuadamente la realidad a analizar. Esto trajo aparejado el problema de que tanto en la etapa de diseño del modelo multidimensional como durante el análisis de datos el usuario se vio obligado a considerar detalles sobre la organización física de los datos en vez de focalizar en los aspectos conceptuales/lógicos de los mismos.

Es por esta razón es que en los últimos años diversos grupos de investigación se han dedicado a la búsqueda de un modelo conceptual general del espacio multidimensional que sea independiente de cualquier implementación. Varias propuestas de modelos conceptuales o lógicos de bases de datos multidimensionales han surgido [AgGuSa97, CaTo98, BiTj01, LIWa96, TrPa98]. En particular, podemos apreciar que últimamente el modelado de base de datos multidimensionales se ha visto enriquecida por la aplicación del paradigma de orientación a objetos a la definición de modelos y frameworks multidimensionales [TrPa98], [Buz98]. En parte, esto se debió al poder expresivo de la orientación a objetos que permite lograr un modelo conceptual que se acerque más a la concepción que tiene el usuario sobre el universo de modelado [AbSaSa00].

Por lo expresado anteriormente entendemos que la aplicación del paradigma de orientación a objetos puede ser útil a los efectos de lograr un modelo conceptual más natural y simple que los modelos propuestos hasta el momento brindando mayor expresividad y representando mejor las necesidades de análisis de los negocios sin perder la flexibilidad de adaptación a cambios, reusabilidad y evolución del modelo.

A continuación describiremos la evolución de los modelos multidimensionales hasta llegar a los modelos multidimensionales orientados a objetos.

Modelos multidimensionales tradicionales

En general un *"Modelo de Datos"* es un conjunto de conceptos que pueden ser utilizados para describir la estructura de una base de datos. Estos modelos pueden ser categorizados en los siguientes niveles:

- Alto Nivel – Conceptual: representa las entidades importantes y sus relaciones. El objetivo de un modelo conceptual es describir los datos en forma independiente de la implementación. El modelo conceptual es más cercano al dominio del problema (la vista del mundo real) que al de la solución (implementación).

- Implementación - Lógico: provee conceptos que pueden ser entendidos por usuarios finales, pero que no se encuentran tan distanciados de la forma en que los datos serán organizados y almacenados en el sistema. Un ejemplo típico es el *"Esquema Estrella"*.

- Bajo Nivel – Físico: Provee conceptos que describen los detalles de la forma en que los datos van a ser almacenados en el sistema (específico sobre el motor de base de datos –DBMS- sobre el cual se implementa).

La instanciación de esta clasificación en los ambientes OLAP según las técnicas aplicadas sería:

Tabla 1: Clasificación de modelos multidimensionales

Categorización de MMDs	**RELACIONAL**	**OBJETOS**	**MULTIDIMENSIONAL**
Conceptual	Modelo de datos multidimensional (Multidimensional Data Model o MDDM)	Modelo de datos multidimensional (Multidimensional Data Model o MDDM)	Modelo de datos multidimensional (Multidimensional Data Model o MDDM)
Lógico	R-OLAP (relaciones, esquema estrella)	OO-OLAP (clases)	M-OLAP
Físico	DBMS relacional	DBMS orientado a objetos	DBMS multidimensional

A pesar de esta clasificación, históricamente, hablar de modelos multidimensionales es muchas veces hablar de modelos lógicos o físicos (notar que no se encuentra instanciado ningún modelo conceptual en la clasificación). Poco se ha desarrollado sobre cómo diseñar un modelo conceptual multidimensional a partir de los requerimientos de los usuarios. Esta es un área, donde las *"Bases de Datos Multidimensionales"* (**MDB**) no han logrado la estandarización de un modelo conceptual como lo es el *"Modelo de Entidad Relación"* (**Modelo E/R**) para las bases de datos transaccionales.

Según [Gol98] la falta de interés y avance en temas asociados al diseño conceptual de MDB's se debe a razones como:

- El DWH fue introducido como consecuencia de necesidades surgidas del mundo empresarial y liderada por usuarios que generalmente no prestan mayor atención o importancia a temas de diseño.

- Tanto el diseño lógico como el físico tienen un rol importantísimo en la optimización de la performance de los sistemas y como es uno de los objetivos del DWH más difíciles de alcanzar, han recibido mayor atención que el diseño conceptual.

Algunas iniciativas de modelos conceptuales multidimensionales han sido desarrolladas con el objetivo de replicar experiencias del mundo transaccional tomando el modelo E/R como punto de partida para la generación de un modelo multidimensional. Éstas no han sido exitosas y cuentan con muchos detractores. En [Kim96] Kimball sostiene que "Los modelos E/R no son comprensibles por los usuarios y no pueden ser navegados ágilmente por los DBMS. Los modelos E/R no pueden ser utilizados como base de un datawarehouse".

Debido a este vacío, modelos lógicos multidimensionales comenzaron a ser utilizados como modelos conceptuales. Este es el caso del *"Esquema Estrella"* que describiremos brevemente a continuación.

Esquema estrella: Star Schema (ROLAP)

Este modelo multidimensional tradicional fue introducido por Ralph Kimball en el año 1996 en su libro "The Data Warehouse Toolkit" [Kim96] y es conocido como "Esquema Estrella o Star Schema" y sus variantes ("Snowflake", "Fact Constellation" entre otros).

Este esquema no es más que una representación multidimensional de las necesidades de análisis y almacenamiento en una arquitectura relacional denominada ROLAP y es el resultado de aplanar el cubo donde sus lados y el espacio de análisis definido por ellos, se transforma en tablas relacionadas entre sí conformando una estrella.

El centro de la estrella consiste en una tabla denominada "**Tabla de Hechos**" donde se almacenan las métricas o puntos del espacio de análisis del cubo. Las métricas representan numéricamente los "eventos" ocurridos en el entorno de análisis. En los vértices de la estrella se encuentran las "**Tablas de Dimensión**" que representan los diferentes lados del cubo y que se relacionan entre sí a través de la tabla de hechos. Se dice que las dimensiones son los "sustantivos" del Datawarehouse.

Las tablas de dimensión se encuentran denormalizadas debido a que en una misma estructura almacenan todos los niveles de agregación posibles (jerarquía) junto con los atributos que la caracterizan. La normalización de atributos a través de otras tablas afectaría los tiempos de respuesta y la complejidad de las consultas. No olvidemos que los mayores desafíos de este esquema son simplicidad y velocidad. Muchos de los principios de diseño de bases de datos relacionales no son tomados en cuenta, principalmente por la característica estática o "silenciosa" de la MDB, es decir, por no recibir actualizaciones en línea.

Las jerarquías están formadas por la unión de atributos discretos de dimensiones en relaciones 1:N y determinan cómo deben ser agregados o detallados los hechos. Por ejemplo, en el caso de estudio que citamos en la sección I, una *Región* está compuesta de varias *Agencias*. Las jerarquías pueden a su vez incorporar atributos no dimensionales que caracterizan a la dimensión pero que no definen criterios de agregación, como por ejemplo el *Responsable de Región*.

Se describen a continuación dos ejemplos de presentación y/o documentación de un esquema estrella basado en el caso de estudio presentado. En la **Figura 1** las

dimensiones se encuentran descriptas por círculos y la tabla de hechos por un rectángulo.

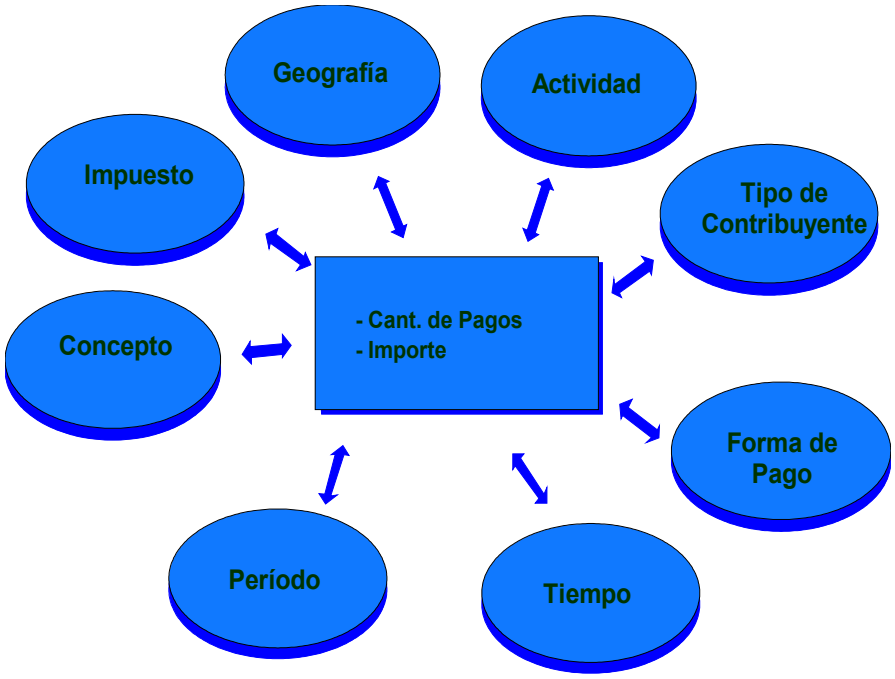


Figura 1: Ejemplo de presentación de un esquema estrella

El diagrama E/R de un esquema estrella se puede apreciar en la **Figura 2**.

Las operaciones que se pueden aplicar a este esquema a nivel cubo son:

- Pivoting o Rotación: Implica la rotación de un lado del cubo por otro. Por ejemplo cambiar el análisis del la *Recaudación por Impuesto* por la *Recaudación por Región*.

- Slice & Dice: Implica la selección de un subconjunto del cubo. Por ejemplo restringir el análisis de la *Recaudacion* solamente a la *Región* A.

- Roll-Up & Drill-Down: Implica la agregación o sumarización y detalle respectivamente de los datos solicitados del cubo. Por ejemplo pasar de ver la *Recaudación por Agencia*, a ver la *Recaudación por País* haciendo *"Roll-Up"* y pasar de ver la *Recaudación Anual* a ver la *Recaudación por Mes* haciendo *"Drill-Down"*.

Nuevas operaciones se han ido agregando como ser *"Expand"* y *"Collapse"* que permiten la visualización de atributos de dimensión no jerárquicos (más adelante nos explayaremos sobre este tema).

Resumiendo:

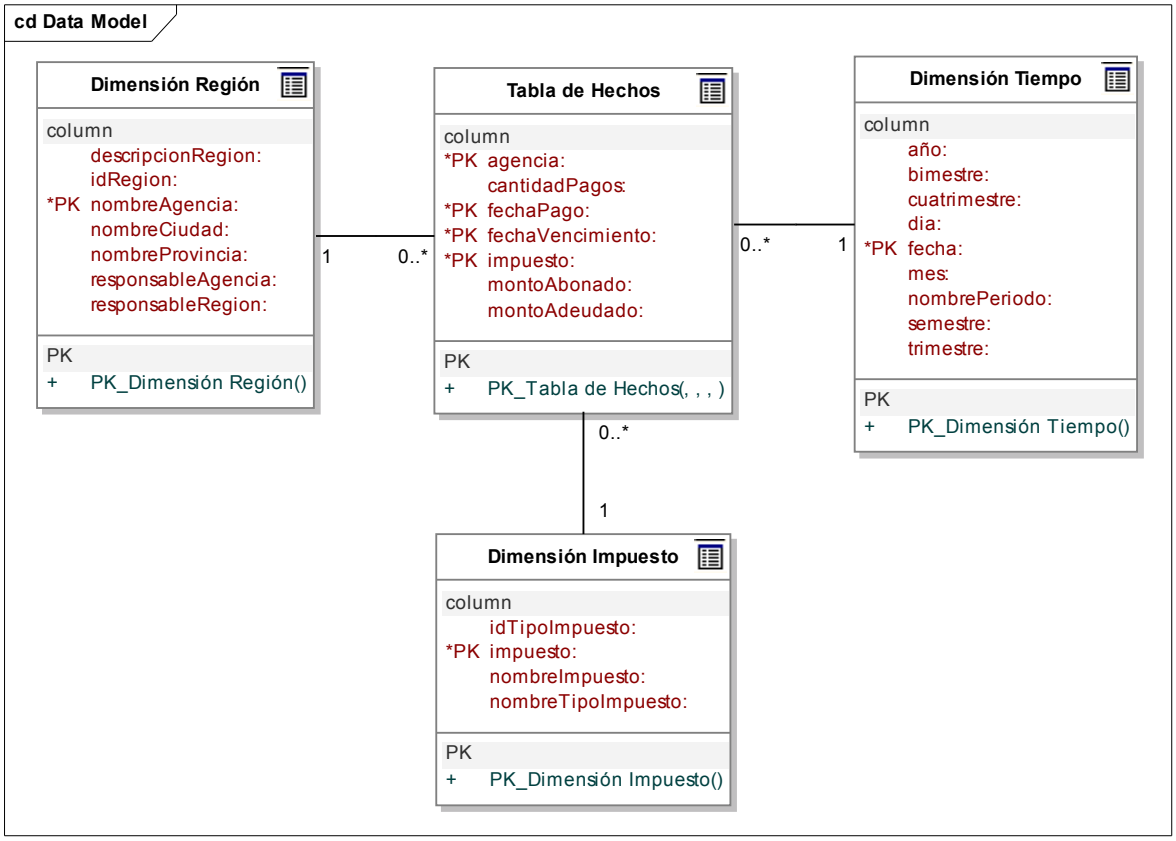


Figura 2: Diagrama entidad-relación para un esquema estrella

- Una única tabla de hechos – la tabla central del esquema estrella - con datos en su mayor nivel de detalle o granularidad.
- La tabla de hechos cuenta con una clave foránea apuntando a cada una de las dimensiones que la definen.
- Cada dimensión es almacenada en una única tabla, altamente denormalizada.

Beneficios

- Fácil de entender, intuitiva.
- Fácil definición de jerarquías, aunque no tan intuitivas al usuario final dado su nivel de denomarlización.
- Buen tiempo de repuesta en general al reducir la cantidad de uniones físicas.
- Estructuras muy simples de metadatos.
- Tecnología de BD estándar.

Limitaciones

- Las operaciones de sumarización para altos niveles de agregación tienen muy baja performance, se requiere la generación de vistas con la complejidad asociada de las mismas.

- Los tamaños de algunas tablas de dimensión se convierten en un problema de mantenimiento y de tiempo de respuesta.

- Complejidad para sostener la evolución de los datos a lo largo del tiempo (*"Slowly Changing Dimensions"*).

- La potencia del SQL no representa adecuadamente la semántica ni las operaciones de una BDM por el simple hecho de estar pensada sobre una tecnología relacional. Esto ha ido evolucionando en algunos motores de base de datos más que en otros a lo largo del tiempo.

- SQL no es lo suficientemente potente ni flexible como para soportar consultas complejas OLAP [Tho97].

Variaciones del esquema estrella: Fact Constellations

Se denomina *"Constelación de Hechos"* (Constellation/Overlapping Schemas) cuando se identifican más de una tabla de hechos que tienen una o más dimensiones en común. Las consultas asociadas a este esquema combinan métricas provenientes de dos o más espacios de análisis unidos por la misma dimensión o lo que es lo mismo, combinan dos o más cubos por los lados que tienen en común.

Las dimensiones que se encuentran en la intersección de dos o más espacios de análisis se las denomina *"Conforming Dimensions"* lo que indica que los valores que toman son consistentes a todos los espacios de análisis que afectan.

En los casos que las tablas de hechos tienen todas sus dimensiones en común es porque una de ellas es un agregado de la otra, permitiendo lograr mayor performance en consultas que requieren un mayor nivel de agregación.

Esta variación incluye una sola operación, "*Drill-Across"* que implica la combinación de datos provenientes de dos o más cubos o tablas de hechos unidos por una o más dimensiones en común.

Beneficios

- Logra mayor performance en consultas de mayor nivel de agregación

- Permite la combinación de dos o más espacios de consulta con tan sólo algunas dimensiones en común:*"Drill Across"*.

Limitaciones

- Crece la complejidad de las consultas, de la administración y del mantenimiento del esquema.

- Las estructuras de la metadata deben reflejar claramente la existencia de diferentes espacios de análisis y sus dimensiones en común, así como la existencia de agregados para optimizar los tiempos de respuesta de las consultas de altos niveles de agregación.

- El tamaño de algunas dimensiones continúa siendo un problema.

Variaciones del esquema estrella: Snowflake

Es otra variación del esquema estrella y surge de la necesidad de lidiar con dimensiones de gran cardinalidad o de muchos niveles de jerarquías. El esquema *"Snowflake"* es un esquema estrella normalizado donde las tablas de dimensión se normalizan a partir de la descomposición en sus niveles de atributos y jerarquías.

En las relaciones N:M la inserción de tablas intermedias permite para un mismo evento, por ejemplo una venta, contar con más de un vendedor responsable. En este caso, se generaría una tabla intermedia donde por cada evento se describirían todos los vendedores que intervinieron, sin crear más eventos en la tabla de hechos y apuntando a las tablas de descripción y atributos de los vendedores propiamente dicha.

Las estructuras jerárquicas en el esquema estrella común se introducían en la misma tabla de dimensión, como si se aplanasen los niveles que las componen. Ahora, en el caso de contar con jerarquías complejas en cantidad de atributos, la normalización parcial de las mismas a través de tablas que contienen punteros a otras tablas con las descripciones, facilita el manejo de esta complejidad. La mayor desventaja, adicional a la complejidad de las consultas, radica en que se debe conocer de antemano la cantidad de niveles que tendrá la jerarquía ya que cualquier cambio implica un cambio estructural en la base de datos.

Beneficios

Se observan principalmente en la performance en esquemas donde se encuentran gran cantidad de:

- Relaciones N:M.

- Estructuras Jerárquicas complejas.

Limitaciones

- Las consultas son más complejas, menos intuitivas y se pierde la facilidad que tienen algunos motores de hoy en día de reconocer consultas del tipo estrella para su mejor optimización.

- Complejidad de mantenimiento y de definición de la metadata.

- Incremento de la cantidad de tablas del esquema.

Motores multidimensionales propietarios (MOLAP)

Los sistemas de bases de datos multidimensionales propietarias o MOLAP surgen como una alternativa de implementación de esquemas multidimensionales aprovechándose de las limitaciones del esquema relacional.

Estos sistemas almacenan los datos en un array de vectores n-dimensionales como una representación del cubo. La generación de estos vectores resulta de la combinación de todo con todo, por lo tanto, muchas combinaciones tendrán la característica de ser "hechos vacíos" indicando eventos que no han ocurrido.

Este tipo de tecnologías requiere el cálculo previo de todas las posibles agregaciones del cubo, favoreciendo la performance en etapas de análisis y gestión en detrimento de actividades de carga y actualización de los datos y/o administración del cubo. Una de las mayores limitaciones que presenta esta técnica es la inflexibilidad de realizar consultas de tipo ad-hoc. Las bases de datos MOLAP deben tener predefinidas todas las posibilidades de navegación y niveles de consolidación al momento de transformación y creación de los vectores.

El espacio requerido de almacenamiento por estas tecnologías es en general, mayor al requerido por tecnologías relacionales debido a que en estas últimas, sólo se almacenan eventos ocurridos.

Beneficios

- Las consultas MOLAP, en términos de análisis multidimensional, son potentes y flexibles.
- Los modelos físicos son más cercanos al modelo multidimensional.

Limitaciones

- No hay un estándar para los motores MOLAP, dependiendo de los proveedores de esta tecnología propietaria completamente.
- Tienen grandes problemas de escalabilidad en espacio lo cual obliga a no poder mantener datos detallados en la mayoría de los casos.
- Carecen de buenos esquemas de autorizaciones y seguridad.
- No son flexibles a cambios/evolución del modelo multidimensional.

Limitaciones de los modelos multidimensionales tradicionales

Adicionalmente a las limitaciones particulares de cada modelo descripto anteriormente, muchos autores sostienen que estos esquemas tradicionales no representan adecuadamente ni la semántica ni las operaciones sobre datos multidimensionales [Gop99]. Parte se debe a su naturaleza relacional y en parte por haber tomado modelos lógicos/físicos como consecuencia de la falta de modelos conceptuales puros.

Los modelos lógicos o físicos son semánticamente pobres con respecto a los modelos conceptuales que brindan al usuario más información sobre la realidad modelada.

A causa de esta brecha semántica es que en estos esquemas no se alcanza el manejo eficiente de la complejidad de las operaciones de agregación cuando se ejecutan sobre entornos que cuentan con dimensiones de gran cardinalidad, con varios niveles de jerarquía y con tablas de hechos con información con mucho detalle y varias métricas combinadas en el mismo espacio de consulta. Esto incide directamente en tiempos de respuesta, mantenimiento y construcción de consultas sobre esquemas de datos complejos.

Una limitación particular a cierto tipo de consultas que utilizan a las métricas como atributo de agrupación (por ejemplo la visualizar para cada región los montos recaudados en rangos del tipo [0..1000] etc., donde el *Monto Recaudado* pasa a ser una característica de agrupación) han generado el debate de varios autores por el tratamiento simétrico entre dimensiones y hechos.

Algunos de estos autores [Agr95] y [Hac97] se han centrado en el manejo simétrico de las dimensiones y los hechos. Esto probablemente logre grandes mejoras a nivel de flexibilidad de las operaciones OLAP. Como consecuencia de esta flexibilidad, se pierde a nivel conceptual, parte de la percepción intuitiva que brinda la separación entre dimensiones y hechos.

Contrariamente, el tratamiento no simétrico brinda mayor información al momento del diseño lógico ya que permite conocer con mayor detalle las características inherentes de cada dato, facilitando el desarrollo de diseños que hagan uso eficiente de las mismas a través de aplicaciones OLAP.

Modelos multidimensionales OO

Se habla de "niveles de expresividad" o de "poder semántico" de los modelos cuando se quiere referenciar al grado con el cual representan o expresan una concepción del mundo real. El nivel semántico de un modelo mide la habilidad y flexibilidad de las estructuras de poder representar conceptualmente al mundo real. Cuanto más expresivo es un modelo, más cercana será su representación de la realidad. Es precisamente este "nivel de expresividad" o "poder semántico" el que está faltando en los modelos multidimensionales convencionales y que puede ser obtenido a través de la utilización de la POO.

Esta característica permite reflejar la realidad similar a la forma de pensar de las personas y una de las ventajas de conceptuar universos de discurso, a través de modelos orientados a objetos, es que su resultado es cercano a la concepción que tiene el usuario del mismo universo.

En [Alb00] los autores presentan los beneficios de un modelo multidimensional basado en el paradigma de objetos para integrar las diferentes vistas multidimensionales manteniendo la semántica de los datos a nivel conceptual. En su trabajo muestran como cada una de estas características puede agregar mayor poder semántico al modelo.

Beneficios de utilizar modelos multidimensionales OO

- Clasificación / Instanciación

De especial interés son las "clasificaciones dinámicas" y las "clasificaciones múltiples" (que no son lo mismo) ya que permitirán representar correctamente la gran cantidad de cambios que suceden en largos períodos de tiempo.

Se refiere con clasificación dinámica a la habilidad de las instancias de cambiar dinámicamente de clases. Por ejemplo: Los contribuyentes se encuentran en dos categorías, *Gran Contribuyente* y el resto. Si por decreto (cambia el piso por el cual un contribuyente pasa a ser *Gran Contribuyente*) o por evolución del contribuyente se requiere que éste sea cambiado de categoría, por ejemplo: ser recategorizado como *Gran Contribuyente*, el modelo debería ser lo suficientemente flexible como para permitir representar este cambio en forma dinámica.

Contrariamente, la clasificación múltiple se refiere a la habilidad de tener una instancia clasificada en más de una clase al mismo tiempo. Por ejemplo: tener a un cliente que al mismo tiempo es proveedor. Esta es una situación bastante común en la realidad de los negocios que encuentra dificultades para su correcto modelado.

Ambas características son altamente deseables, especialmente en el ámbito del datawarehousing, donde los conceptos de "no volatilidad" y "variación en el tiempo" junto con las necesidades de análisis de largos períodos de tiempo enfatizan su importancia.

- Generalización / Especialización

Es la relación *"SuperClase/SubClase"* que como consecuencia trae la herencia, donde cada clase hereda las propiedades de su/sus superclases adicionalmente a sus propios atributos y métodos.

A través de esta dimensión se facilitará el modelado de la especialización de hechos que generarán cubos. En este caso, podríamos tener un cubo para cada tipo específico de pago (crédito o efectivo) y un tercer cubo con las propiedades en común de ambos pagos, como ser el monto total de la lo abonado.

La generalización también nos permite generar nuevas dimensiones de análisis, en este caso, consultas generales de la recaudación de los impuestos en general a lo largo del tiempo sin necesitar el detalle de los tipos de impuestos o sus características particulares, facilitando la navegación.

- Agregación / Descomposición

Es la habilidad de construir nuevos objetos como consecuencia de la agregación de otros, que a su vez, pueden ser agregaciones también. Según el objetivo puede denotar una composición o una simple agregación.

A través de la relación de composición se modelará la relación existente entre los hechos y las dimensiones, debido a que un cubo implica la relación de composición entre las dimensiones que afectan al conjunto de métricas (no habría cubo sin sus partes) así como las relaciones de composición jerárquica de los niveles de las dimensiones. También nos permitirá el modelado de relaciones de composición entre cubos, facilitando las operaciones *"drill-across"*.

A partir de la relación de agregación que se representarán las operaciones de análisis multidimensional más comunes como el *"roll-up"* y el *"drill-down"*. Ayuda a la definición de jerarquías de análisis a través de los links de composición. La jerarquía tendrá a la clase con mayor nivel de detalle asociada a la clase de los hechos como hoja, y la clase que representa globalmente todos los puntos en la dimensión como raíz. En el medio, otros niveles de jerarquía representarán las diferentes granularidades. El nexo entre la clase hoja de la jerarquía y la clase de hechos formará la *"class-key"* de los hechos y es lo que la diferencia de los otros atributos.

- Mensajes (Llamador/llamado)

Implica el intercambio de mensajes entre objetos y la habilidad para aceptar y/o ejecutar métodos asociados, delineando el comportamiento del objeto. Esta característica lo diferencia en gran medida a las entidades utilizadas para modelos como el E/R que modela entidades que representan tablas relacionales o simples contenedores de datos sin ningún comportamiento.

La inclusión de métodos en los modelos de datos permite representar no sólo datos sino también comportamiento. Esta propiedad nos permitirá el modelado del tipo de función de agregación o patrón de agregación asociado (ver más adelante las

implicancias de un *"Patrón de Agregación"*) a un hecho o métrica determinada así como la definición de métricas calculadas.

En particular, también esta característica es importante para el diseño y modelado de las políticas de seguridad en un entorno de análisis multidimensional. Los métodos permiten la definición de roles y perfiles de acceso y visualización de acuerdo a reglas predefinidas (por ejemplo: acceder a la agregación y no al detalle).

- Derivabilidad

Permite la representación de métricas derivadas de otras básicas. Generalmente no es conveniente almacenar datos derivados (nuevas métricas obtenidas en base a otras primarias) excepto por razones de performance. Pero lo que si es necesario almacenar es que estos datos derivados existen y cómo se calculan u obtienen. Esto permite la completitud del modelo conceptual y su acercamiento al mundo real visto por el usuario. Estos mecanismos permiten reestructurar fácilmente el esquema para que represente lo que quiere el usuario más allá si la métrica en sí misma es un dato atómico o calculado, lo que es importante de tener en cuenta en el diseño son las definiciones de cómo son obtenidas las métricas.

- Dinamismo

Refiere a los cambios en el tiempo. Esta propiedad nos permitirá modelar lo que en la teoría del datawarehousing se conoce como *"slowly changing dimensions"* o lo que es lo mismo, cómo manejar el problema de la evolución de los datos a través del tiempo. Esto lleva consigo el dilema de la pérdida de información (en caso de perder su historia) o la provisión de información equivocada (análisis donde muestran al dato en un estado que no fue así en todos las líneas de tiempo).

Esta dimensión es utilizada básicamente para el seguimiento de la historia. En el dominio del datawarehousing es común analizar el pasado, por lo tanto se debe llevar registro de los estados pasados, por lo que a veces es común manejar esquemas de versionado o *time-stamping* que permitan ver la *"Verdad Histórica"*.

La aplicación de estas características del paradigma de objetos puede enriquecer semánticamente al modelado de esquemas multidimensionales, principalmente los modelos conceptuales. Esto es importante cuando la mayoría de los esquemas tratan de una tabla de hechos central y de diferentes dimensiones denormalizadas alrededor, relacionadas con la tabla de hechos por una clave foránea. Esta no es una mala idea, sólo que existe mucha más información sobre el universo de análisis de lo que este último esquema puede contener.

Adicionalmente a los beneficios obtenidos de aplicar el POO para obtener modelos multidimensionales que sean semánticamente más ricos y como consecuencia más cercanos a la realidad compartida por el usuario final, a nivel físico o de implementación también encontraremos que la extensión del análisis a datos no numéricos como imágenes puede ser manejada a través de las propiedad de encapsulamiento de datos junto a métodos de visualización y manipulación.

Es por esto, que no sólo a nivel conceptual se encuentran beneficios como consecuencia de la aplicación del POO. Igualmente, de ahora en adelante nos focalizaremos en los modelos conceptuales.

Propiedades deseables en un MMD conceptual

Cuando hablamos de propiedades deseables en un *"Modelo Multidimensional Conceptual"* (**MMC**) nos referimos a las propiedades inherentes al análisis multidimensional que permitan capturar todas las características particulares de un dominio de negocios del mundo real. Pretendemos que las técnicas y herramientas de modelado tengan la habilidad de capturar y modelar dichas propiedades de la misma forma que lo hacen con los conceptos básicos como dimensiones, hechos, cubos y operaciones.

Una de las primeras publicaciones que pone énfasis en la importancia de contar con un MMC como base para la construcción de un datawarehouse pertenece a [Gol98] y [Gol99]. Estos autores son unos de los primeros en focalizarse en el modelado multidimensional conceptual alcanzando a definir claramente la mayoría de las propiedades deseables de todo MMC a través de un modelo conceptual gráfico denominado *"Dimensional Fact Model"* (**DFM**) o *"Modelo Dimensional de Hechos"* que consideramos importante mencionar detalladamente continuación.

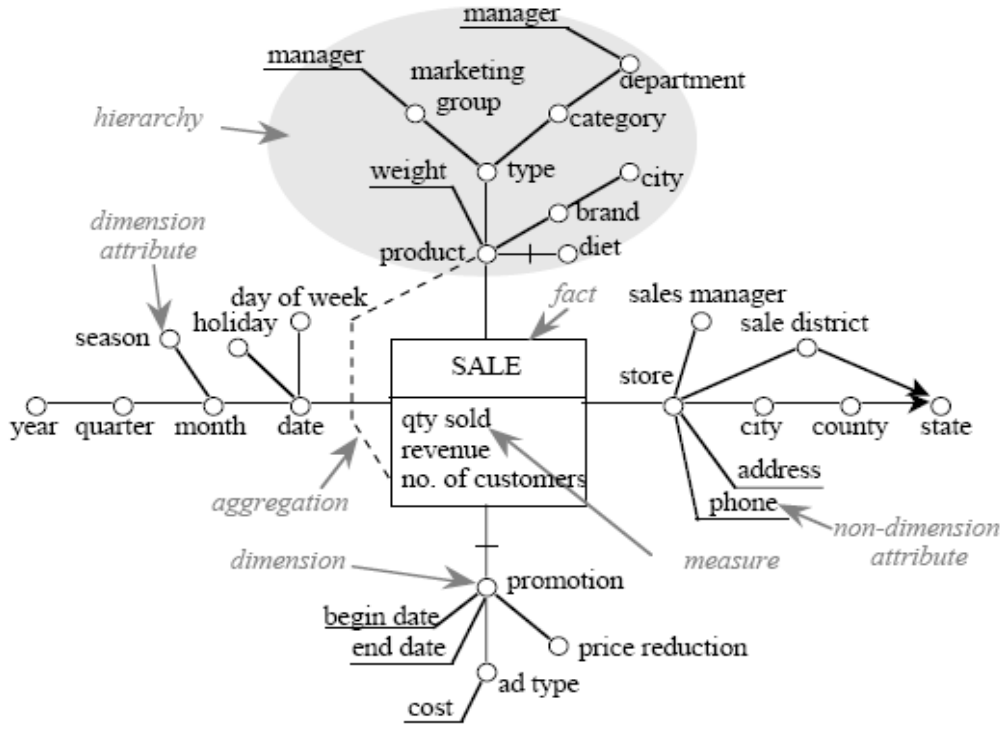


Figura 3: Ejemplo de diagrama de DFM de una cadena de VENTAS ([Gol98])

Es en [Gol98] que se introduce el DFM para la representación de los requerimientos de los usuarios y se compone de un *"Esquema Dimensional"* que a su vez está formado por un conjunto de *"Esquemas de Hechos"* compuesto por:

- Hechos
    - Objeto de análisis – *"Key Performance Indicador"* (**KPI**'s), por ejemplo: Ventas – Movimiento de Stock

- Medidas
    - Atributos continuos numéricos que describen al hecho de diversas maneras (un conjunto de medidas puede ser considerado un hecho) – Por ejemplo: *Cantidad Vendida*, *Ganancia*, *Cantidad Materiales Vendidos*.
    - Patrón de agregación: Permite clasificar las medidas en aditivas y no-aditivas sobre una dimensión en particular. (Esta propiedad se la indica con líneas punteadas que unen la medida con la dimensión con la cual no puede ser agregada).
    - Variedad de operadores de agregación: Las medidas pueden ser agregadas por operadores diferentes al estándar que es la suma.
- Dimensiones
    - Atributos discretos relacionados en forma de árbol y cuya raíz determina la mínima granularidad adoptada para representar los hechos. Por ejemplo: *Vendedor*, *Tienda*, *Región*, *Cliente*.
- Jerarquías
    - Se componen de atributos de dimensión que se encuentran asociados por relaciones a –uno y determina cómo deben agregarse los Hechos y de Atributos no-dimensionales que a diferencia de los anteriores, no pueden ser utilizados para operaciones de agregación ya que sólo proveen más información sobre el atributo asociado.
    - La dimensión que es raíz de una jerarquía define el menor nivel de agregación posible o la granularidad.
        - Por ejemplo una jerarquía con atributos dimensionales podría ser *Producto -> Tipo -> Categoría -> Departamento*.
        - Una jerarquía de atributos no-dimensionales sería: *Dirección de la Tienda*, *Teléfono del Cliente*, *Peso del Producto,* y son denominados *"Atributos Terminales"*, ya que no hay más camino posible a partir de ellos.
    - Representación de consultas sobre el DFM
        - Las consultas se basan en un lenguaje de expresiones *"Fact Instance Expressions"* o *"Expresiones de Instancia de Hechos"* y se focalizan en qué datos deben ser consultados y a qué nivel de agregación deberán ser consolidados.
        - Por ejemplo: *SALE(date, product, store, promotion;date.year >= '1995', product = 'P5').qtySold.*
    - Esquemas de dimensión superpuestos

- Permite la combinación de dos o más esquemas de hechos a partir de las dimensiones que tienen en común.

- La combinación de ambos esquemas debe realizarse al mínimo nivel común de granularidad de ambos esquemas. Por ejemplo: Si uno se encuentra a nivel de día, y otro a nivel de mes, la combinación necesariamente deberá realizarse a nivel de mes.

- Se introduce la operación *"Drill-Across"* para la combinación de esquemas de hechos.

Los conceptos introducidos por estos autores coinciden los que consideramos conceptos y componentes necesarios en todo MMC a partir de nuestra amplia experiencia práctica y por lo tanto serán los que tendremos en cuenta a la hora de validar la propuesta aquí presentada en cuanto a completitud y utilidad.

Es más, veremos que estas propiedades han sido ya utilizadas como bases para otros modelos conceptuales. En particular más adelante observaremos que tiene grandes coincidencias con el modelo conceptual OO GOLD de Trujillo [Tru99] en la definición de *"Jerarquías de Atributos Dimensionales"* (**ARRP**), y *"Jerarquías de Atributos No Dimensionales"* (**ACP**).

A continuación describiremos entonces cada una de los conceptos y propiedades que consideramos que un MMC debe soportar:

- Jerarquías
  - Jerarquías de Dimensión Explícitas: definen caminos de navegación explícitos entre niveles de jerarquía y los elementos de una dimensión. Se encuentran agrupados en niveles presentando una relación de orden parcial.

  - Jerarquías Múltiples: Los caminos de navegación descriptos en el ítem anterior pueden ser múltiples.

  - Jerarquías de Dimensión Implícitas o Especializaciones: definen diferentes formas de agrupar los elementos de una dimensión pero no necesariamente definen un camino preestablecido de navegación. El ejemplo más común de este tipo de jerarquías lo presenta las especializaciones de la dimensión *Producto en Familia*, *Tipo*, *División* etc.

  - Jerarquías Dependientes de Tiempo o Versión: La evolución natural que presentan los datawarehouse a lo largo del tiempo requieren de la necesidad de mantener mecanismos de versionado que permitan analizar los hechos sucedidos de acuerdo a las jerarquías o agrupaciones vigentes al momento en que se sucedieron los mismos. Esta decisión se encuentra estrechamente relacionada a la decisión de analizar *"La Verdad Actual"* (no importa en qué momento haya tenido lugar el evento, se analizará según las condiciones de agrupación actuales en la organización) o *"La Verdad Histórica"* (se analizará el evento de acuerdo a las agrupaciones vigentes en el momento de su ocurrencia) como consecuencia de las *"Slowly Changing Dimensions"* [Kim96], [GIO00].

- Atributos
    - Atributos de Dimensión o de Navegación: definen atributos propios de los elementos de una dimensión y definen agrupaciones de dichos elementos de dimensión. A diferencia de las jerarquías, los atributos de dimensión o de navegación no determinan ningún camino predefinido de navegación, simplemente, agrupan elementos de dimensión. Ejemplo de esto puede ser el *País de Origen* de un *Cliente*.

    - Atributos de Dimensión o Atributos Calificativos: definen atributos propios de los elementos de una dimensión, o de sus niveles. Son características adicionales que no representan agrupación alguna sólo brindan un nivel de información mayor. Ejemplos de esto pueden ser el atributo *dirección* asociado a la dimensión *Cliente*.

    - Atributos de Dimensión Derivados: implica la posibilidad de derivar un atributo a partir de otros. Ejemplo de esto sería la posibilidad de obtener el *Costo de Transporte* en base a los kilómetros a realizar y al producto a transportar.

    - Atributos de Dimensión (Calificativos o de Navegación) Dependientes de Tiempo: no sólo las jerarquías pueden tener la propiedad de evolucionar a través del tiempo, los atributos de dimensión pueden poseer esta misma propiedad. Ejemplo puede ser el *Responsable de un Area de Ventas*, o el *Area de Ventas* asignada a un *Vendedor*. Este va a ir cambiando a lo largo del tiempo, pero a nivel de gestión se desea analizar los datos junto al responsable en ese momento.

- Métricas
    - Métricas Derivadas: implica la posibilidad de derivar métricas a partir de otras básicas a nivel de cubo o lo que es lo mismo, al ejecutar los procesos de *"Extracción, Transformación y Carga"* (**ETL**). Ejemplo de esto es la derivación del *Total Vendido* a partir de la fórmula *Cantidad Vendida* * *Precio Unitario*.

    - Métricas Calculadas: implica la posibilidad de calcular en tiempo real, es decir, al momento de ejecutar una consulta, nuevas métricas calculadas. Particularmente esto es necesario para el modelado de métricas porcentuales, cuya particularidad de no agregación requieren de ser calculadas nuevamente para cada paso de navegación ejecutado.

    - Patrones de Agregación: los patrones de agregación se encuentran estrechamente relacionados con las métricas y sus propiedades de agregación. Existen métricas aditivas que pueden ser sumarizadas independientemente de las dimensiones por las cuales se requiera su agregación. También hay métricas semi-aditivas, cuyo comportamiento depende de la dimensión sobre la cual se requiere aplicar la agregación. Un ejemplo de este tipo de métricas puede ser el *Stock* o la *Cantidad de Empleados* en una empresa. Ambas requieren de una agregación por excepción si se encuentran sobre la dimensión tiempo, es decir, la cantidad

de empleados del año anterior, no es la suma de empleados mensuales, sino directamente los empleados del último mes. En este caso, la operación por defecto que es *SUMA* se cambia por *ULTIMO VALOR* para el caso de la dimensión *Tiempo*.

- Roles y Perfiles: la definición de roles basados en la política de seguridad de visualización de los objetos como dimensiones, hechos o cubos, así como la visualización de objetos parciales o niveles de agregación parciales.

- Consultas Predefinidas (no queries ad-hoc): la definición de las consultas con sus restricciones de filtros, excepciones, métricas calculadas, métricas restrictas, jerarquías de visualización y demás propiedades.

- Constelaciones: la posibilidad de modelar no sólo un cubo de análisis sino varios que tengan o no dimensiones en común.

- Propiedades de consultas MultiCubo: Esto implica las restricciones de aplicación de la operación *Drill-Across* y cómo deben conformarse las dimensiones resultantes de una operación entre cubos.

Estas propiedades son el resultado del estudio de diversas publicaciones dedicadas al diseño conceptual multidimensional.

Evolución de los modelos multidimensionales OO

La introducción del concepto de orientación a objetos en las publicaciones relacionadas al modelado multidimensional data de 1998. Dos importantes investigaciones abrieron el camino, una presentando un modelo físico implementado sobre bases de datos orientadas a objetos [Buz98], la otra presentado un modelo multidimensional orientado a objetos denominado OOMD [Tru98].

En [Buz98] se presentan las ventajas del modelado multidimensional sobre bases de datos orientadas a objetos con respecto a bases de datos relacionales o multidimensionales. A través de un framework denominado Object-Oriented OLAP (**O3LAP**), que permite el mapeo de un esquema lógico estrella a un esquema físico orientado a objetos, los autores definen en términos de clases y métodos el esquema multidimensional.

El mapeo es bastante simple, tanto las dimensiones como los hechos se corresponden con clases. La clase de hechos se define como una clase asociativa con respecto a las dimensiones para representar la granularidad del cubo modelado. El mapeo de las dimensiones introduce uno de los puntos de conflicto que luego se repetirá en publicaciones posteriores de otros autores como ser el modelado en términos de objetos de las múltiples jerarquías de dimensión, las relaciones entre los niveles y la relación de atributos que no implican un camino de navegación explícito. En este caso, se definen dos tipos de clase de dimensión, una para *"Dimensiones no-asociativas"* para modelar las dimensiones que tienen atributos sin un camino de navegación explícito (como por ejemplo: *Producto*, *Familia*, *Grupo*, *División*, *Línea*) y un para

"Dimensiones asociativas" donde se modelarán las dimensiones con caminos jerárquicos explícitos (por ejemplo: *Región -> País -> Ciudad -> Agencia*). Si bien es una solución posible a la necesidad de modelar la complejidad inherente a las relaciones de las dimensiones con respecto a sus jerarquías, no parece ser una solución flexible teniendo en cuenta la evolución natural de un datawarehouse a lo largo del tiempo y la posibilidad que dimensiones definidas como no asociativas pasen a ser asociativas. Paralelamente, el paradigma de orientación a objetos podría ser explotado de forma tal de modelar las relaciones no asociativas como relaciones de "generalización-especialización" y las relaciones asociativas como agregaciones de composición.

Los métodos asociados a las clases permiten una fácil implementación tanto de atributos derivados como de métricas derivadas así como de los patrones de agregación particulares a una métrica dada. Al tratarse de un modelo físico, se hace mucho hincapié en la persistencia de los objetos, que los autores logran a través del uso de *"Extents"* u *"Objetos Raíz"*. Esta es una característica fundamental para la implementación sobre motores orientados a objetos de modelos multidimensionales. Definen también consultas simples y compuestas en base a las clases de dimensión y hechos.

Por último, los autores clasifican las clases O3LAP según la taxonomía de Yourdon [You95] en datos, control e interfaz, de forma tal que las dimensiones y los hechos pertenecen a la categoría de datos y las consultas y operaciones OLAP pertenecen a la categoría de control con lo cual se alejan del paradigma de objetos.

Si bien nuestro objetivo es la obtención de un MMC OO, esta aproximación de un modelo físico multidimensional OO nos permite validar nuestras afirmaciones sobre la flexibilidad que el paradigma de OO le ofrece al modelado multidimensional en general, y a observar diferentes enfoques, en este caso de implementación, sobre cómo tratar las dimensiones y sus relaciones jerárquicas.

Contrariamente a O3LAP, la publicación de [Tru98] se acerca más al objetivo de investigación de la presente tesis ya que propone el primer modelo multidimensional conceptual orientado a objetos – **O**bject **O**riented **M**ultidimensional **D**atabase Modelling - (**OOMD**). Trujillo et Al. introduce la necesidad de que el mundo del datawarehouse adopte un modelo conceptual con alto nivel de abstracción así como lo tiene el mundo relacional. Sostiene asimismo, la importancia de tener un modelo conceptual multidimensional independiente de su implementación física y que el paradigma de orientación a objetos es la herramienta natural para brindar a dicho modelo de la independencia física y de potencialidad expresiva requerido por un modelo multidimensional. El mayor aporte de este trabajo radica en la formalización de los autores de los objetos modelados a través de un lenguaje de especificación formal.

Los autores, primero establecen la diferencia entre propiedades estáticas como las jerarquías de navegación de una dimensión y las dinámicas, como las operaciones que se aplican sobre dicha jerarquía para ejecutar el paso de navegación. Ambas podrán ser modeladas a partir del paradigma de objetos a través del concepto de encapsulamiento. Luego formalizan sucesivamente los conceptos básicos, como la clase *Dimensión* y la clase *Hecho*, ésta última se especifica como *una clase de tipo composición* con una relación de agregación con las clases *Dimensión*. Como se podrá observar, esta definición de la clase *Hecho* es similar a la implementación en O3LAP donde la clase *Hecho* es una clase asociativa con respecto a las clases de *Dimensión*. La clase *Cubo*, por su parte, se basa en la asociación de una única clase de *Hechos* y

de n clases *Dimensión* con las operaciones (*Roll-up, Drill-down, Pivot y Slice/Dice*) o eventos que se encuentran permitidos sobre dicha clase *Cubo*.

La definición de estos conceptos requiere a su vez de la definición de otros conceptos necesarios como la relación *"Attribute Roll-up"* (**ARR**). Esta relación es la formalización de la relación de las dimensiones con sus jerarquías de navegación o clasificación, que en O3LAP fue tratada como una dimensión diferente o "dimensión asociativa".

Esta relación ARR tiene la característica de establecer una relación de orden parcial entre sus elementos permitiendo la definición explícita de los caminos de navegación y la aplicación de las operaciones *Drill-down* y *Roll-up* sobre los mismos.

Esta es la primer aproximación de Trujillo al modelado multidimensional conceptual orientada a objetos, en las sucesivas publicaciones ha ido refinando sucesivamente este mismo modelo. En su primera publicación realiza una definición de la clase de *Hechos* donde mezcla el concepto de *"Degenerated Dimensions"* (que es un caso especial de una dimensión con una cardinalidad similar a los hechos – por ejemplo *Número de Ticket de Venta*) junto con la definición de los hechos en sí mismos. Lo hace a través de la definición de una clave en la clase de *Hechos* para reflejar este caso particular de una *"Dimensión Degenerada"*. Asimismo, en la misma clase de *Hechos* definen la relación ARR o lo que es similar, una relación jerárquica explícita entre los hechos, definición que no tiene mayor sentido. Ambas definiciones son modificadas en los sucesivos refinamientos del modelo.

El aporte principal radica en ser la primera aproximación formal de un MMC basado en el paradigma OO. Con respecto a las propiedades buscadas, sólo la formalización de las relaciones de jerarquías de navegación explícitas es contemplada en OOMD.

Evidentemente, a pesar de que la primera aproximación de Trujillo fue innovadora en términos de modelado multidimensional conceptual y el nivel de formalización de la misma, adolece de la mayoría de las propiedades deseables a representar mediante un modelo conceptual multidimensional. En 1999 Trujillo publica otro trabajo con un modelo conceptual multidimensional más completo que lo denominó GOLD [Tru99].

El modelo GOLD, además de modificar o extender algunas definiciones de conceptos básicos resultantes de OOMD, introduce al modelo propiedades como métricas y atributos derivados, jerarquías múltiples y agregación de hechos a través de las dimensiones. Se debe destacar, que este tipo de propiedades no son en general consideradas en lo que denominamos los modelos lógicos o modelos tradicionales, esto se debe a la poca expresividad del esquema de modelado utilizado donde sólo las propiedades estáticas son tomadas en cuenta. La orientación a objetos permite el modelado natural tanto de propiedades estáticas como propiedades dinámicas y como consecuencia su funcionalidad y comportamiento.

En lo que se refiere a mejoras introducidas en GOLD, una de las más las más significativas es la diferenciación entre atributos que conforman jerarquías de navegación y atributos calificativos. En OOMD sólo existía ARP, una relación para modelar jerarquías de navegación de las dimensiones. En esta publicación, el autor distingue entre dos tipos de relaciones entre niveles dimensionales: *"Atribute Roll-up Relation Paths"* (**ARRP**) a través de la cual modela las relaciones jerárquicas con caminos de navegación predefinidos a partir de un grafo acíclico, y *"Attribute Classification Paths"* (**ACP**) para poder especificar lo que conocemos como atributos

calificativos o que brindan mayor información al elemento de dimensión pero no son atributos "navegables". Estos caminos ACP son siempre de longitud 1 lo cual deja ver que en realidad se está forzando el tratamiento de ambas relaciones como caminos en un árbol cuando la esencia de las mismas y sus fines con respecto a los objetos de dimensión son diferentes. Mientras que los niveles o jerarquías explícitas tienen como fin agrupar en niveles de granularidad menor a un conjunto de objetos dimensionales, los atributos de calificación sólo brindan información adicional de cada uno de los mismos, sin fines de agregación.

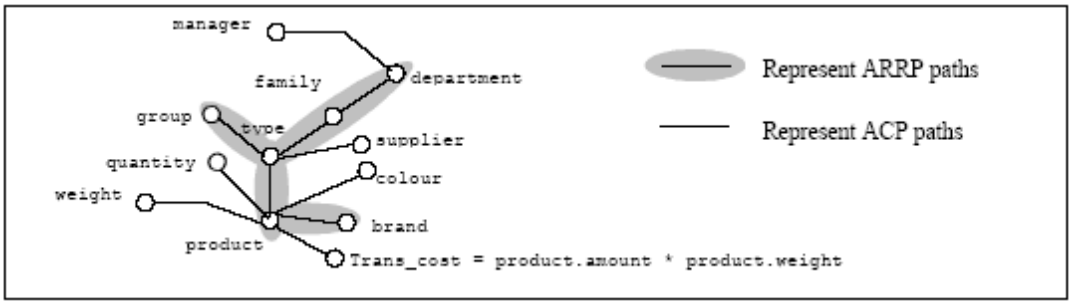


Figura 4: Representación gráfica de caminos ARRP y ACP para representar relaciones entre atributos en el modelo GOLD ([Tru99])

Esta nueva relación ACP requiere de operaciones asociadas que la explote, por lo que se introducen dos operaciones adicionales, *"Combine/Divide"*. La primera permite la visualización de un atributo de calificación mientras que la segunda lo oculta. Estas operaciones, al igual que las anteriores, son modeladas como métodos públicos de la clase *Cubo*.

Por último, para modelar el dinamismo de la ejecución sucesiva de diferentes pasos de navegación, la aplicación de operaciones sobre la clase *Cubo* genera *Vistas* que comparten las mismas propiedades que la clase *Cubo* pero sobre un conjunto menor de datos y a las cuales se les puede volver a aplicar otras operaciones para generar otras *Vistas*. Sobre esta clase *Cubo* y las operaciones OLAP el autor propone extensiones como trabajo futuro.

En cuanto a los aportes GOLD [Tru99] presenta un gran avance con respecto a OOMD y hacia la formalización de varias de las propiedades que entendemos como necesarias en un modelo multidimensional conceptual:

- Atributos Derivados
- Métricas Derivadas
- Jerarquías de Navegación Múltiples (ARRP)
- Atributos Calificativos
- Patrones de Agregación (Hechos aditivos, semi-aditivos y no-aditivos)
- Operaciones de Visualización de Atributos Calificativos (*Combine/ Divide*).

Continúa siendo su mayor aporte la formalización que sostiene la especificación del MMC OO y la flexibilidad que se obtiene al aplicar el paradigma OO con respecto a otros modelos.

Seguiremos más adelante con mejoras y avances sobre GOLD, pero primero analizaremos otras publicaciones contemporáneas a estos modelos, de forma tal de ir acompañando la evolución de los mismos y comparando los aportes de cada uno según el momento de su publicación.

En [Rav99] los autores, si bien presentan un modelo conceptual OO para un datawarehouse, no es un modelo conceptual multidimensional. Basan su estudio en el modelo conceptual de los procesos de extracción y transformación de un repositorio integrado y homogéneo de información que se nutre de fuentes de datos heterogéneas. A pesar de la divergencia de objetivos con respecto a los de este trabajo, hay ciertas propiedades que ambos modelos deben resolver y o manipular como es la evolución de los datos a lo largo del tiempo y la necesidad de mantener y modelar la historia. Es en esta área donde proponen un diseño conceptual innovador y lo sustentan con un prototipo para la generación automática de warehouses denominado *"Warehouse Generator"* (**WarGen**).

WarGen se basa en *"Clases de Objetos Genéricos"* (los datos), *"Clases de Enlaces"* (semántica de las relaciones entre las clases de objetos genéricos encapsulando datos y comportamiento) y restricciones sobre éstas. Pero el mayor aporte sin duda es la introducción de "*Clases Temporales"* y "*Clases de Almacenamiento" (***Archiving***)* para manipular la evolución temporal.

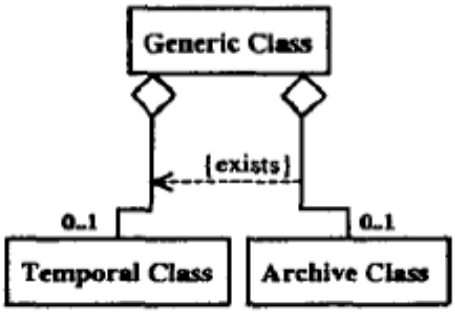


Figura 5: WarGen – Clases Temporales y Clases Archiving y su relación de composición con la Clase Genérica ([Rav99])

La clase *Temporal* se encuentra asociada a la clase de objetos genéricos y es a través de la definición de un *"Atributo Filtro"* que se define cuál de los atributos se le realizará el seguimiento temporal. En la siguiente figura se muestra cómo la clase de objetos genérica *Hospital* tiene asociada en una relación de composición una clase temporal *T_Hospital* con un filtro sobre el atributo *presupuesto* (*budget*) y cómo se van creando tantas instancias de *T_Hospital* como cambios del presupuesto anual se hubieren llevado a cabo.

Esto resuelve el problema del seguimiento de la evolución temporal de los datos que sean sutiles de modificarse a lo largo del tiempo, pero no resuelve el problema de cómo lidiar con el manejo de grandes volúmenes de información a través de la definición de diferentes niveles de agregación a lo largo del tiempo. Ejemplo de esto es la especificación de mantener los datos de presupuesto a nivel de año desde 1997 hasta la actualidad y la agregación a nivel de hospital para los años anteriores. Para

soportar esta propiedad los autores introdujeron la clase *Almacenamiento*, definiendo la política de agregación histórica sobre el mismo tipo de filtros que la clase *Temporal*. Continuando con el ejemplo de Presupuesto, se definió que el presupuesto de los años anteriores a 1997 sea consolidado o agregado de acuerdo a su promedio (*AVG(Presupuesto)*), por lo tanto, la clase de almacenamiento de la **Figura 7** consolidará todos los años de acuerdo con el promedio de los presupuestos anuales.

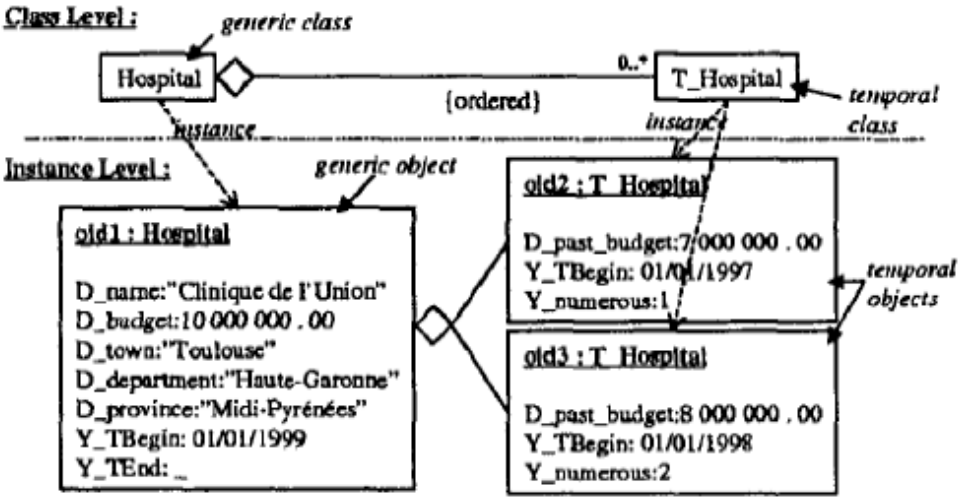


Figura 6: WarGen – Clases Temporales con Filtros ([Rav99])

El manejo de la evolución temporal de las dimensiones o atributos de dimensiones es una propiedad que no suele ser tenida en cuenta al momento del modelado y que es responsable del fracaso de varios DWH.

Este trabajo es uno de los pocos que avanza en el modelado de la evolución temporal a través de clases temporales y clases de almacenamiento con intervalos de tiempo de validez.

Siguiendo con la misma línea de investigación en [Rav00] dos de los autores dan un paso más en la formalización de lo propuesto en su publicación anterior, definiendo a un objeto *datawarehouse* como la composición de:

- Un identificador
- Un estado actual
- Un conjunto de estados pasados (temporales)
- Un conjunto de estados de almacenamiento

Y especificando los estados como la combinación de un intervalo de vigencia y un conjunto de valores válidos para dicho intervalo. Los intervalos son conjuntos ordenados, no vacíos, disjuntos y no continuos de las mismas unidades temporales.

Adicionalmente al manejo temporal de la información, y al tratarse de un modelo conceptual para el datawarehouse desde el punto de vista de un repositorio integrado, homogéneo, no multidimensional, los autores proponen siete (7) funciones de

"construcción" que permitirán la definición de los procesos de extracción, carga y transformación (ETL) del datawarehouse.

Con ésto, completan su modelo conceptual de un datawarehouse y el paso siguiente radica en el modelado de los *Datmarts*, que para los autores son las estructuras multidimensionales del DWH.

El mismo autor propone en [Tes01] un esquema multidimensional que no es más que un modelo lógico muy similar al esquema estrella sólo que le agrega el concepto de *Jerarquía Explícita* y *Caminos de Navegación Explícitos* (estos no son modelados explícitamente en el modelo estrella). Lo más innovador de esta publicación es la introducción de nuevas operaciones multidimensionales adicionales a las estándar conocidas hasta el momento, éstas se encuentran asociadas a la posibilidad de que un usuario dinámicamente cambie dimensiones, hechos y jerarquías entre sí. La operación *DRotate* implica la habilidad de un usuario de cambiar la visualización o la exposición de una dimensión por otra dimensión perteneciente al mismo esquema. Bajo el mismo concepto define *FRotate*, para el intercambio de hechos y *HRotate* para la rotación o intercambio de jerarquías activas. Estas se encuentran implementadas en un nuevo prototipo denominado GEDOOH, que es el sucesor del WarGen, que combina una interfaz gráfica, un generador automático de datawarehouses y un generador automático de datamarts.

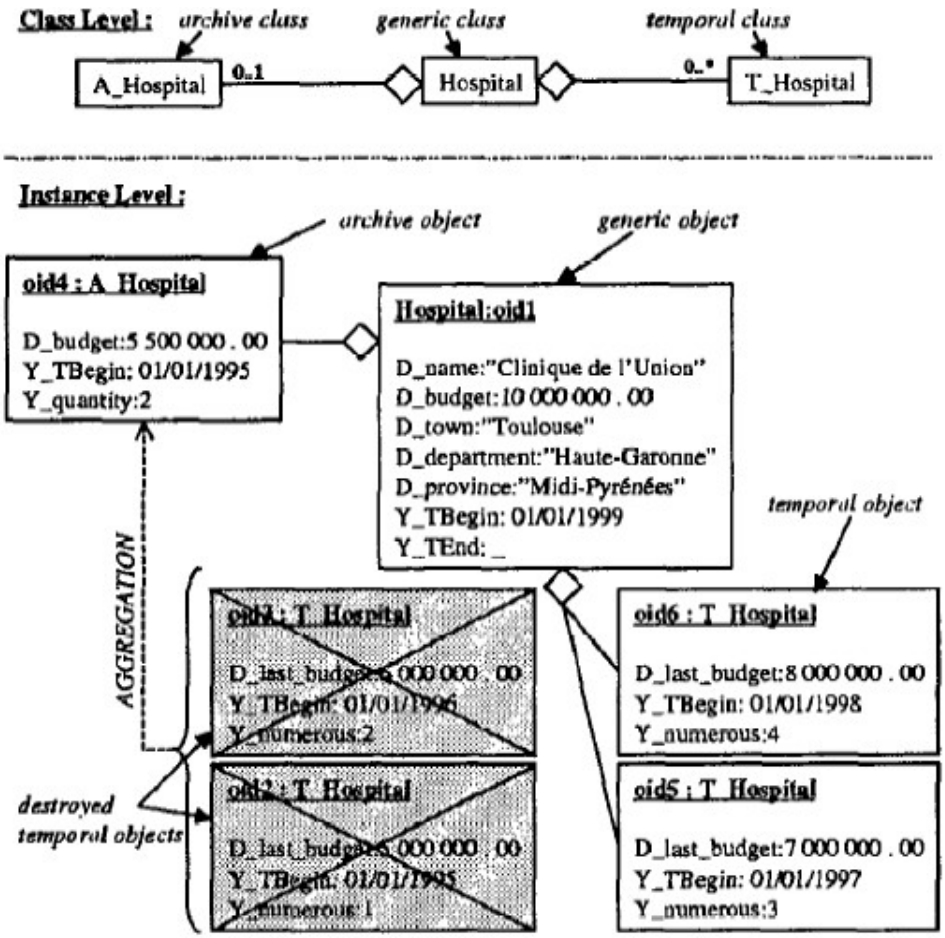


Figura 7: WarGen – Clases Arachiving y la agregación de Clases Temporales ([Rav99])

Estas operaciones nos resultan muy intuitivas y demostrativas de las posibilidades de manipulación de información multidimensional por parte de un usuario y complementan las tradicionales como *Roll-up, Drill-down, Slice/Dice. Jumping* sería tal vez la más cercana a estas operaciones, pero definitivamente, éstas últimas son mucho más descriptivas.

Otro trabajo contemporáneo, que continúa con la línea de [TrPa98] y [Tru99] desde el punto de vista de búsqueda y propuesta de un MMC OO es "*An Object Oriented Multidimensional Data Model for OLAP*" de [NgTjWa00].

A diferencia de los trabajos que lo preceden, los autores logran proponer no sólo la formalización de un MMC capaz de capturar y representar la naturaleza jerárquica de las relaciones entre los miembros de una dimensión, sus estructuras complejas consecuencia de las relaciones entre los miembros de las dimensiones y las métricas, sino que también proponen su representación en términos de clases utilizando UML. Esta representación en UML no es una simple instanciación de un caso, sino que representa un metamodelo conceptual para el diseño de modelos multidimensionales. Este modelo formal contempla la definición de *Dimensiones, Métricas, Cubos de Datos y Operaciones*.

- Dimensiones: Se pueden definir *Jerarquías Múltiples* con caminos predeterminados de navegación pero no hacen referencia alguna a atributos de "calificación" o atributos que no sean atributos de navegación. Asimismo, tampoco hemos encontrado referencia alguna de cómo manejar la evolución del tiempo de ciertos atributos de dimensión (Slowly Changing Dimensions).

- Métricas: Presentan la posibilidad de utilizar diferentes operadores de agregación como *SUM, COUNT, MAX* o *Promedios* pero no presentan el tratamiento de patrones de agregación para métricas semi-aditivas o con restricciones de agregación por una o más dimensiones.

- Cubos de Datos: Los definen como una "colección de celdas". Cada celda es la intersección entre los miembros de un conjunto de dimensiones y los valores de las métricas. Es el primero que los trabajos que introduce el concepto de celda para modelar el espacio n-dimensional conformado por las dimensiones y las métricas en su mayor nivel de detalle o granularidad. Veremos más adelante, que esta iniciativa es seguida en particular por los trabajos de Alberto Albelló y otros. Adicionalmente, define agrupaciones de celdas en "grupos de celdas" que no es más que la representación de todas las posibles combinatorias de los niveles de dimensión por las métricas.

- Operaciones: Las operaciones se llevan a cabo a través de "operadores del Cubo" se aplican sobre el cubo de datos para lograr los diferentes pasos de navegación. Dichos operadores son *Jumping, Roll-Up* y *Drill-Down.*

- Representación del Modelo: Una vez definidos estos conceptos formalmente, los autores representan el modelo en términos de diagramas de clases utilizando UML, que es la notación estándar para el análisis y diseño OO.

La formalización del modelo conceptual y su representación en UML es el mayor aporte de los autores aunque dicho modelo resulta muy difícil de entender y los autores no presentan una instanciación de ejemplo. Por otra parte, el metamodelo así como las

restricciones que aplican al mismo están expresadas en un lenguaje formal no estándar.

Como mencionamos anteriormente, un grupo de investigadores: Alberto Albelló, José Samos y Felix Saltor, continuaron con la propuesta de modelar a través de celdas el espacio multidimensional generado por la combinación de las dimensiones con los hechos. Estos autores comenzaron con publicaciones como [Alb01] y [AbSaSa01b] donde realizaron una investigación y clasificación de los modelos multidimensionales como también una publicación exaltando los beneficios de un modelo multidimensional OO. Conceptos de ambos trabajos han sido ya tomados y descriptos a lo largo de la presente tesis, sin embargo, sus últimas dos publicaciones [Alb01b] y [Alb01c] - *"Understanding Analysis Dimensions"* y *"Facts in a Multidimensional Object-Oriented Model"* respectivamente - son los que, basados en las relaciones de agregación de las dimensiones y en las relaciones entre cubos y su forma de navegar, proponen un nuevo modelo OO multidimensional.

El primer trabajo, [Alb01], fue dedicado a la investigación de los problemas para representar a nivel conceptual las dimensiones de análisis y sus jerarquías de agregación con el objetivo de encontrar, utilizando los beneficios de la POO, la manera de modelar la semántica de agregación y los caminos de navegación a través de las dimensiones. Como conclusiones de su investigación, sostienen que para modelar las relaciones entre los niveles de agregación se debe utilizar las relaciones de composición. Esta relación de composición, se debe entender como una "colección" de objetos de similar composición estructural (por ejemplo: Un convoy de camiones) a diferencia de una relación compleja como la que pueden presentar los elementos de un motor, donde cada pieza cumple una función diferente. Entonces, teniendo esto en cuenta, una operación *Drill-down* o *Roll-up* se realizará entre niveles o colecciones a partir de una relación de composición, sin embargo, una operación *Slice* o selección se realizará sobre relaciones de simple agregación al tratarse de los mismos elementos. Los autores finalizan su trabajo formalizando el concepto de análisis de dimensión basado en la relación de composición (*Part-Whole*) y en la necesidad de utilizar la relación de especialización para tener la capacidad de mostrar solamente los atributos comunes a un subconjunto de instancias.

Asimismo, frente a la discusión histórica sobre si la normalización de dimensiones es un error o no (Kimball sostiene en [Kim96] que en la mayoría de los casos, a pesar de optimizar el espacio físico, las consultas son más lentas y complican el modelo a los usuarios finales), los autores sostienen que es bueno o malo según el contexto. Si se habla de modelos lógicos o físicos, la denormalización de las tablas de dimensiones es lo más razonable en términos de performance. En cambio, a niveles de modelos conceptuales, la posibilidad de hacer explícitos los caminos de agregación y sus diferentes semánticas es esencial para poder modelar la realidad de forma más cercana a como es percibida.

En conclusión, a niveles conceptuales, cuanta más semántica y más explícito sea el diseño, más cercano será el modelado con respecto al mundo real.

El segundo trabajo de estos autores [Alb01c], se orienta al estudio de los *Hechos*, o *Facts* en modelos multidimensionales. Los autores presentan un modelo basado en que las métricas se encuentran agrupadas en "celdas" de diferentes clases formando los cubos n-dimensionales y éstas se encuentran agrupadas en base al hecho que representan.

Es importante destacar que un conjunto de métricas conforman el hecho de análisis. Relacionando esto con celdas, todas las métricas pertenecientes al mismo hecho se encontrarán en la misma celda ya que la celda es la representación del hecho. Pero, así como hay diferentes dimensiones y diferentes niveles de agregación, habrá una celda que reúna las métricas para cada una de las combinaciones posibles entre dimensiones y sus niveles jerárquicos. En particular, celdas básicas se combinan para definir nuevas celdas a un nivel de agregación mayor. No nos olvidemos, que la celda es la materialización del espacio n-dimensional de un cubo.

Los autores denominan "Ca" o *Celdas Atómicas* al conjunto de celdas que no pueden ser descompuestas en celdas en otras con menor nivel de agregación. Es decir, son las celdas que contienen las *Métricas Básicas*, o *Métricas Atómicas*, o en su mayor nivel de desagregación o detalle. Esto lleva a que la cantidad de celdas totales dado un conjunto de celdas básicas sea igual a = $2^{\text{Card. (Ca)}} - 1$. Donde Ca es el conjunto de celdas del cubo por ejemplo Ca = {A,B,C,D}

Los hechos en sí mismos no brindan información. Sólo combinados con las dimensiones que los definen es que toman sentido y con sus niveles de agregación. Los autores identifican clases de celdas formadas por celdas que contienen métricas al mismo nivel de agregación. Por ejermplo: las celdas que se encuentren a nivel *Mensual*, en términos de tiempo, formarán parte de la misma clase. Esta definición de clases de celdas permite que sólo la comparación de celdas que se encuentran en la misma clase sea válida, ya que es cuando se habla del mismo tipo de información o a la misma *Granularidad*.

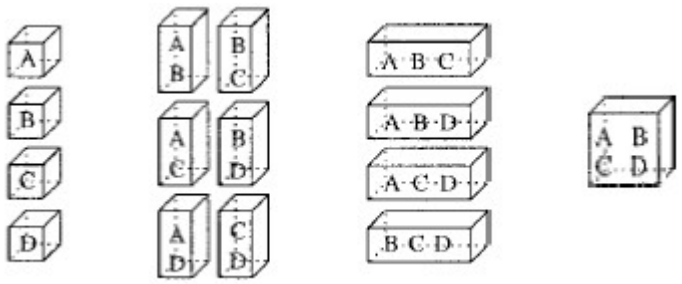


Figura 8: Ejemplo de celdas de un cubo: P(Ca) con Ca = {A,B,C,D} ([Alb02])

Es a través de clases de celdas que los autores manejan el concepto de granularidad y la agrupación de clases de celdas es lo que termina por formar un cubo. Este trabajo no deja de lado patrones de agregación de las métricas de una celda, así como la posibilidad de obtener celdas con métricas derivadas como tampoco la definición de operaciones multidimensionales. Las operaciones multidimensionales se encuentran definidas en base a las clases de celdas y su nivel de agregación.

Ambos trabajos, sobre las dimensiones y sobre los hechos, detallan las propiedades y relaciones inherentes a cada tipo de objeto con sus posteriores formalizaciones. Particularmente, definen con gran nivel de detalle y formalizan el concepto de *Celda*. Asimismo, demuestran que, a nivel conceptual, es esencial que el manejo de *Jerarquías de Agregación* sea "explícito". De esta manera logrará una mayor cercanía al modelado del mundo real y como consecuencia, un mejor entendimiento del usuario de negocio, para su posterior validación, del modelo multidimensional propuesto. Es decir, cuanta más información a nivel explícito se pueda manejar en un modelo conceptual, mayor será su acercamiento al mundo real.

El resultado de estos trabajos se ve plasmado en su propuesta YAM^2 [Alb02] de metamodelo en tres niveles, *Superior, Medio* e *Inferior*. Esta división la han incorporado los autores y se encuentra relacionada con el nivel de detalle de los objetos contenidos por cada nivel.

En el nivel inferior se encuentran las *Medidas* y los *Descriptores* o *Atributos de las Dimensiones*.

En el nivel medio se encuentran objetos que representan niveles de agregación de dichos objetos de nivel inferior de forma tal que un conjunto de *Medidas* forman las *Celdas* y éstas a su vez se agrupan en *Cubos* n-dimensionales. Paralelamente, los *Descriptores* se agrupan en *Niveles de Dimensión* que se encuentran relacionados entre sí.

Finalmente, en el nivel superior se encuentra el *Fact* formado por *Celdas* que representan el mismo tipo de *Hecho* y las *Dimensiones* unidas por las *Estrellas*. Esto forma un *Esquema Multidimensional*.

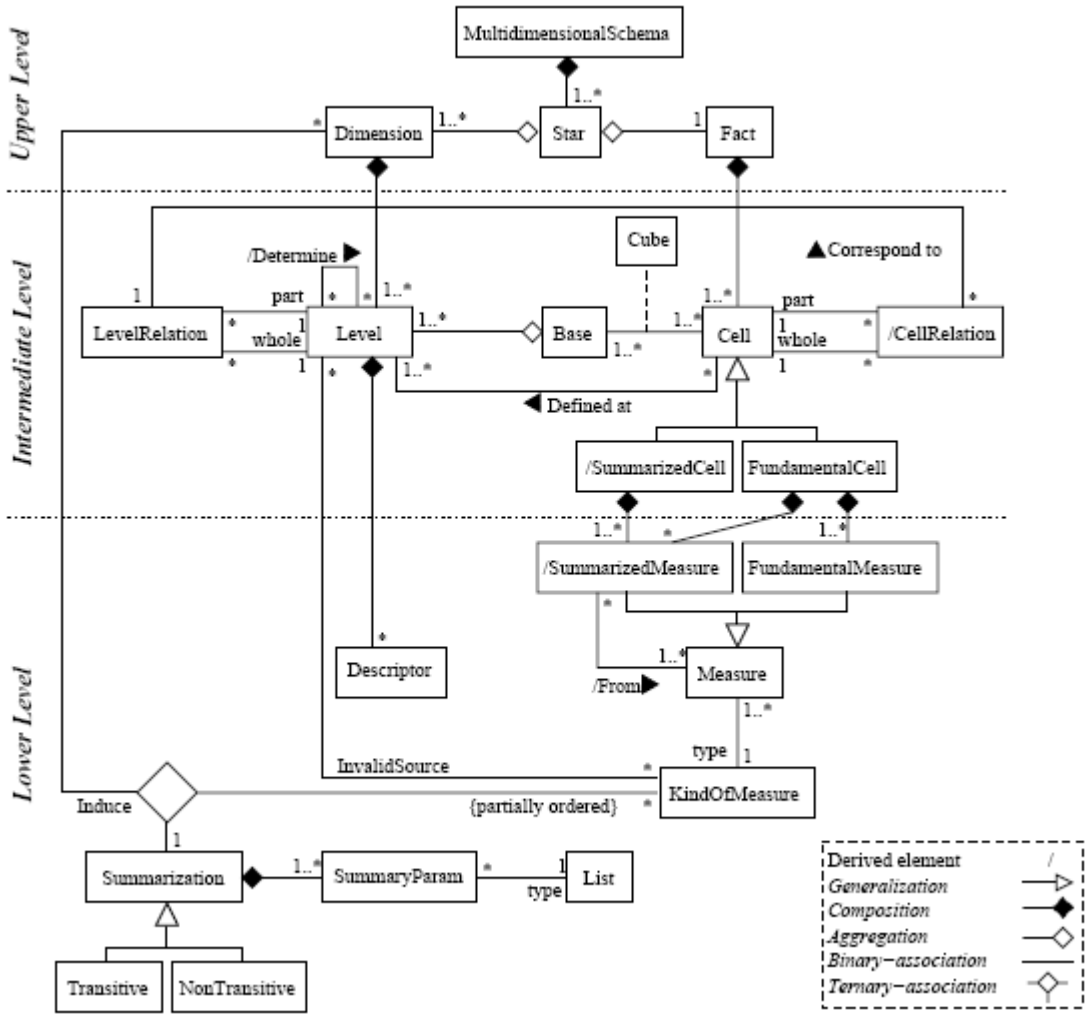


Figura 9: Metamodelo presentado por Abelló en su tesis doctoral: YAM^2 [Alb02]

Un punto a destacar, con referencia a las dimensiones, es que es el único hasta ahora que es capaz de tener una única definición de dimensión y compartirla por medio de diferentes hechos en esquemas multiestrella, evitando así definir la misma dimensión más de una vez y permitir el uso de *"Conformed Dimensions"* [Kim96]. Además es el único capaz de definir más de un rol para una dimensión en cuanto al mismo hecho conectándolos por medio de asociaciones diferentes.

Este modelo resulta ser el más completo de los estudiados hasta el momento ya que considera la mayoría de las propiedades multidimensionales deseables enunciadas previamente.

A pesar de la división en diferentes niveles que propone YAM^2 [Alb02], este es un modelo complejo no solamente para usuarios finales, sino también para diseñadores. Principalmente debido al gran número de relaciones y clases necesarias en el diseño.

Para cerrar con la evolución de los modelos multidimensionales OO consideramos conveniente presentar los últimos trabajos de Trujillo et. Al. y mostrar cómo fue la evolución de su modelo original OOMD presentado en 1998.

Son tres los trabajos que ha presentado como continuación de sus iniciales OOMD y GOLD.

El primero de ellos, [Tru00], presenta un lenguaje de especificación formal OO que permite la generación automática de prototipos para su modelo GOLD presentado previamente. Para lograr transformar un modelo conceptual en un prototipo se focalizaron en la identificación de patrones de diseño y sus relaciones con el modelo de datos. Recordemos que el modelo GOLD considera entre otras las propiedades de métricas derivadas, patrones de agregación, jerarquías múltiples, de calificación. No contempla evoluciones temporales de atributos o jerarquías (dependientes de tiempo). Lo que si presenta GOLD es un lenguaje de definición OO *"Gold Definition Language o GDL"* que será el utilizado para la implementación de dichos prototipos.

Dividieron la identificación de patrones de cardinalidad y de comportamiento. Esta identificación se encuentra estrechamente relacionada con la definición de GOLD [Tru99] para relacionar los atributos dimensionales a través de caminos ARRP que permitían la agregación y desagregación de los mismos, y los caminos ACP que brindaban más información sobre un atributo pero sin permitir su agregación.

Estos caminos definen grafos a íclicos con cardinalidad de 1 a muchos para los ARRP y 1 a muchos o 1 a 1 para los ACP. Este patrón de agregación que poseen los caminos ARRP permite la aplicación de operaciones como *Drill-down* y *Roll-up*, que son típicas operaciones de agregación y desagregación. En esta publicación, los autores introducen dos nuevas operaciones *Combine/Divide* para soportar los patrones de comportamiento de los atributos de caminos ACP. De esta manera, la operación *Combine* permite que pueda visualizar información adicional o de calificación, y *Divide* permite la operación inversa. Concluyen el trabajo con la formalización de las operaciones mencionadas.

El mayor aporte de este trabajo reside en la identificación de patrones que permiten la generación automática de prototipos multidimensionales OO a partir de un modelo conceptual OO que es el GOLD [Tru99].

El segundo de los trabajos [Tru01b] se basa en una investigación a partir de GOLD [Tru99] y de la extensión realizada a UML para soportar propiedades multidimensionales junto con la presentación de la herramienta CASE que soporta el modelado multidimensional conceptual a partir de la implementación de este UML extendido.

Han dividido al modelo conceptual en *Estructural* y *Dinámico*. El modelo estructural cuenta con las definiciones de las dimensiones, atributos, hechos y cubos entre otros mientras que el modelo dinámico representa las necesidades de gestión de los usuarios y por ende las operaciones OLAP.

Ejemplos de cada uno de ellos se muestran en las figuras a continuación.La **Figura 10** representa la parte estructural del modelo conceptual OO para el *Fact* o *Hecho "Venta de Productos"*. Asimismo, se visualizan las extensiones UML realizadas por los autores para soportar las métricas derivadas, los patrones de agregación, y la representación de las jerarquías con su cardinalidad y especializaciones.

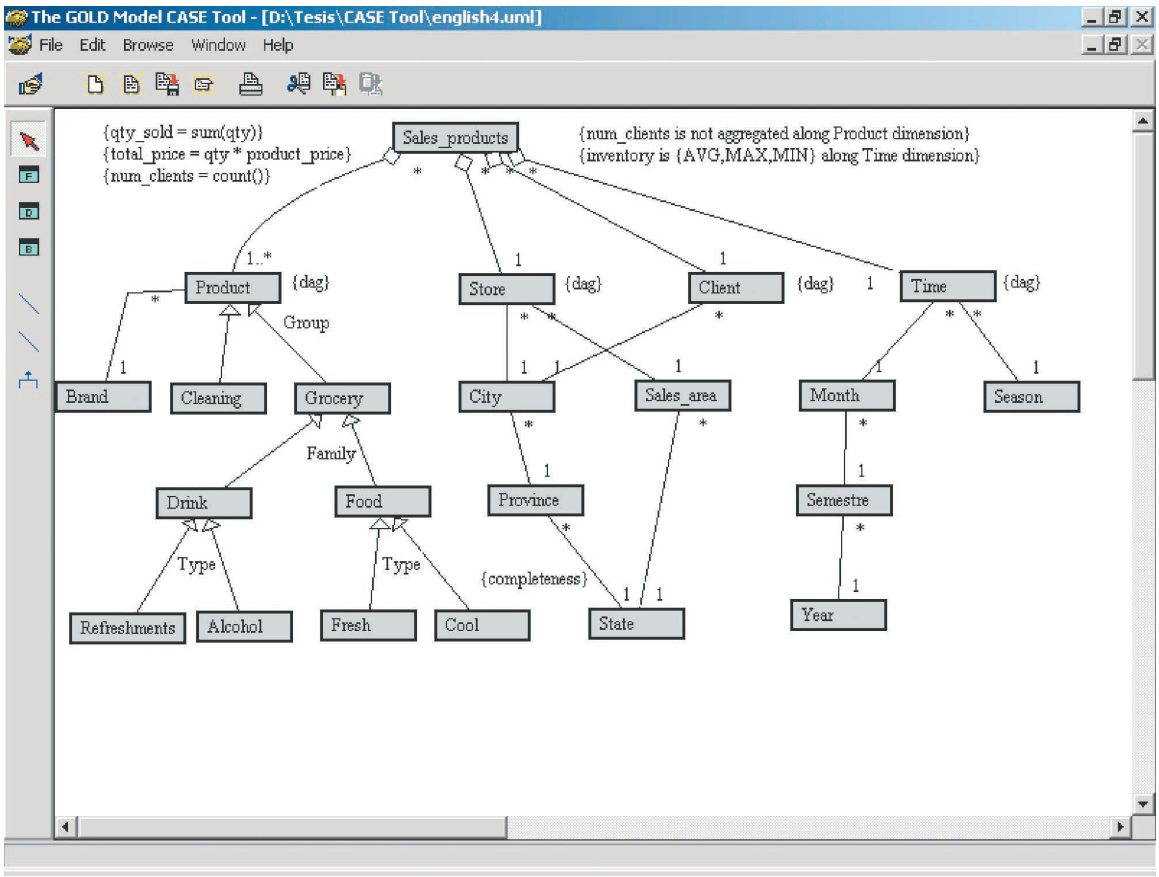


Figura 10: Modelo UML para el hecho "Venta de Productos"

En la **Figura 11** veremos un ejemplo de la parte dinámica del modelo.

Esta imagen representa los requerimientos iniciales del usuario (caminos de análisis) y su respectivo diagrama de estado.

Definitivamente, el avance que han logrado estos autores desde su OOMD hasta la implementación de prototipos y la materialización de una herramienta CASE basada en UML extendido para la representación y formalización de MMCs es de destacar. Son unos de los que más han aportado al estudio del modelado conceptual sobre modelos multidimensionales.

Trabajos recientes

En los últimos años UML se ha convertido en el lenguaje de modelado más utilizado en la industria del software y junto con *"Meta Object Facility"* (**MOF™**) [QVT], provee también la base para *"Model-Driven Architecture"* (**MDA**) [MDA04], que unifica cada paso del proceso de desarrollo del software. Sumado a esto, el surgimiento del *"Common Warehouse Metamodel"* (**CWM**), para estandarizar el intercambio de metadata entre las diferentes herramientas de *"Business Intelligence"* (desde ahora en adelante **BI**, su sigla en inglés), dieron más y mejores herramientas que favorecieron el desarrollo de nuevas propuestas para modelado multidimensional. Es así que paralelamente al desarrollo de nuestra investigación han surgido nuevos trabajos que proponen modelos conceptuales y procesos para el desarrollo del DW basados en estas herramientas.

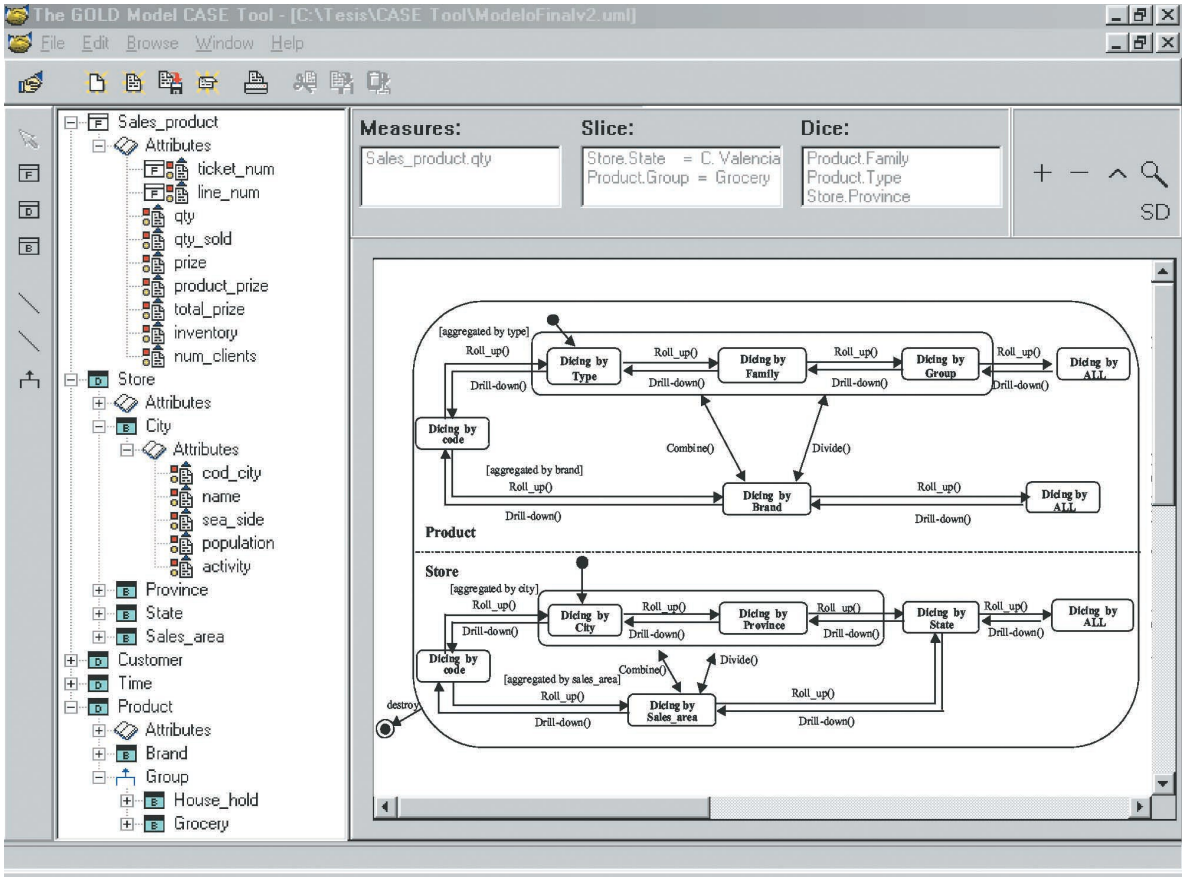


Figura 11: Herramienta CASE desarrollada por los autores del modelo GOLD [Tru99] para generar prototipos. ([Tru01b])

Por ejemplo durante los últimos años un alumno de Trujillo, Sergio Luján Mora, presentó una serie de papers [Tru01], [Mor02], [Med02], [Mor02b], [Mor02c], [Mor03], [Mor04], [Mor04b] en donde desarrolla una propuesta para **modelado multidimensional orientada a objetos** utilizando una extensión de UML. Su trabajo fue evolucionando a partir del modelo GOLD [Tru99] de Trujillo.

La ventaja de esta propuesta es que utiliza diferentes niveles de detalle al igual que YAM^2 [Alb02] pero en este caso utilizando una extensión a los paquetes UML lo cual permite representar modelos extensos a diferentes niveles de complejidad relacionando varios datamarts. El hecho de ser una extensión de UML lo convierte en un modelo fácilmente estandarizable pero los autores no modelan las operaciones multidimensionales sobre el mismo.

Otro exponente de esta evolución es [Maz05], un trabajo de Mazón, Trujillo, Serrano y Piattini: *"Applying MDA to the Development of Data Warehouses"* publicado en 2005. En este trabajo los autores se basan en la falta de propuestas completas que provean a los consultores de DWH con métodos integrados y estándar para el diseño de todo el DW (desde los procesos ETL hasta las consultas) y en el nuevo estándar para el desarrollo de ciclos completos de aplicaciones (comprendiendo desde el diseño, implementación, integración hasta la administración) basado en modelos de desarrollo de software, denominado "*Model Driven Architecture"* **(MDA)** [Fra03], [Klep03], [Mel04], [MDA04] -, para redefinir el proceso completo del DWH en base a este nuevo estándar MDA.

Denominaron a esta aproximación *"**M**ulti**D**imensional **M**odel **D**riven **A**rchitecture"* (**MD^{2}A**), es decir, la aplicación del framework MDA a una de las etapas del desarrollo del DW: la del modelado multidimensional.

Este nuevo estándar separa la especificación de la funcionalidad de un sistema en un "*Modelo Independiente de Plataforma" o "Platform Independent Model"* (**PIM**) de la especificación de la implementación de dicha funcionalidad en una tecnología específica en un *"Modelo Específico de Plataforma"* o *"Platform Specific Model"* (**PSM**). Paralelamente brinda un *"Modelo Computacional Independiente"* o *"Computacional Independent Model"* (**CIM**) para el modelado de requerimientos permitiendo describir al sistema en relación con su entorno. En consecuencia, MDA brinda el framework no sólo para construir estos modelos formalmente (utilizando UML) sino también para especificar las transformaciones automáticas entre los mismos para obtener el producto final. Las especificaciones de las transformaciones se basan en el estándar de *"Query, View, Transformation"* (**QVT**) [QVT].

Uno de aportes más importantes de esta propuesta, según sus autores, radica en que la utilización de este framework permitirá a los desarrolladores de DWH disminuir tiempos y costos de desarrollo al generarse el DWH de forma automática a partir de los PIM's definidos. Otros beneficios mencionados radican en el cambio de perspectiva ya que los desarrolladores cambiarían el foco de los aspectos lógicos y técnicos de un DWH a los aspectos conceptuales de cómo resolver en forma eficiente y eficaz los desafíos de las necesidades del negocio a modelar. Portabilidad es otro de los beneficios mencionados por los autores consecuencia de la independencia de los PIM´s de la tecnología subyacente.

Este framework divide el proceso desarrollo de DWH en cinco niveles:

- Nivel Fuente: Para la definición de las Fuentes de Datos (Ej. Sistemas OLTP).

- Nivel de Integración: Donde se definen los mapeos entre fuentes de datos y el DWH. (Se corresponde con los procesos de ETL).

- Nivel de Warehouse: Donde se define la estructura del DWH siendo el modelado multidimensional la base para la definición de esta estructura.

- Nivel de Personalización (Customization): Donde se definen estructuras especiales como los cubos de datos que son utilizadas por las aplicaciones de usuario final para acceder al DWH.

- Nivel de Aplicación: Donde se definen las implementaciones de las aplicaciones para los usuarios finales generalmente conocidas como OLAP, data mining u otras aplicaciones de reportes.

Cada uno de estos niveles puede ser analizado a través de tres niveles según MDA [MDA04]. Estos niveles como los mencionamos anteriormente son: CIM para la definición del punto de vista de los requerimientos del DWH, PIM's para las definiciones del punto de vista conceptual del DWH (Ej. El PIM de Aplicación se basa en el CIM y en el PIM de Personalización) y PSM para la definición de su implementación técnica sea ROLAP, HOLAP o MOLAP.

Sólo con la definición de los modelos de CIM y PIM se deriva automáticamente el resto de los modelos aplicando las respectivas transformaciones. La definición de las transformaciones entre PIM's y PSM's es uno de los temas de discusión todavía en MDA.

Es por este motivo que los autores dedican su trabajo en definir las transformaciones entre PIM y PSM sin poner mayor atención a la definición de los mismos.

El enfoque más generalista en la evolución de los metamodelos multidimensionales lo podemos atribuir a CWM.

Uno de los aspectos más importantes del DW es la metadata: utilizada para construir, mantener, manejar y usar los datos del DW. Desafortunadamente la proliferación de herramientas de manejo de datos y de análisis resultó en tantas representaciones distintas de metadata como herramientas hay en el mercado lo cual implica que no es posible mantener un único repositorio que implemente un único metamodelo para toda la metadata de la organización. Lo que se necesita en este caso es un estándar para intercambio de metadata.

CWM es una respuesta a estas necesidades ya que provee un framework para representar metadata acerca de orígenes y destinos de datos, transformaciones, análisis, procesos y operaciones que crean y manejan datos del DW.

Las siglas se traducen como *"Common Warehouse Metamodel"*, un estándar abierto para el intercambio de metadata específica de DW.

Su principal propósito es especificar interfaces que pueden ser utilizadas para facilitar el intercambio de metadata de warehouse y de inteligencia de negocio entre herramientas, plataformas y repositorios de metadata de DW en ambientes distribuidos y heterogéneos.

CWM se basa en tres estándares del OMG:

- UML (Unified Modeling Language): un estándar de modelado que define un lenguaje de modelado orientado a objetos que es soportado por un rango de herramientas de diseño gráficas.

- MOF (Meta Object Facility): un estándar de repositorio de meta modelado y de metadata. Define un framework extensible para definir modelos para metadata y provee herramientas con interfaces programáticas para almacenar y acceder la metadata en un repositorio.

- XMI (XML Metadata Interchange): un estándar para intercambio de metadata que permite que la metadata sea intercambiada como flujo de datos o archivos con formato estándar basado en XML.

La arquitectura completa ofrece una amplia gama de posibilidades de implementación para herramientas de desarrollo, repositorios y frameworks de objetos.

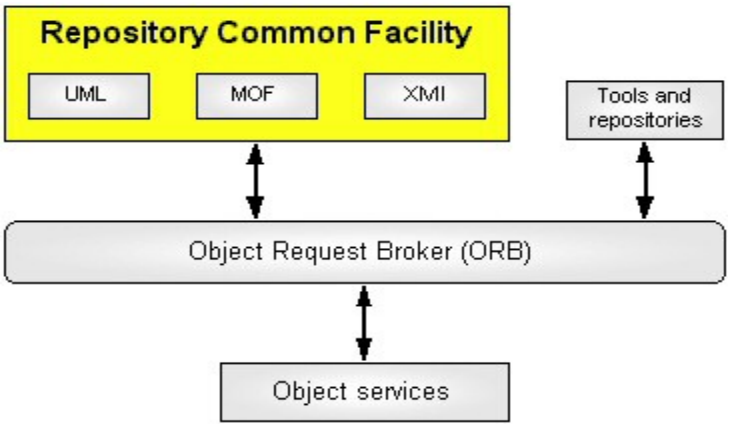


Figura 12: Diagrama representando modelo de arquitectura de CWM (adaptado de [CWM02])

Los aspectos clave de la arquitectura incluyen:

- Una arquitectura de meta modelado de cuatro niveles para manipulaciones de metadata de propósito general en repositorios de objetos distribuidos.

- El uso de notación UML para representar metamodelos y modelos.

- El uso de modelos de información estándar (UML) para describir la semántica del análisis de objetos y el diseño de modelos.

- La utilización de MOF para definir y manipular metamodelos en forma programática usando interfaces CORBA de granularidad fina. Este enfoque aprovecha los fuertes de la infraestructura distribuida de CORBA.

- El uso de XMI para intercambio de metadata basado en flujos de datos.

CWM consiste en un conjunto de sub-metamodelos que representan metadata común al DW en las siguientes mayores áreas de interés a DW y BI.

- Modelo de Objectos (subconjunto de UML).
- CWM Foundation.
- Orígenes de datos relacionales.
- Orígenes de datos de registros.
- Orígenes de datos multidimensionales.
- Orígenes de datos XML.
- Transformaciones de datos.
- OLAP.
- Minería de datos.
- Visualización de la información.
- Nomenclatura de negocio.
- Proceso de warehouse.
- Operación del warehouse.

El metamodelo en detalle

El metamodelo usa estereotipos y valores etiquetados para extender UML con clases del modelo de objetos. Se especifican restricciones con OCL para que puedan ser reforzadas por herramientas gráficas que utilizen UML 2.0.

Dentro de los sub-metamodelos está el de OLAP que define un metamodelo de conceptos OLAP comunes a la mayoría de los sistemas OLAP y provee una facilidad donde instancias del metamodelo de OLAP son mapeadas a estructuras de despliegue además, asegura la navegación a través de jerarquías del modelo lógico y sus varios modelos de recursos.

Pero lejos de conceptualizar el modelado del DW este metamodelo es incompleto en el aspecto de modelar las propiedades deseables para un DW enunciadas anteriormente:

- Una jerarquía sólo puede estar contenida en una dimensión, lo cual es muy limitante y evidentemente quita flexibilidad al modelo.

- *Medida* es un caso particular de la clase *Dimensión* lo que consideramos una gran falencia desde el punto de vista conceptual ya que se pierde de modelar el comportamiento especial de los hechos y medidas que se diferencian claramente del de una dimensión, así como también los patrones de agregación que constituyen un concepto fundamental en cualquier herramienta OLAP.

- No modela versionado de dimensiones ni jerarquías.

- Agrega *Time* como dimensión especial pero no le confiere propiedades ni operaciones especiales.

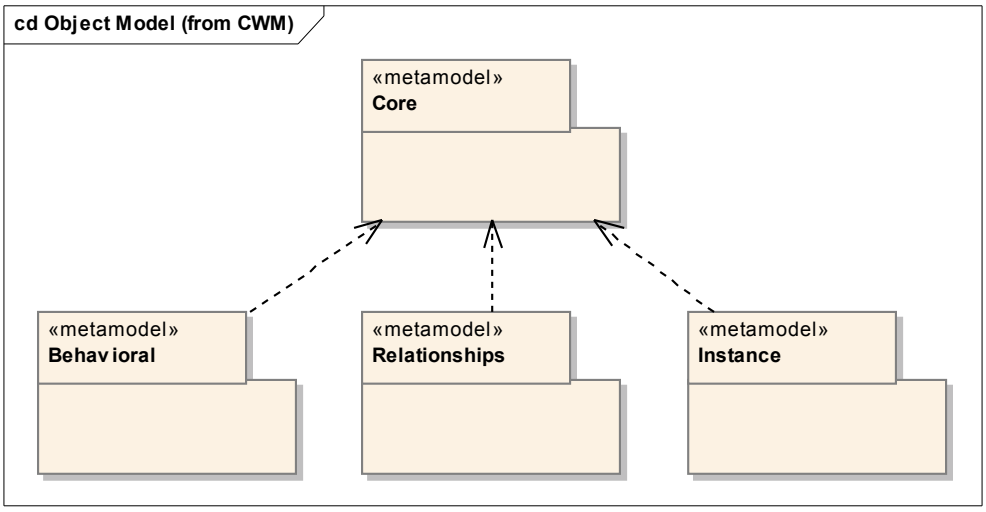


Figura 13 : Modelo de Objetos de CWM (adaptado de [CWM02])

También mezcla conceptos de interfase como el representado en la clase *CubeRegion* como se puede apreciar en la **Figura 14** y de bajo nivel como el representado en la clase *CubeDeployment* en la misma figura.

Todas estas falencias harán que CWM tenga que evolucionar y ser mejorado antes de poder ser utilizado para el efectivo intercambio de metadata multidimensional.

El modelo de CWM es demasiado genérico ya que constituye una enorme construcción sobre el metamodelo de UML con una arquitectura muy pesada dado que su objetivo es estandarizar el intercambio de metadata en todos los pasos del proceso de DW.

Rumbo al metamodelo

En el capítulo anterior vimos que a lo largo del tiempo surgieron una cantidad de técnicas y trabajos tendientes a conceptualizar el modelado multidimensional y conseguir un nuevo avance en la lucha por la independencia de los modelos lógicos/físicos subyacentes.

Los modelos presentados hasta el momento ejemplifican cada uno de los intentos de modelado multidimensional.

Vimos que algunas propuestas hacen más hincapié en el modelado orientado a objetos mientras que otras lo hacen en el plano formal y/o en el modelado de operaciones más complejas para la manipulación del cubo.

Otras encaran el problema desde el punto de vista de la carga y hacen hincapié en el modelado del almacenamiento de los datos históricos para reflejar los constantes cambios del negocio que impactan en las diferentes jerarquías dimensionales del datawarehouse.

Por otra parte observamos dos líneas constantes: una iniciada por Trujillo y Abelló y la otra por Nguyen. En la dos primeras los autores evolucionan hasta llegar a lograr formalizar con un alto grado de completitud la generación de modelos multidimensionales ya sea por medio de un lenguaje de generación de prototipos como es GDL en el caso de Trujillo con su modelo GOLD [Tru99] o por medio de un metamodelo como es el caso de Albelló con Yam^2 [Alb02]. Inclusive Trujillo llega a implementan una herramienta CASE que permite generar modelos multidimensionales lo cual resultaría muy útil para su aplicación comercial.

A Nguyen resulta importante nombrarlo porque es la primera publicación que propone meta modelar el datawarehouse y define las características de su modelo tanto con notación formal como utilizando UML que es un lenguaje estándar de modelado, aunque su modelo resulta ser incompleto con respecto a las propiedades deseables enunciadas capítulos atrás como el modelado de patrones de agregación para las métricas o versionado de dimensiones, además de ser una solución exclusivamente teórica.

Coincidimos ampliamente con el análisis hecho por Albelló de las dimensiones y los hechos pero como ya mencionamos previamente este modelo es demasiado complejo para ser utilizado y convertirse en estándar.

En cuanto a los últimos trabajos estudiados, el metamodelo presentado por Luján Mora es adecuado para ser utilizado en herramientas comerciales porque está definido a partir de una extensión de UML, lenguaje de modelado estándar dentro de la industria del software. Una falencia de este modelo es que no incluye el modelado de las operaciones multidimensionales.

A continuación, y a modo de conclusión de esta investigación del estado de arte en modelos multidimensionales, haremos una comparación de los modelos presentados con respecto al modelado de entidades básicas, de las operaciones así como de las características deseables en un datawarehouse enunciadas previamente y a su posibilidad de aplicación práctica:

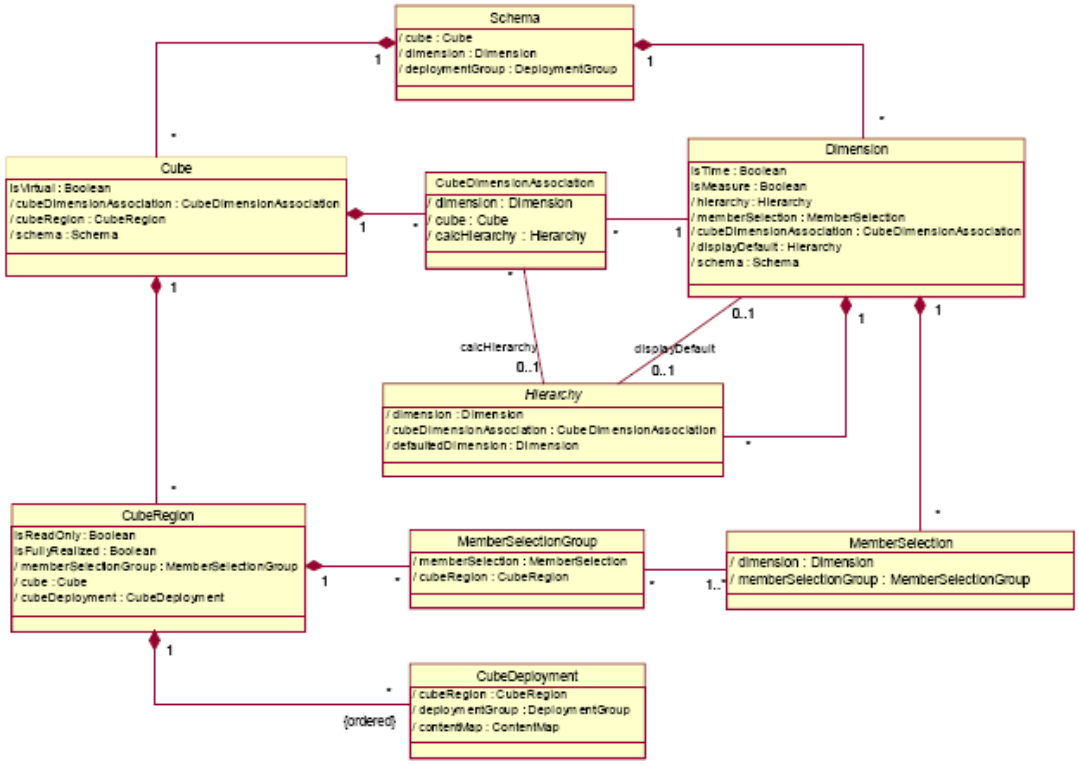


Figura 14 : CWM – Metamodelo OLAP ([CWM02])

- Modelado de entidades básicas del cubo: dimensiones, hechos y jerarquías

GOLD [Tru99] no define ni jerarquías ni niveles de clasificación explícitamente como si lo hace Nguyen, en su lugar modela la navegabilidad de las jerarquías dentro de una dimensión con un grafo acíclico. La dificultad que vemos en utilizar el modelo GOLD [Tru99] en cuanto a la navegabilidad de las jerarquías es que es más difícil de entender porque el grafo es un recurso de implementación más que una herramienta de modelado. Resulta mucho más natural manejarse a nivel conceptual con niveles y jerarquías sin hacer referencia a una estructura de datos particular. Si utilizamos un grafo se corre el riesgo de perder nociones conceptuales como por ejemplo qué atributos dimensionales quedan en cada nivel jerárquico.

En cuanto a O3LAP, que es un modelo físico, encontramos que no modela bien las jerarquías dimensionales porque divide en dimensiones no asociativas cuya relación jerárquica está dada por sub-clasificación como por ejemplo: *Producto – Familia – Grupo – División – Línea*. Y asociativas para las cuales define caminos jerárquicos explícitos por composición: *Region – Pais – Ciudad – Agencia*. Esta división lleva a confusión porque para el modelado conceptual de las jerarquías de una dimensión sólo tiene sentido hablar de dimensiones asociativas. Este error se hace aun más evidente al plantearse qué pasaría en el caso en que se deseara agregar a este modelo jerarquías que evolucionen con el tiempo: Cómo se convertiría una jerarquía originalmente asociativa en una no asociativa?. Por otra parte el modelo presentado no respeta el paradigma de objetos porque sigue la categorización de Yourdon que separa en control y datos.

En el modelo GOLD [Tru99] se confunden atributos de navegación con atributos calificativos de las dimensiones cuando se hace referencia a los caminos ARRP y ACP. Al parecer los autores intentan modelar dos conceptos diferentes como son los de los atributos navegacionales y los calificativos con la misma entidad conceptual "caminos", cuando sólo los ARRP son verdaderos caminos. En conclusión al no modelar los conceptos de jerarquía y nivel en su diseño lo vuelve mucho más complicado a la hora de utilizarlo.
En cuanto al modelado de la clase dimensión en el modelo GOLD [Tru99] el hecho de definir pertenencia, nuevo y borrar está reflejando que toma a la clase dimensión como una representación estática de una dimensión, la "congelan", no provee la funcionalidad para construirla o modificarla.

En este aspecto los trabajos de Albelló y Lujan Mora son completos y concisos.

Y, como ya mencionamos en el capítulo anterior, en CWM una jerarquía sólo puede estar contenida en una dimensión, lo cual es muy limitante y quita flexibilidad al modelo. Además modela *Medida* como un caso particular de *Dimensión* lo cual constituye una gran falencia desde el punto de vista conceptual ya que se pierde de modelar el comportamiento especial de los hechos y medidas que se diferencia claramente del de una dimension como lo son los patrones de agregación para una medida. Tampoco modela versionado de dimensiones ni jerarquías.

- Modelado de consultas

Tanto Trujillo como Albelló definen formalmente operaciones sobre el modelo multidimensional. Mientras que Trujillo define sólo algunas operaciones básicas como ser *roll-up, drill-down, slice* y *dice*, Albelló se extiende más ampliamente definiendo un álgebra completa y cerrada que incluye operaciones entre cubos. El álgebra de operaciones definida por Trujillo no es cerrada porque para definir el resultado de operaciones aplicadas al Cubo introduce el concepto de Vista que, si bien comparte propiedades comunes con el Cubo, es una entidad diferente. En los trabajos de Luján Mora no se ve modelado de consultas y CWM no enfoca el modelado de consultas desde el punto de vista conceptual.

- Propiedades deseables

    - Jerarquías dependientes de tiempo (Historical Truth): Ninguno de los tres modelos presentados las menciona. Estas deberían poder ser modeladas para poder agregar o cambiar jerarquías dentro de una dimensión con el paso del tiempo y poder seguir haciendo comparaciones con dimensiones de jerarquías anteriores. En el caso del modelo GOLD [Tru99] de Trujillo esto se traduciría en modelar los períodos de tiempo en los caminos ARRP. En CWM no se modela el versionado de las jerarquías.

    - Modelado de las Dimensiones Degeneradas: como ya lo mencionamos al describir el modelo GOLD la definición de un *Atributo Clave* (KA) en la clase *Hecho* no sería correcto, siendo en particular una *"Degenerate Dimension"* y no una clave de *Hecho*. En los trabajos de Luján Mora se hace especial mención a esta característica en donde se presenta una metaclase *DegeneratedFact* que es contenedor, en una relación de composición, de una *DegeneratedDimension*.

- Drill Across: En ninguno de los modelos estudiados, excepto YAM^2 y Lujan Mora, se pueden definir operaciones del tipo *drill across* combinando clases *Cubo* a través de las dimensiones que tienen en común o de *pivoting* que permita intercambiar las filas por las columnas.

- Combine / Divide: sobre atributos que no comparten relación alguna (esta es la alternativa que propone GOLD [Tru99] para utilizar los mismos operadores que usa para los atributos calificativos en el camino ACP, para ejecutar operaciones entre atributos que no se encuentran relacionados por ningún camino del grafo), ya que de esta forma, el patrón de comportamiento de estas operaciones no es consistente. Deberían ser creadas dos operaciones más, como *EXPAND* y *COLAPSE* que a partir de la aplicación sobre dos atributos que no comparten relación alguna se obtenga una Vista con la agregación/desagregación de las métricas por ambos atributos. YAM^2 lo menciona pero no lo implementa y Nguyen no hace referencia a esta funcionalidad, por lo que entendemos que sería parte de la funcionalidad a ampliar.

- Autorización, seguridad y perfiles: En los últimos años el acceso a la información se ha tornado un tema clave en las organizaciones. Con respecto a esto vemos que ninguno de los tres modelos analizados propone una solución para esto. En el capítulo IV veremos una alternativa propuesta por Trujillo y Piattini para el modelado multidimensional de la seguridad utilizando UML más OCL.

- Métricas Calculadas: GOLD y Albelló presentan esta funcionalidad, la cual debería ser ampliada en el modelo de Nguyen.

- Aplicación practica

Los propulsores de GOLD propusieron una extensión a UML y llegaron a desarrollar una herramienta CASE para construir modelos pero el modelo está especificado en una extensión propietaria del lenguaje formal GDL lo cual implica pobres posibilidades de ser utilizado, probado y aprobado por una gran comunidad de usuarios ya que solamente se pueden entender las características del modelo a través de los ejemplos presentados.

YAM^2 presenta un metamodelo conceptual completo, excepto por las propiedades deseables que no cumple anteriormente enumeradas, y está especificado mediante una extensión de UML por lo tanto sería simple definir una herramienta CASE que genere modelos. Quedaría pendiente en este caso la definición de las operaciones multidimensionales a partir del metamodelo.

En cuanto a los trabajos más recientes podemos destacar que Luján Mora define un metamodelo conceptual por medio de un perfil de UML lo cual lo hace inmediatamente utilizable en herramientas CASE existentes en el mercado – todas aquellas que utilizan UML 2.0 – para generar modelos. En este caso también faltaría desarrollar un modelo de consulta.

En conclusión en el camino que hemos recorrido hasta aquí va quedando claro que ninguno de los modelos presentados anteriormente es completo en el sentido de aproximarse a un estándar para modelado multidimensional. Con estándar queremos decir que pretendemos un modelo que pueda ser utilizado tanto para generar modelos

como para permitir consultarlos. Buscamos un modelo que permita manejar los cambios históricos de manera simple y transparente y que sirva a cualquier dominio de aplicación. Todo esto nos llevó a plantear que lo que la industria necesita realmente es un metamodelo que sea al datawarehousing lo mismo que el modelo E/R es a las bases de datos relacionales hoy día de manera que cualquier herramienta multidimensional pueda ser implementada utilizando el mismo.

Para ello concluimos que las diferentes propiedades expuestas sobre los modelos estudiados servirán a los efectos de formar la base del Metamodelo Conceptual Multidimensional OO que proponemos a continuación.

OODW - Un metamodelo multidimensional OO

En los capítulos anteriores introdujimos los conceptos básicos relacionados con OLAP, recorrimos el camino de los modelos multidimensionales enfatizando en la evolución de las propuestas orientadas a objetos como un paso más en la búsqueda de mejores técnicas para modelado multidimensional. Analizamos pros y contras de cada modelo y concluimos en la necesidad de la existencia de un metamodelo para que el modelado orientado a objetos de un datawarehousing pase la barrera experimental y se convierta en un modelo formal que pueda ser aplicado a cualquier caso práctico con sólo instanciarlo.

En este marco se desarrolló la presente tesis que pretende ser el siguiente paso natural en la evolución en la búsqueda de mejores técnicas de modelado para bases de datos multidimensionales a través de la presentación de un metamodelo que pueda capturar los conceptos puros del mundo multidimensional y que pueda ser instanciado en cualquier dominio de aplicación.

Optamos para la definición de nuestro metamodelo la utilización de un lenguaje de modelado estándar, UML, debido a que es masivamente aceptado como el lenguaje de modelado OO. La especificación de UML [UML03] define: *"UML es un lenguaje para la especificación, visualización, construcción y documentación de artefactos para sistemas de software, modelado de negocios y sistemas fuera del dominio del software. Representa una colección de las mejores prácticas de ingeniería que probaron ser exitosas en el modelado de sistemas de gran complejidad".*

Define un conjunto de diagramas gráficos:

- Diagrama de Caso de Uso.
- Diagramas de Clases.
- Diagramas de Comportamiento.
- Diagramas de Implementación.

Consideremos que el uso de UML como el lenguaje de modelado es la mejor aproximación en este momento por las siguientes consideraciones:

- UML sigue el paradigma OO que ha probado ser semánticamente más potente que otros paradigmas ya que se encuentra más cercano a la concepción del usuario [Alb01].
- Es un lenguaje popular en el ámbito de la ingeniería de software, por lo que minimizará la necesidad de aprender nuevas notaciones o metodologías.
- Es un estándar del *Object Management Group* (**OMG**) que unifica muchos años de esfuerzos en el análisis y diseño OO.
- UML ha sido masivamente aceptado por la comunidad científica y se ha ubicado como el "lenguaje dominante de modelado".

A continuación entonces presentamos el metamodelo multidimensional orientado a objetos: OODW.

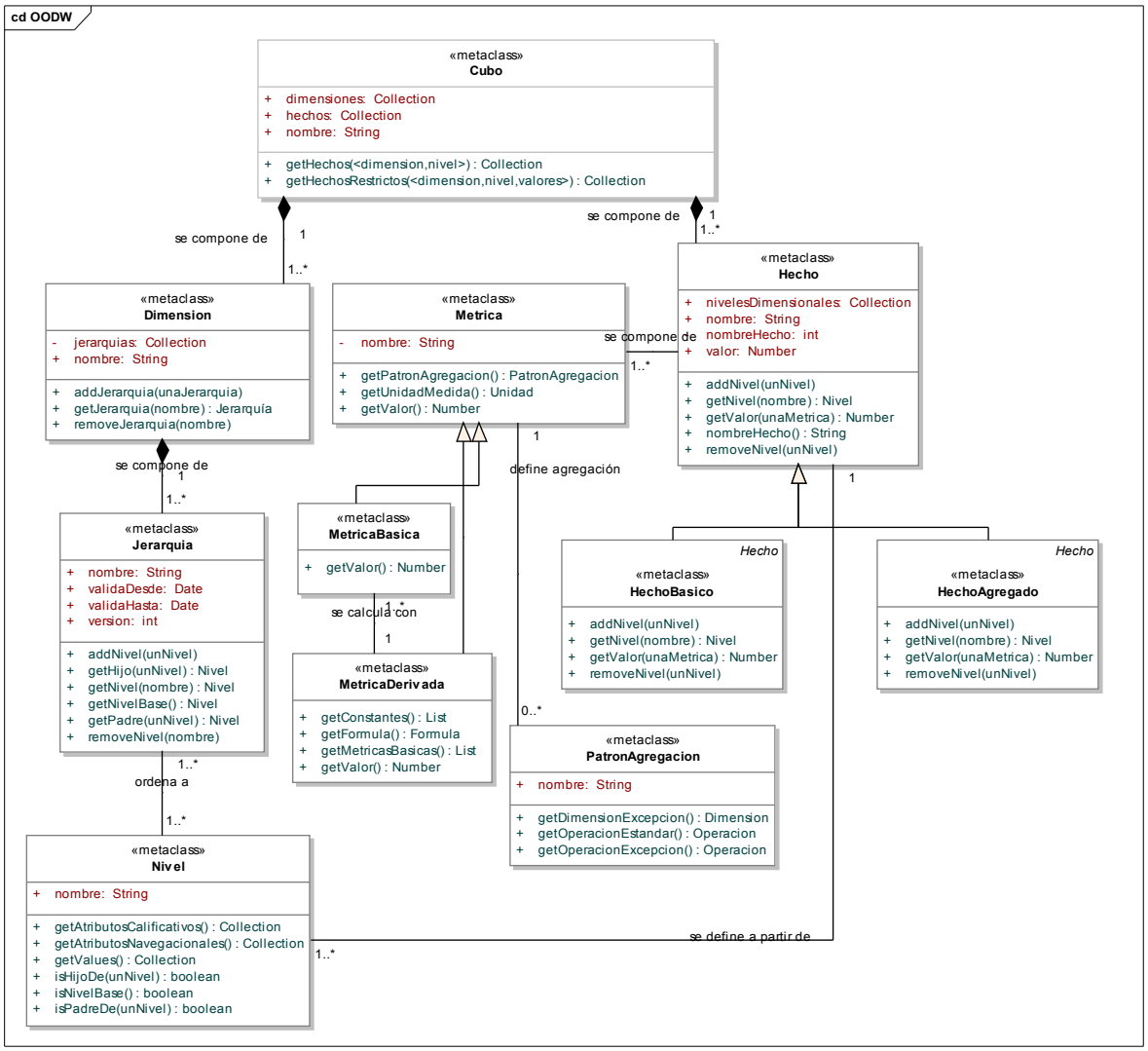


Figura 15: OODW - Metamodelo multidimensional orientado a objetos propuesto

Ya a simple vista se puede observar que OODW soporta las propiedades deseables para un modelo multidimensional, expuestas en capítulos anteriores, como la posibilidad de permitir definir al analista del negocio jerarquías de dimensión explícitas y múltiples, o reflejar la evolución natural del datawarehouse a partir del versionado de las jerarquías dimensionales para analizar *"verdad actual"* y *"verdad histórica"*.

En cuanto a los atributos de dimensión calificativos –no navegacionales- estos se encuentran modelados en la clase *Nivel* así como también los atributos derivados.

Métricas derivadas y métricas básicas están modeladas en sendas clases con esos nombres respectivamente. La agregación de dichas métricas está modelada en la clase hecho que se especializa en las clases *HechoBasico* y en *HechoAgregado* para reflejar precisamente que el valor de estos últimos debe calcularse como resultado de la agregación de los niveles antecesores con respecto a las jerarquías a las que

pertenecen los niveles que la componen. Veamos un ejemplo a continuación con un caso sencillo:

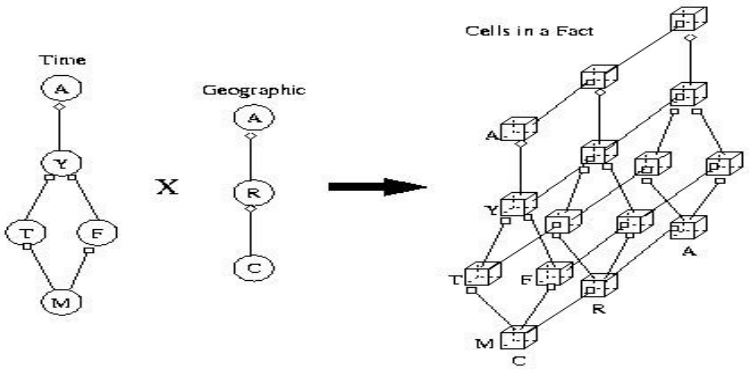


Figura 16: Grafo de celdas en un Hecho con dos dimensiones

Para los patrones de agregación existe una clase que los modela explícitamente y también se ha agregado la relación con la métrica.

Los roles y perfiles no están considerados en este modelo porque dada su vital importancia y su escaso y/o incorrecto tratamiento en las herramientas existentes hemos reservado un capítulo especial en esta tesis aplicando para resolverlo el paradigma de orientación a aspectos.

Consultas predefinidas: las trataremos más extensamente en el próximo capítulo en donde estudiamos las aplicaciones prácticas del modelo.

El metamodelo de la dimensión

En el capítulo anterior comentamos que en general las propuestas presentadas se basan directamente en modelos multidimensionales montados a partir de escenarios particulares lo cual les quita generalidad. Pocos autores proponen la conceptualización a un nivel superior o de metamodelo, como ser Nguyen, Albelló, Pruebe y Luján Mora cuyos modelos fueron oportunamente analizados y criticados.

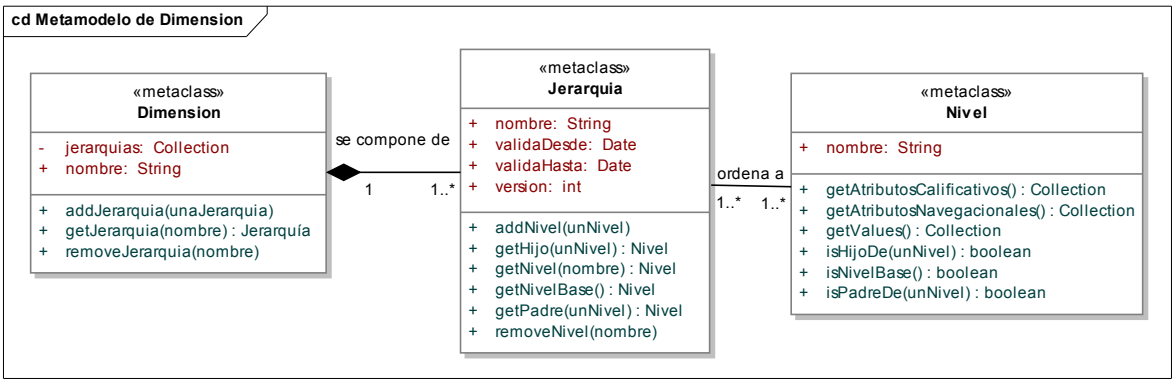


Figura 17: Metamodelo de la dimensión

En este caso presentamos el metamodelo de la dimensión compuesto por la dimensión propiamente dicha: una clase que representa la agregación de múltiples jerarquías conformadas por niveles que guardan una relación de orden y composición entre sí.

Las jerarquías pueden ser múltiples de forma tal de representar las diferentes formas de agrupar niveles en distintos vistas de negocio o caso de estudio según sea requerido. Asimismo, estas jerarquías tienen la propiedad de poder versionarse en el tiempo lo cual permite resolver naturalmente el problema de versionado de las dimensiones planteado en capítulos anteriores como *"Slowly Changing Dimensions"*.

Finalmente, la clase *Nivel* representa en base a una relación "Padre-Hijo" los componentes de las diferentes jerarquías. Un *Nivel* puede participar en más de una jerarquía, en particular, se destaca el *Nivel Base*, que será el nivel que determinará la granularidad -menor nivel de detalle- del *Cubo*.

El metamodelo de las métricas

Una métrica comprende una unidad de medida, y un patrón de agregación, además de su nombre. A nivel de metamodelo no se conoce el valor, ya que sólo tiene sentido al ser combinado con los objetos dimensionales, niveles, que lo definen. Clasificamos las métricas en básicas -no confundir con primarias- que no son resultado de la aplicación de una fórmula si no que se obtienen generalmente de procesamiento de los datos provenientes de las bases de datos operativas y métricas derivadas, que son consecuencia de la aplicación de una fórmula (operadores) sobre métricas básicas.

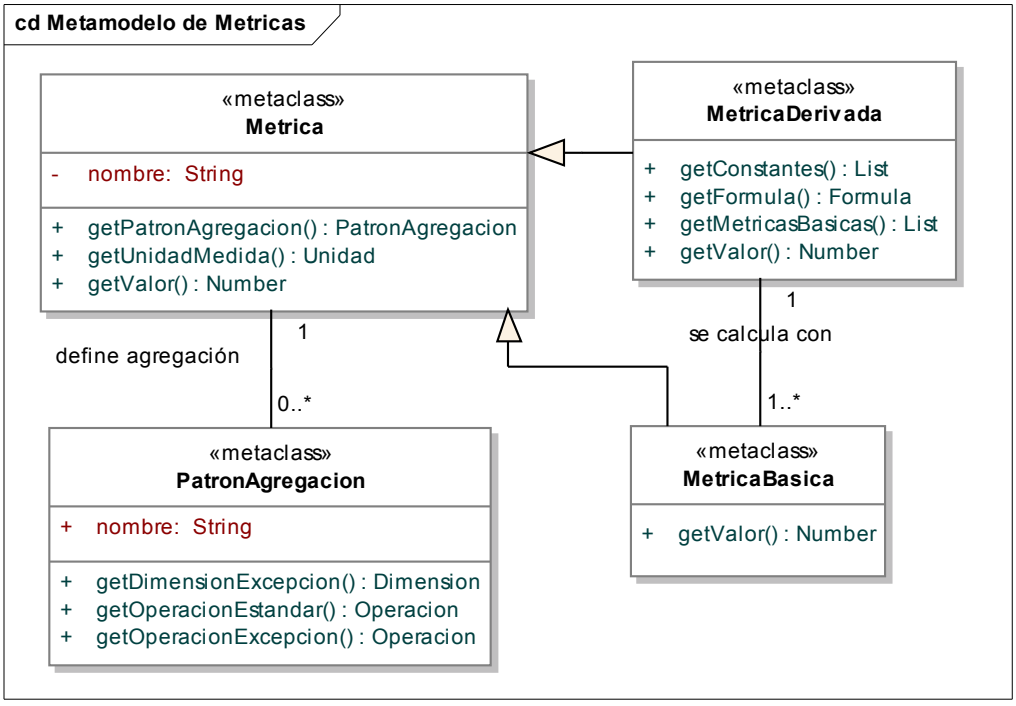


Figura 18: Metamodelo de las métricas

Los patrones de agregación permiten definir operaciones de agregación sobre las métricas. Una métrica normalmente tendrá una operación de agregación estándar (*SUM; MAX; MIN*) para la mayoría de las dimensiones del cubo y una operación de

agregación por excepción (porcentaje, etc.) que se aplicará sólo cuando se está agregando la métrica por la dimensión por excepción.[1]

El metamodelo de los hechos

Algunos modelos consultados, como los de Nguyen y Albelló, consideran pertinente modelar las celdas del cubo, nosotros optamos por obviar este paso porque, como ya expusimos anteriormente, la celda resultaría ser mas bien un recurso de implementación que un concepto importante para el modelado dimensional o con fines prácticos para el modelado de un escenario de negocio.

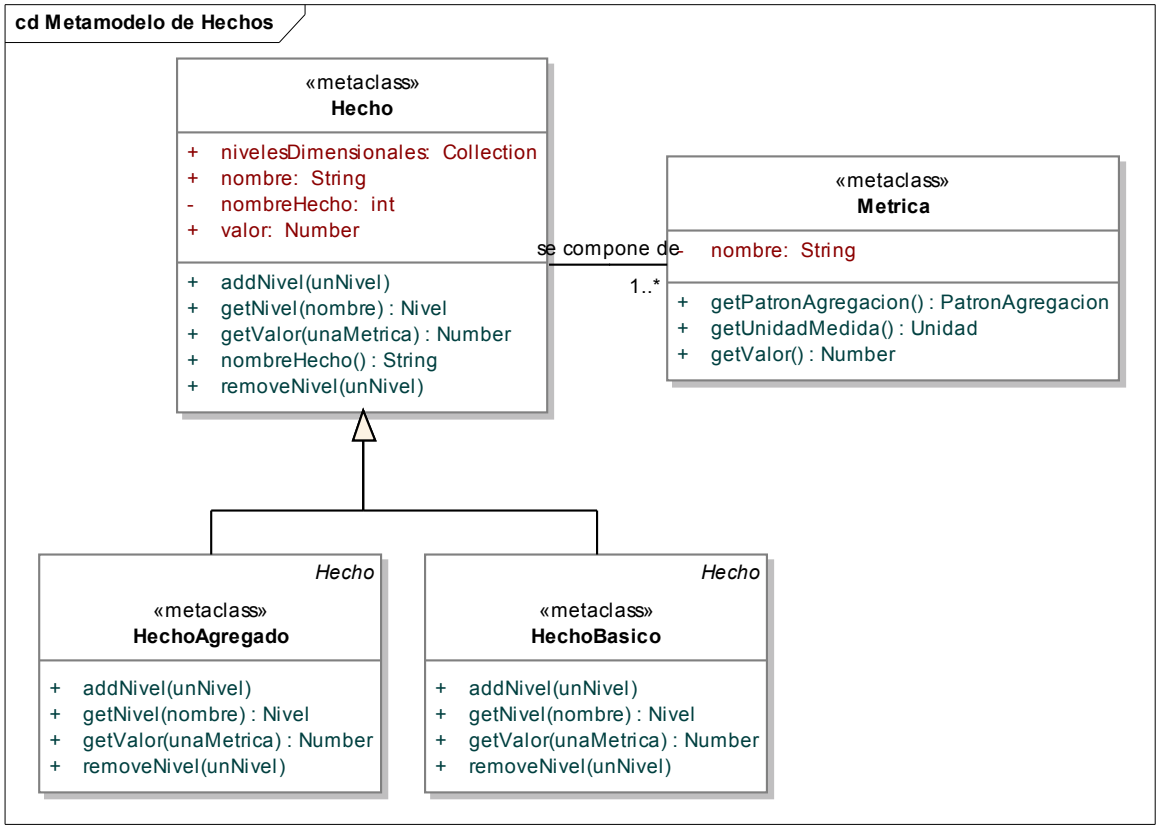


Figura 19: Metamodelo de Hechos

En este modelo la clase *Hecho* representa el hiperplano cuyas coordenadas resultan definidas por un conjunto de niveles dimensionales y cuyos valores son los valores de las métricas para cada combinación de valores de los niveles. Así, cada instancia de *Hecho* resulta ser un punto en dicho plano.

Por ejemplo si la métrica es costo y los niveles *Producto, Mes* y *Provincia*. Entonces los valores individuales para la dicha métrica serían por ejemplo: <c1, agosto, formosa, 20>, <c2, setiembre, cordoba, 30> y así sucesivamente hasta cubrir todo el hiperplano de valores para las tuplas <producto, mes, provincia> que serían, en total: (cp*12*22), donde cp = cantidad de productos y c1,c2…cn = códigos de los productos.

[1] Estas propiedades de agregación de las métricas fueron deducidas a partir de casos reales en los que sólo se crea una agregación por excepción.

Es importante destacar la diferencia entre hechos básicos y hechos agregados. Los primeros resultan de la combinación de los niveles de menor granularidad de las dimensiones y sus valores se obtienen a partir de la carga directa o desde el sistema transaccional. En cambio, los hechos agregados obtienen su valor a partir de la agregación o sumarización de hechos de menor agregación teniendo en cuenta el patrón de agregación de cada métrica.

El metamodelo del cubo

Y por último llegamos al cubo que representa el punto de entrada hacia las dimensiones y los hechos.

A la hora de modelarlo consideramos importante tener en cuenta los posibles usos de este modelo.

Basándonos en la experiencia práctica y en el conocimiento de las falencias de las herramientas existentes hasta el momento en el mercado decidimos que las aplicaciones más importantes podrían darse tanto para construir modelos multidimensionales, por ejemplo a partir de una herramienta que utilice el metamodelo como prototipo, y también para herramientas de consulta. Para esto vimos que en la clase *Cubo* sería necesario proveer operaciones para obtener cierto subconjunto de hechos dadas las dimensiones, sus niveles y en algunos casos restricciones sobre los valores de los mismos.

Así se podrían implementar operaciones de consulta como *roll-up, drill-down, slice, dice*, etc. que veremos con mas detalle en el próximo capítulo.

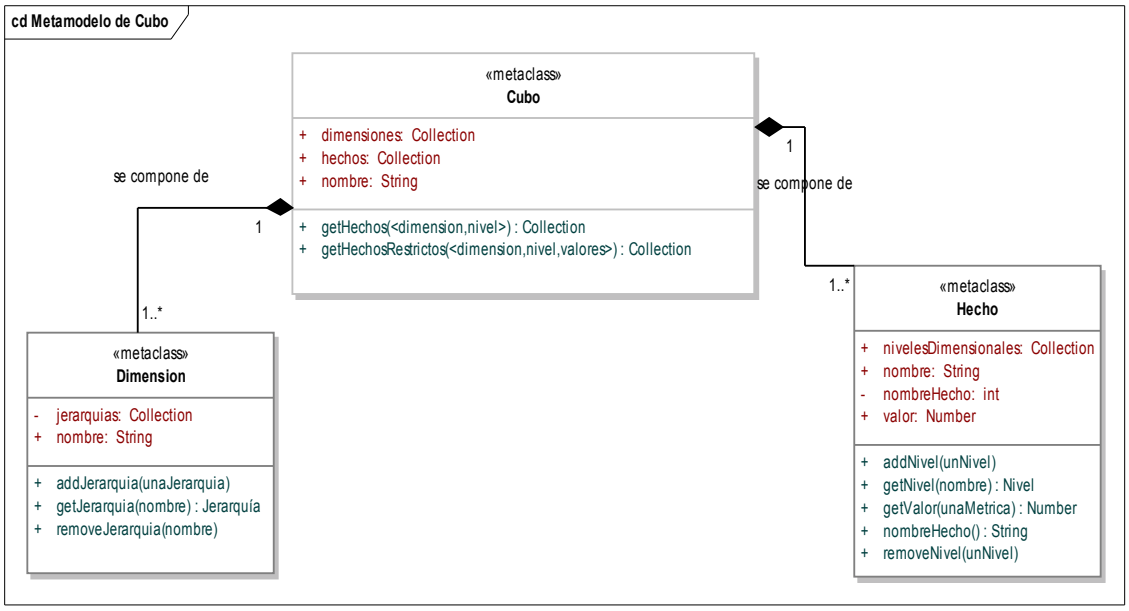


Figura 20: Metamodelo del Cubo

En cuanto a comparación con los modelos anteriores, muy ricos en formalización, si es que GDL facilita la generación de modelos y YAM^2 también con su metamodelo basado en una extensión de UML lo que se propone en este trabajo es útil ya que una herramienta de generación de prototipos podrá derivarse automáticamente de la implementación de las clases del metamodelo.

Estamos proponiendo una opción alternativa a YAM^2 a Luján Mora, presentada a través de un metamodelo especificado en UML sin extensiones lo que la hace inmediatamente implementable en cualquier lenguaje moderno de programación orientado a objetos como ser Java, C++ o Smalltalk. De hecho al documentar los modelos en una herramienta de modelado el código del metamodelo puede ser generado automáticamente en cualquiera de los lenguajes citados sin necesidad de implementar explícitamente una herramienta que a partir del lenguaje de modelado genere el código.

Instanciación de OODW

El objetivo de este capítulo se cumple validando la factibilidad de implementación del metamodelo OODW con una instanciación particular. Se consideró por consiguiente utilizar el caso de estudio como ejemplo idóneo para llevar adelante esta tarea, poniendo énfasis entonces en la usabilidad real del metamodelo presentado.

En el ejemplo de la **Figura 21** se instancia un solo hecho que es *"Pago de Impuestos"* relacionado con sus respectivas dimensiones de análisis: *geografía, contribuyente, impuesto, forma de pago,* etc*.*

En cuanto a las dimensiones a primera vista se puede apreciar la relación de composición entre una dimensión y sus jerarquías. Podemos ver también que en algunos de los casos hay varias jerarquías dentro de una misma dimensión como es el caso de *Tiempo* que tiene dos. Asimismo queda evidenciada en el gráfico la relación de composición entre una jerarquía y sus niveles.

Por otra parte el hecho *Pago de Impuesto* no está relacionado con la dimensión en su totalidad sino directamente con los niveles de las dimensiones que le dan definición al valor de las medidas contenidas en el.

En la **Figura 22** podemos ver una instancia de la clase de hecho relacionada con todos los niveles que lo definen:

En dicho gráfico se puede apreciar claramente la relación del hecho básico *Pago de Impuesto* con cada uno de los niveles base que lo definen.

La instanciación de la metaclase *HechoBase* en la clase *PagoDeImpuesto* es una relación de agregación compuesta por *Métricas Base* y *Derivadas* con respecto a las dimensiones.

Esta instanciación representa la combinación de todos los niveles de mayor granularidad de las dimensiones válidas para el dominio con las métricas para obtener la tupla con los valores del *HechoBase*.

Para la instanciación del *HechoAgregado* basta con tomar cualquier combinación válida de niveles (excepto la representada que es la de mayor granularidad) permitida en el dominio de aplicación estudiado. A modo de ejemplo, en el gráfico, sólo el *HechoBase* es representado.

Cabe recordar que no todas las combinaciones dimension/hecho son válidas o "tienen sentido". Ejemplo: Puede haber *Impuestos* exclusivos para "Grandes Contribuyentes" que son un subconjunto de la población total de contribuyentes del dominio.

Ahora pongamos un poco la lupa sobre las métricas.

Instanciación de la métrica: MontoAdeudado

Esta métrica *MontoAdeudado* es el resultado de una fórmula cuyos operandos son otras clases instancias de *MetricaBasica*. Su patrón de agregación, modelado en la

metaclase *PatronAgregacion*, es el tradicional (SUMA) para cualquier *Dimensión* sin excepciones.

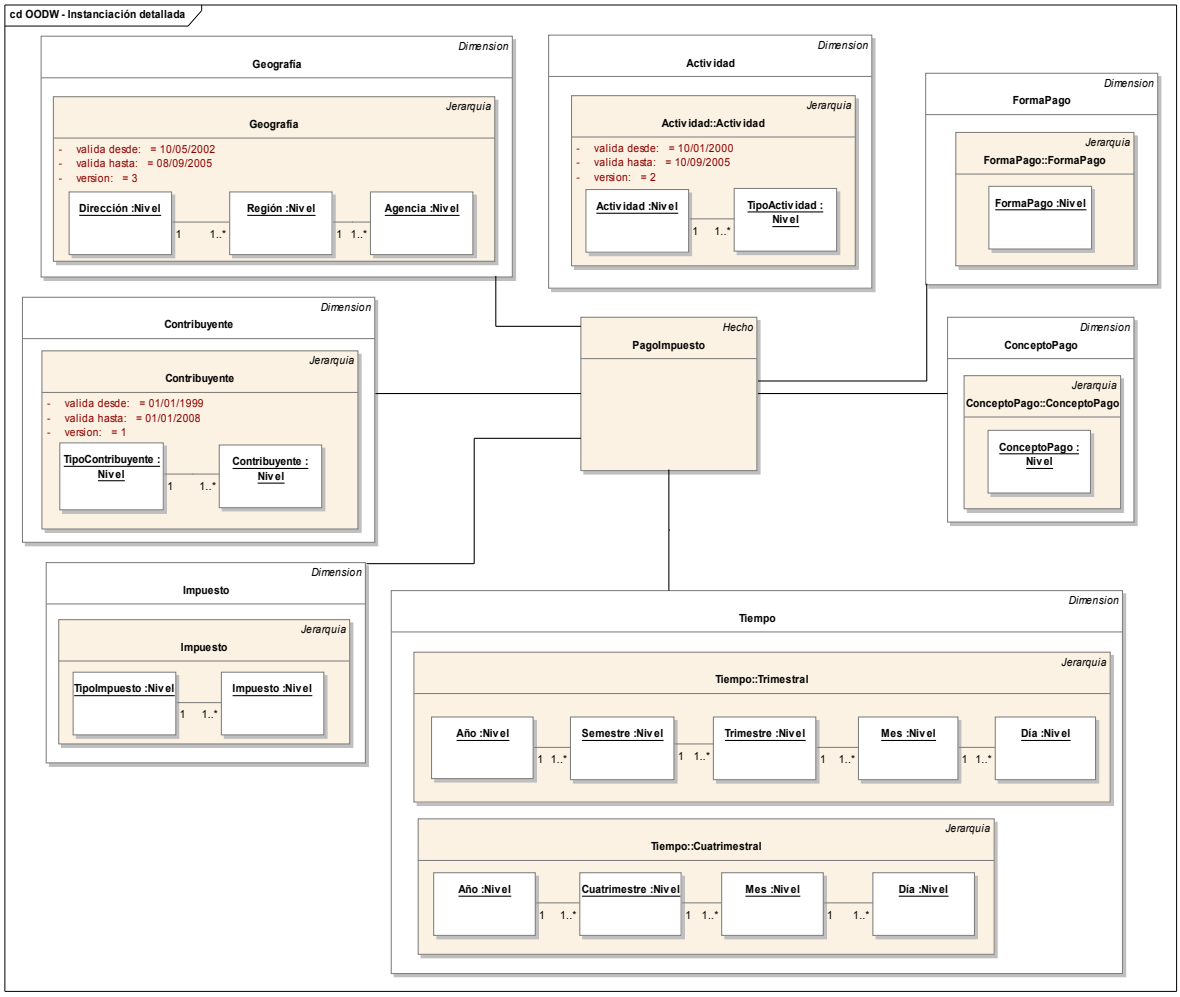


Figura 21: Ejemplo de instanciación de OODW con las dimensiones detalladas

Instanciación de la métrica: CantContribuyentes

Esta métrica representa la *"Cantidad de Contribuyentes del País"* y es instancia de *MetricaBasica* ya que surge de un valor no calculado en este modelo en particular. Esta *Métrica*, a diferencia de la anterior, tiene un patrón de agregación "Semi-Aditivo" ya que *"Cantidad de Contribuyentes"* es resultado de la aplicación de la operación de *SUMA* para todos los niveles excepto para el *TIEMPO* donde se toma *"Cantidad de Contribuyentes"* correspondiente del último día. Su dominio está conformado por niveles de las dimensiones *Geografía* y *Tiempo* ya que, por las características del negocio, son las únicas que tienen sentido para esta medida (conocer *"Totales de Contribuyentes por Actividad"* no tendría sentido debido a que un mismo contribuyente puede tener más de una actividad).

Instanciación del cubo y las dimensiones

La **Figura 25** grafica un solo hecho, en particular un hecho base, que es *PagoImpuesto*.

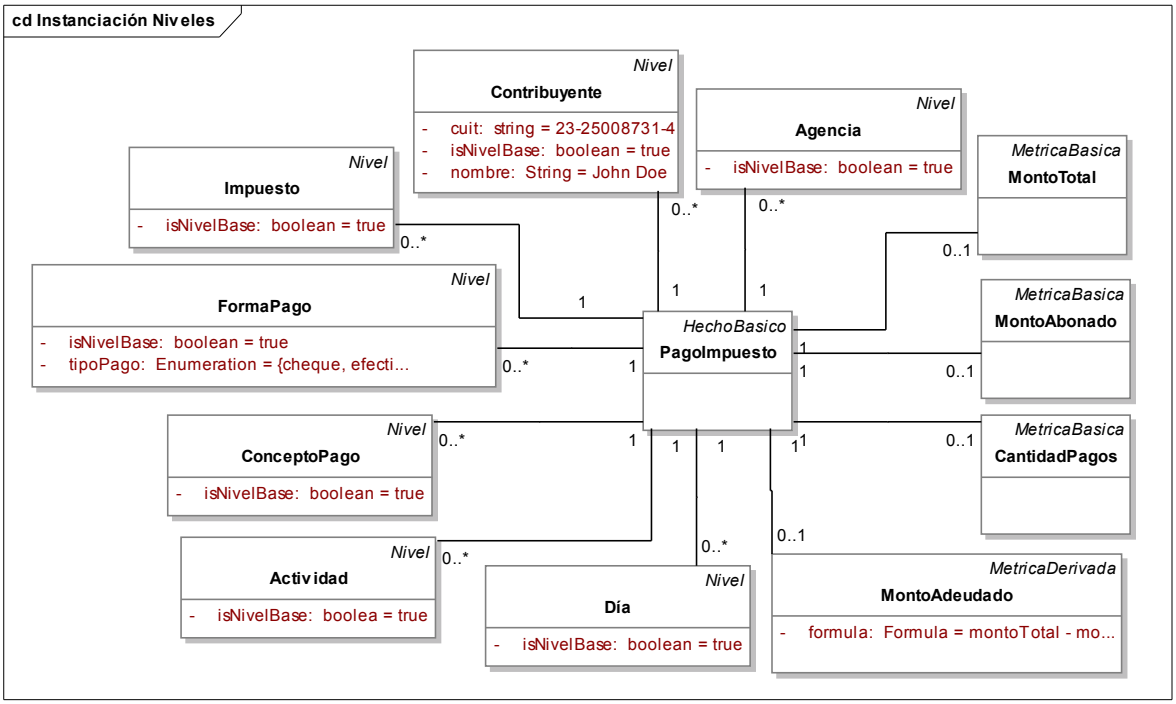


Figura 22: Ejemplo de instanciación de OODW con detalle de niveles base

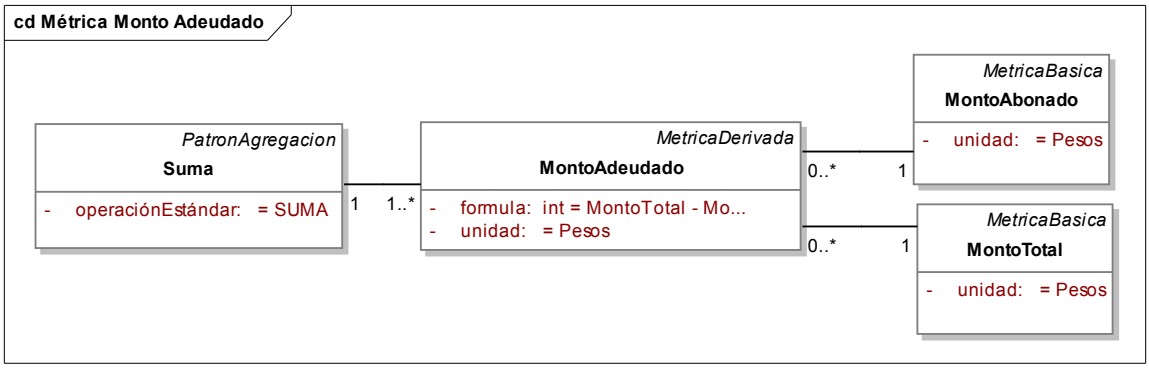


Figura 23: Ejemplo de instanciación del metamodelo de las métricas – Monto adeudado

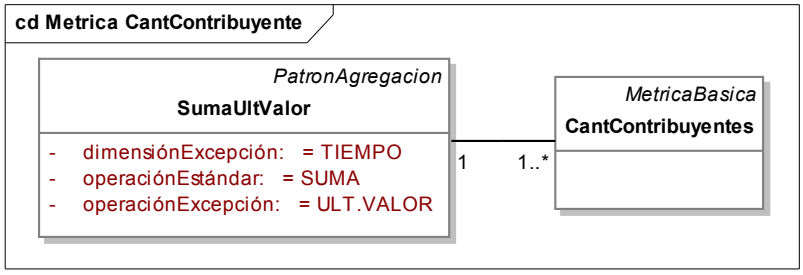


Figura 24: Ejemplo de instanciación del metamodelo de las métricas - CantContribuyente

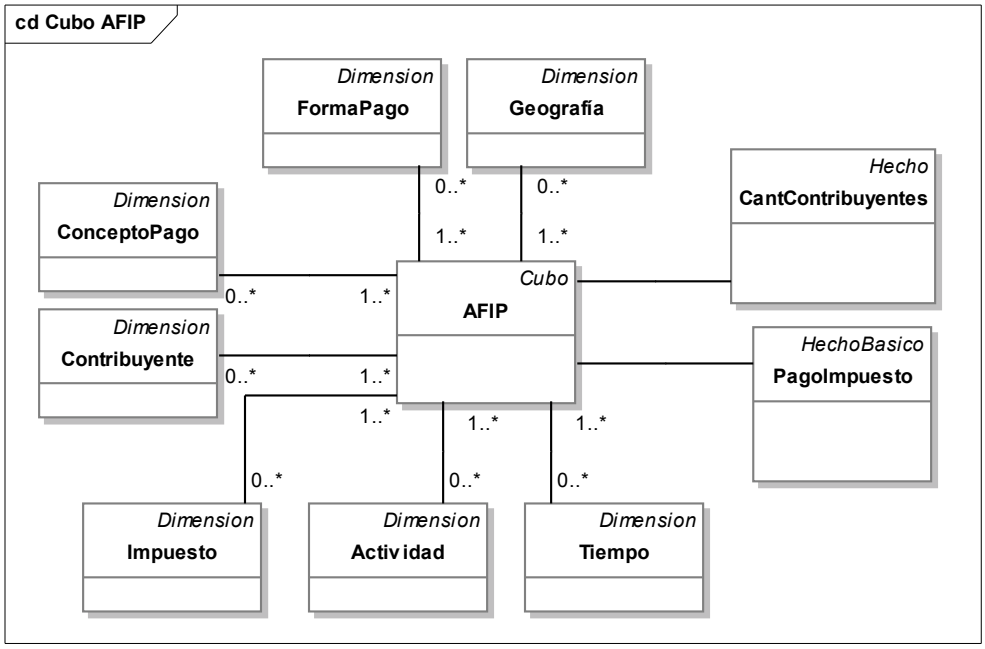


Figura 25: Ejemplo de instanciación del metamodelo del cubo – Cubo AFIP

Aplicaciones de OODW

Nos dedicaremos en este capítulo a presentar diferentes alternativas de aplicación del metamodelo multidimensional. La razón principal de navegar en esta dirección radica en la necesidad de validar la capacidad de aplicación práctica del metamodelo propuesto en aplicaciones reales. La construcción de herramientas a partir del mismo va a permitir la llegada a una comunidad masiva de usuarios que permitirá recibir feedback y validar la factibilidad de aplicación del metamodelo conceptual.

Existen además otros motivos fuertes que llevaron a que nos inclinásemos por presentar estas alternativas y están directamente relacionados con las falencias de las herramientas que podemos encontrar en el mercado hoy en día ([Hyp01], http://www.oracle.com/technology/documentation/warehouse.html). Observamos que principalmente el nicho de construcción de modelos multidimensionales así como también el de herramientas de consulta está bastante poco explotado y las pocas herramientas disponibles obligan al usuario a conocer en parte el modelo lógico/físico subyacente o, peor aun, no permiten cubrir una funcionalidad completa como ya vimos en el caso del versionado de dimensiones, hechos degenerados, patrones de agregación y varias operaciones sobre el cubo que no son todavía soportadas en su conjunto por la mayoría de las herramientas existentes en el mercado.

Construcción de modelos multidimensionales

Sería entonces de gran utilidad para los analistas disponer de una herramienta que permita crear modelos conceptuales multidimensionales de manera simple y permitiendo al usuario abstraerse completamente del modelo lógico/físico. Además ésto permitiría que el analista valide con el usuario el modelo antes de implementarlo lo cual sería de gran utilidad hacerlo en una etapa temprana del diseño del datawarehouse, reduciendo de esta manera el riesgo de re trabajo en etapas posteriores del desarrollo del DW (una de las mayores razones de fracaso en los proyectos de este estilo) [Cut03].

Por todas estas razones es que resultaría extremadamente útil contar con una herramienta que supla esta necesidad. En el caso de que la misma se construya a partir del metamodelo conceptual presentado su utilización sería como se puede apreciar en el dibujo de la **Figura 26**.

En la **Figura 27** exponemos un modelo de diseño de alto nivel de la herramienta de construcción de modelos que utilizaría el metamodelo multidimensional.

Veamos cómo el usuario puede generar un cubo con esta herramienta. En el ejemplo de la **Figura 28** se muestra la construcción de una dimensión y una métrica a los fines de claridad práctica aunque obviamente esto puede extrapolarse al caso de un cubo con múltiples dimensiones y métricas en donde además las dimensiones cuenten con varias jerarquías y las métricas puedan tener uno o más patrones de agregación asociados.

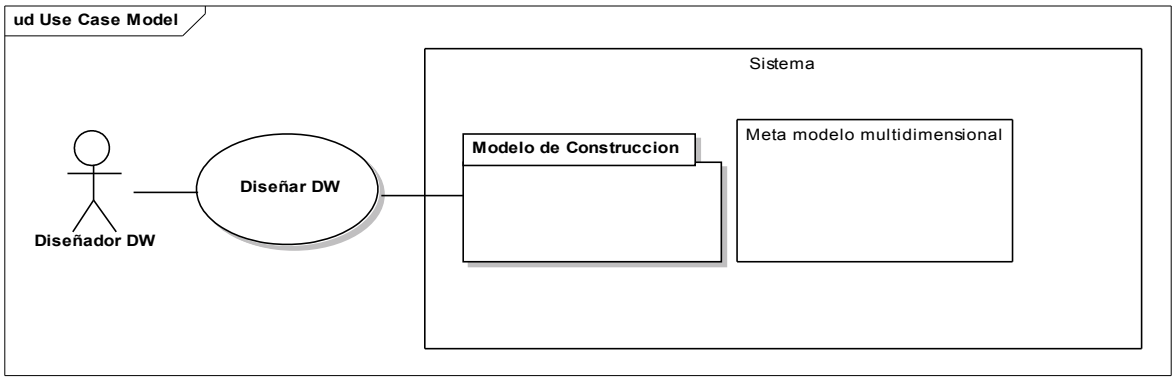


Figura 26: Caso de Uso - Construyendo un datawarehouse

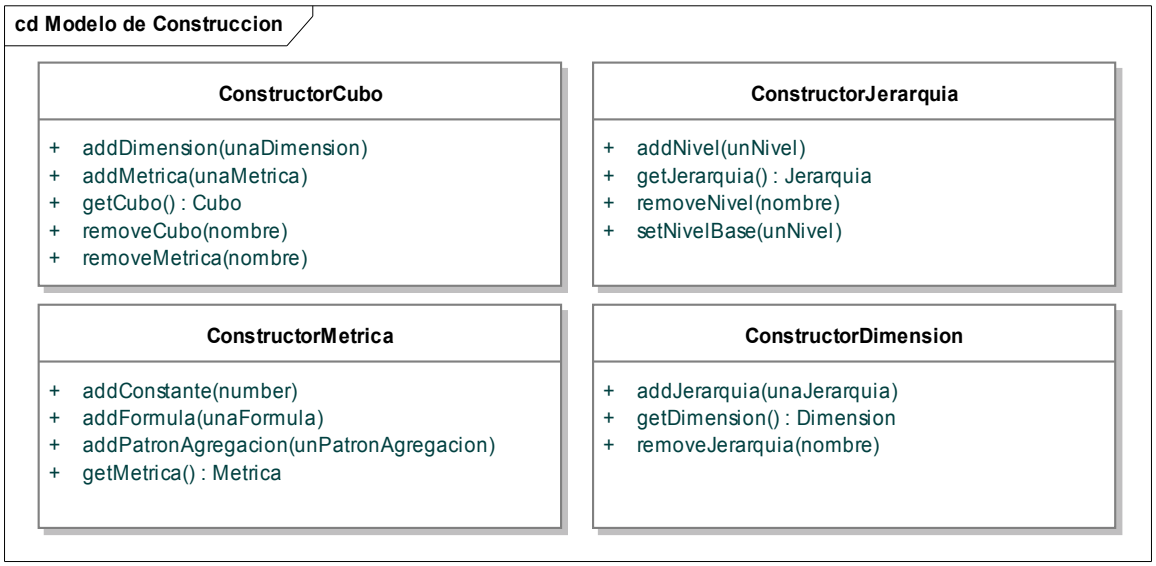


Figura 27: Diagrama de clases - Modelo de Construcción

Consulta de modelos multidimensionales

Sería muy útil también que las herramientas de consulta pudieran valerse del metamodelo multidimensional para poder independizarse totalmente de los modelos lógicos y/o físicos subyacentes.

Para ello es que presentamos también el modelo de consulta basado en el metamodelo multidimensional expuesto anteriormente. En la **Figura 29** se presenta un modelo de diseño que se puede utilizar como base para implementar una herramienta de consulta multidimensional.

En el modelo presentado se puede observar que una consulta puede estar compuesta de condiciones, excepciones y filtros. Una condición se establece sobre una medida determinada restringida por una dimensión. Por ejemplo un ranking: *"mostrar top 10 agencias que más recaudan"*. En este caso la condición es mostrar la medida recaudación por la dimensión agencia. Una excepción se establece también sobre una medida, representa un semáforo. Por ejemplo: *"Mostrar resaltados los casos en que monto abonado es menor a 10"*. Por último un filtro se establece sobre una dimensión

o nivel de dimensión, puede ser sobre un valor único, una lista o un rango de valores dentro del nivel o dimensión. Por ejemplo un filtro sobre la dimensión tiempo podría ser: *"Mostrar sólo la recaudación del 2005"*.

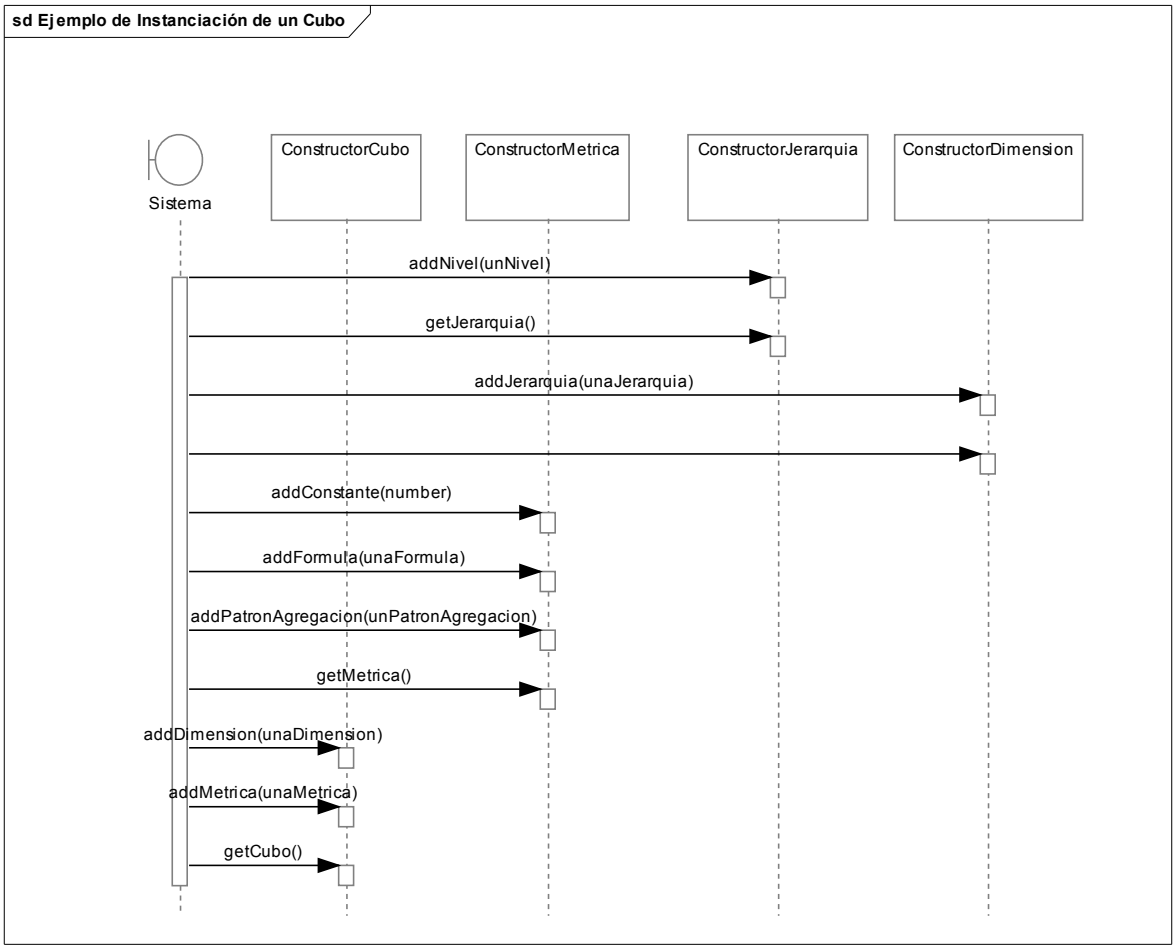


Figura 28: Diagrama de interacción - Ejemplo de instanciación de un cubo

Cada vez que se ejecuta una consulta se utiliza y se van generando vistas específicas que van a ser explotadas por la herramienta para mostrar los resultados de la consulta y permitirán ejecutar nuevas consultas. Cada vista utiliza instancias de la clase *PasoNavegación* que guarda estados anteriores, el diseño del mismo respeta el patrón de diseño Memento [Gam95] para ir guardando el estado del cubo luego de la aplicación de cada consulta, este objeto tiene la inteligencia como para devolver la vista anterior en caso de que se necesite deshacer la última operación y/o volver al cubo original.

En la **Figura 30** podemos ver las diferentes operaciones sobre el cubo que aquí se diseñan explícitamente: *rollup, drilldown, slice&dice, collapse, expand, jump, back*[2].

2

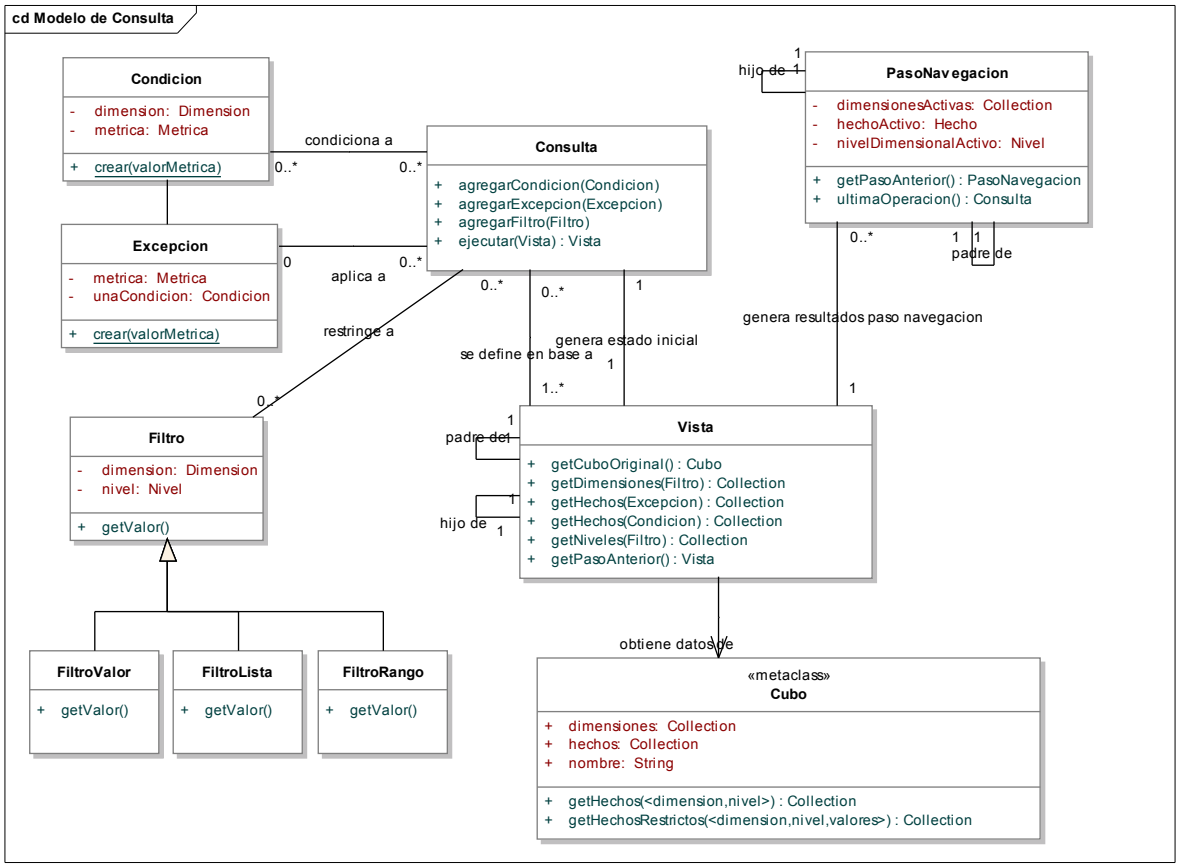


Figura 29: Metamodelo de Consulta

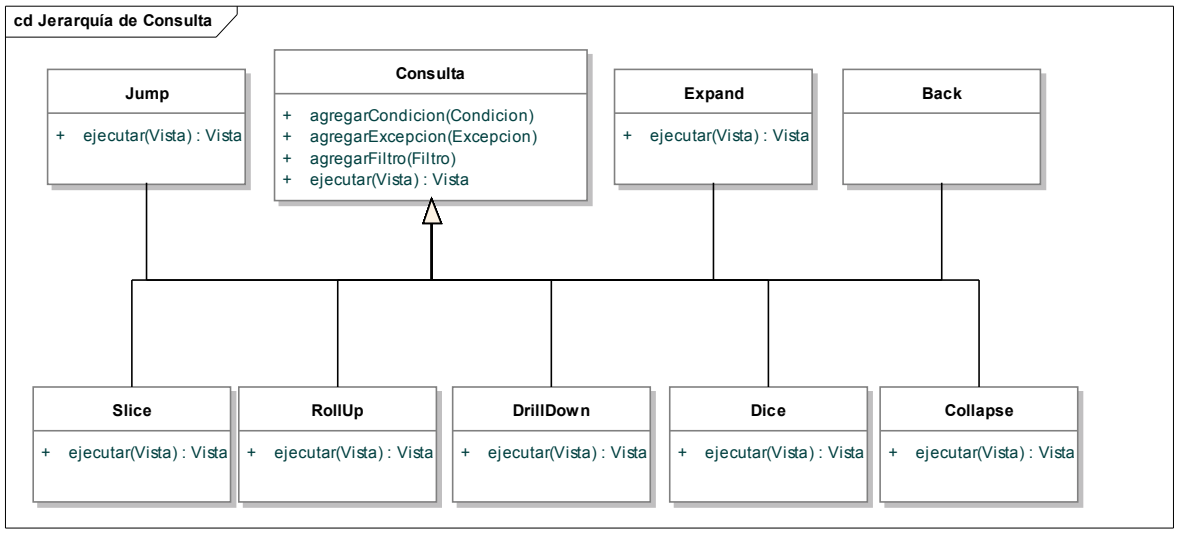


Figura 30: Jerarquía de Consulta

SECCION III – Aplicación de AO a Seguridad en OLAP

En este capítulo se presentan una investigación sobre modelado de seguridad en DW, y luego una introducción a la teoría de orientación a aspectos y a su utilización en el ciclo de desarrollo de software haciendo hincapié en la etapa de diseño para seleccionar posteriormente una serie de artefactos de diseño apropiados para extender el metamodelo multidimensional OO, presentado en la sección anterior, con aspectos para el modelado de seguridad y perfiles.

Modelos de seguridad en datawarehousing

La evolución de los datawarehouses ha sido paralela e independiente de la creciente sensibilidad y evolución de los conceptos de seguridad y privacidad de la información.

Motivo de esta independencia es el hecho que en los comienzos, el datawarehouse estaba pensado para ser consultado sólo por altos niveles jerárquicos de las organizaciones. Como resultado, pocas eran las restricciones de seguridad requeridas por estas soluciones y los productos comerciales se aprovecharon de esta situación, brindando aplicaciones OLAP con poco o ningún nivel de seguridad.

Sin embargo, a medida que las soluciones de análisis multidimensional (OLAP) y datawarehouses se convirtieron en herramientas indispensables para la gestión de la información en las organizaciones, la comunidad de usuarios ha ido en considerable aumento, incluyendo a personas ajenas a la organización (como ser distribuidores y socios entre otros) y la necesidad de control de acceso a los datos, caracterizados por una alta sensibilidad, se transformó en un requerimiento indispensable.

Uno de los grandes obstáculos que presenta la implementación de seguridad en datawarehousing radica en su misma naturaleza ya que es un repositorio integrado de información, mayormente de mucho valor y sensibilidad, para fines de consulta de la empresa. Entonces, cómo protegemos la sensibilidad de la información y controlamos el acceso a la misma sin restringir al mismo tiempo la libertad de análisis y navegación (Ej.: La realización de consultas ad-hoc) para lo cual fueron pensados los datawarehouses y herramientas OLAP.

En los últimos años los investigadores y los desarrolladores de productos comerciales de datawarehousing y seguridad se han dedicado a unir estos dos campos de estudio.

Por un lado, se han desarrollado herramientas comerciales OLAP con implementaciones de seguridad propietarias y sintaxis no aptas para propósitos de diseño conceptual y documentación. Como se puede observar, en este campo también es poco lo desarrollado en términos de modelado conceptual.

Por otro lado, se ha intentado sin éxito, mapear las políticas de seguridad existentes en los ambientes transaccionales a ambientes multidimensionales. Algunas razones de este fracaso radica en que la seguridad en entornos transaccionales se aplica básicamente a tablas mientras que en entornos multidimensionales se debe aplicar a dimensiones, jerarquías, ratios y demás objetos característicos del entorno multidimensional.

Control de acceso

Como mencionamos anteriormente, los DWs por su naturaleza crean un conflicto de seguridad porque, por una parte, la meta es hacer que todos los datos necesarios estén accesibles de la manera más fácil posible, por otra parte estos datos son usualmente muy valiosos y sensitivos. Focalizaremos el concepto de seguridad en autorización y control de acceso.

Se podría dividir el control de acceso en dos partes:

- El acceso a las fuentes de datos y la carga en el datawarehouse.

- El acceso para consultas.

La primera parte se concentra en el control de acceso del proceso de extracción, transformación y carga del datawarehouse (ETL) donde se destacan dos roles importantes, el del desarrollador, quien debe tener acceso básicamente a la metadata, no a los datos en sí mismos, y el rol de operación, que son los responsables de ejecutar los procesos de extracción propiamente dichos, donde requiere el derecho de ejecución y acceso a los datos. En ambos casos es importante notar que los derechos de acceso son sólo de lectura, ya que no se modifican los datos fuente. Sólo en caso de que surjan problemas en la carga, o en algunos procesos particulares de "*cleansing"* (purificado de datos), generarán la necesidad de otorgar mayores privilegios a estos dos roles.

La segunda parte, y más controvertida, se refiere al acceso de los usuarios al datawarehouse con fines de análisis. Principalmente, se requiere el establecimiento de políticas de acceso para los usuarios finales. La complejidad de éstas políticas varía de acuerdo al tipo de análisis a realizar y el nivel y el rol del usuario en la organización. Las políticas más sencillas son las que se encuentran asociadas a reportes predefinidos que no permiten mayores actividades de análisis. En estos casos, simples listas de acceso por usuario, o grupos de usuarios es suficiente. Las políticas asociadas a análisis multidimensionales y ad-hoc, dadas sus características exploratorias de información, son generalmente muy complejas.

Según [Pri00] los requerimientos de seguridad asociados a OLAP pueden clasificarse según su complejidad en los siguientes niveles:

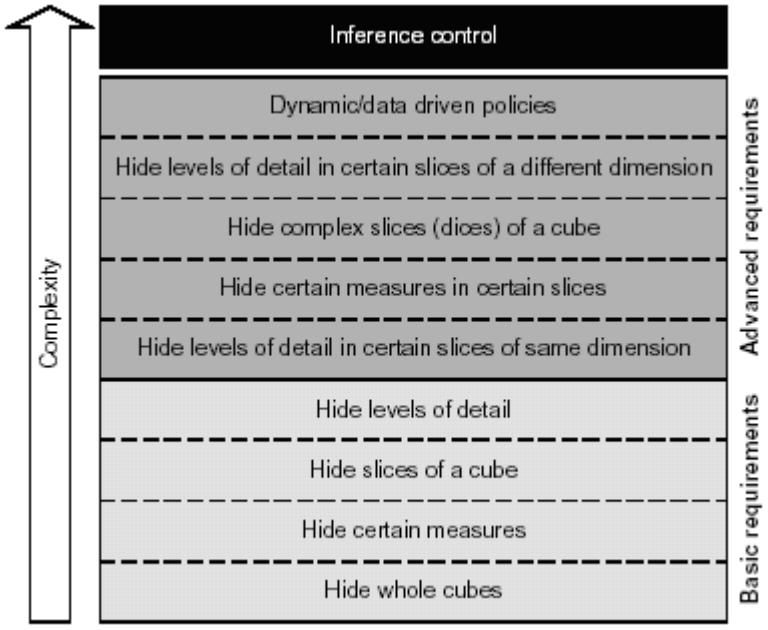


Figura 31: Diferentes requerimientos de controles de acceso para OLAP ([Pri00]).

Los denominados *"Basic Requirements"* (requerimientos básicos) son generalmente los más sencillos de implementar y hoy en día son provistos por la mayoría de las aplicaciones OLAP. En cambio, los *"Advanced Requirements"* (requerimientos avanzados), generalmente tratan la aplicación de restricciones de seguridad a caras del

cubo, o subconjuntos de los espacios de análisis de los mismos, y no siempre pueden ser implementadas ya sea debido a su complejidad o debido a la posibilidad de generar resultados erróneos, imperceptibles al analista, como consecuencia de las restricciones sobre algunos de los datos.

Adicionalmente, los responsables del establecimiento e implementación de políticas de acceso en un datawarehouse tienen que lidiar con problemas de inferencia de información. Esto implica que existen casos donde, a pesar de encontrarse protegida la información a nivel de detalle, existe alguna clasificación paralela sobre otra dimensión sin restricciones que puede develar datos que de forma directa no se encontrarían disponibles o al solicitarlos agregados se convierten en accesibles. Se protege el detalle pero no los niveles agregados. Esto muchas veces no es detectado por los mismos responsables de la seguridad del datawarehouse.

Una posible solución es prohibir consultas que involucren menos de un cierto número de registros (políticas como *"query-set size control"* o *"smallest indicador"*) [Wan03]. Se demostró que de todas formas combinaciones de consultas agregadas pueden ser usadas para inferir datos de detalle ocultos y estas inferencias de consulta múltiple son mucho más difíciles de tratar. En conclusión se necesita ahondar más en investigación sobre este tema pero aun así es muy probable que no se resuelvan la mayoría de los casos en la práctica.

Políticas de seguridad

La política más implementada por los mecanismos de seguridad de los datawarehouses, debido a su naturaleza de repositorio de información para fines de gestión, se denomina *"Mundo Abierto"* que significa que *"todo se encuentran permitido excepto que sea expresamente prohibido".*

Es importante poder tratar por separado políticas y mecanismos de seguridad. Las políticas expresan el qué de lo que se está restringiendo y los mecanismos especifican el cómo, la forma de implementación de dichas restricciones.

Ello permitirá que al definir reglas de control de acceso se pueda razonar acerca de ellas independientemente de su implementación, poder compararlas entre sí e inclusive comparar distintos mecanismos de implementación para una determinada política.

Mecanismos de seguridad

Proveen los medios para la definición de los sujetos de seguridad (*Usuarios, Grupos, Roles*) que pueden o no, acceder a ciertos objetos del datawarehouse (cubos, dimensiones, hechos o una combinación de estos entre otros).

Entre las propuestas para implementar mecanismos de seguridad en los datawarehouses observamos principalmente dos tendencias:

- Vistas: Este mecanismo define un subconjunto declarativo del sistema que puede ser accedido por la herramienta de consulta, limitando como consecuencia las capacidades de navegación (también se oculta parte de los metadatos o de miembros de las dimensiones). Diferentes usuarios, grupos o roles verán, de acuerdo a sus privilegios, cubos derivados que provienen tal vez del mismo cubo

origen. Esto es totalmente transparente al usuario final, porque no se le es negada ninguna consulta o navegación, ya que no sabe de su existencia.

- Reglas: Los usuarios conocen la existencia de todos los objetos del universo del datawarehouse, sólo que no tienen permiso de acceso a alguno de ellos de acuerdo a reglas predefinidas. Estas pueden ser muy complejas y difíciles de mantener. Algunos reportes pueden ser prohibidos o incluir celdas en blanco. Para informar al usuario acerca de resultados falsificados las celdas que fueron dejadas en blanco por razones de seguridad son usualmente marcadas explícitamente.

Ambas propuestas tienen sus ventajas y desventajas; en cuanto a las vistas su transparencia puede ser peligrosa porque los datos perdidos pueden falsificar metas y resultados de análisis. Si se implementan las políticas con reglas lo que suele ocurrir es que los usuarios aunque tengan el acceso restringido a ciertos objetos, al poder ver datos sumarizados de todo el universo pueden inferir valores de datos restringidos.

En la práctica las implementaciones más frecuentes son:

Las que se basan en arquitecturas relacionales: aplican el control de acceso a través de la división del datawarehouse en varios datamarts o de la utilización de vistas SQL y filtros trasladando de esta manera las restricciones de acceso a la arquitectura relacional. Estas están en general basadas en vistas.

Las que se basan en aplicaciones OLAP o herramienta de consulta multidimensional aplicando el control de acceso sobre los componentes de visualización a ciertos usuarios/grupos/roles a través de reglas.

En conclusión, para la implementación de un mecanismo de seguridad robusto, la combinación de ambos sería la más recomendada. Sin las vistas, la definición e implementación de reglas puede llegar a ser muy complejo, y sin las reglas, restricciones del tipo *"el usuario U puede acceder a la dimensión A y ver en detalle solamente el miembro A1, el resto de los miembros sólo podrán ser visualizados a su 3er nivel de sumarización"* son casi imposibles de implementar.

Ejemplo de un modelo conceptual de seguridad multidimensional

Dos de los autores que más se han dedicado al estudio de seguridad sobre bases de datos multidimensionales, Priebe y Pernul, presentan en [PrPe01] una aproximación al modelado conceptual de la seguridad sobre datawarehouses.

El modelo que presentan surge como la necesidad de cubrir la falta de investigación sobre el tema, la creciente necesidad de proteger información sensible en un entorno naturalmente abierto pensado para la gestión de datos de todos los niveles de la corporación y la fácil y correcta documentación de las políticas de seguridad del datawarehouse.

El modelo se basa en:

- Un *"Esquema de Políticas de Seguridad"* centralizadas y estáticas.

- Las restricciones de seguridad se aplican sobre objetos (*Cubos, Dimensiones, Jerarquías,* etc) y *Sujetos* (o *Roles*).
- Los roles son no-jerárquicos y no se superponen.
- Se utilizan un esquema de restricción de seguridad negativo: En general, las restricciones de seguridad pueden ser positivas - *"Grants"*- o Negativas – *"Denials"* - pero como un datawarehouse se basa en la política de *"Mundo Abierto"* (*"Open World Policy")* se siguió con esta filosofía - todo se encuentra Permitido excepto que sea explícitamente negado.
- Las restricciones, desde ya, son solamente de lectura.

El metamodelo conceptual multidimensional sobre el cual introdujeron las extensiones de seguridad está expresado en un lenguaje propietario de los autores -*ADAPTed UML*- que consiste en una extensión de UML que utiliza símbolos *ADAPT* –una notación propietaria de ORACLE Express para describir modelos multidimensionales - como estereotipos de UML. Este metamodelo básico introduce los elementos *Cubo, Medida, Nivel de Dimensión* y *Atributo de Dimensión* (propiedad calificativa de la dimensión). La granularidad del cubo está dada por el nivel base de dimensión.

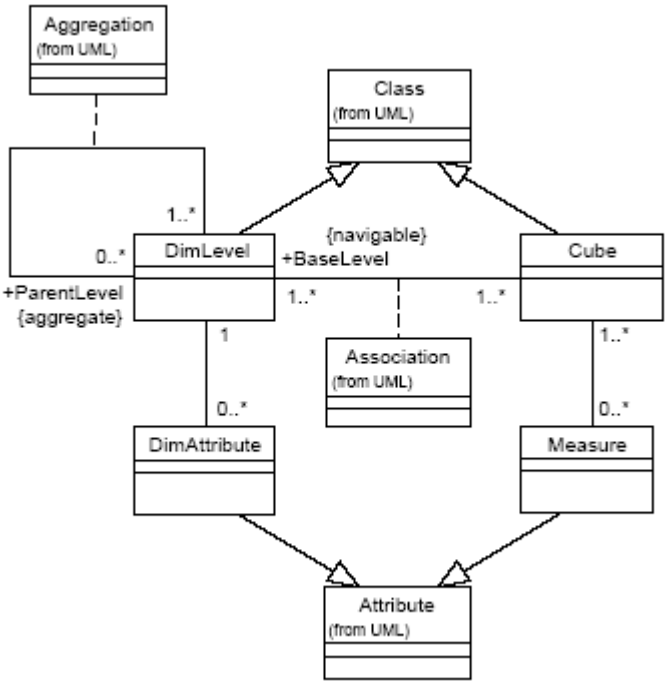


Figura 32: Metamodelo multidimensional propuesto por [PrPe01]

Este metamodelo es utilizado para la instanciación del escenario sobre el cual los autores aplican el esquema de seguridad. Este escenario se basa en el seguimiento y gestión de tours de turismo a castillos y monumentos en Bohemia y la República Checa. Se ha seleccionado este escenario ya que la administración de estos castillos se encuentra organizada jerárquicamente. Todos excepto dos de los castillos son administrados por un instituto regional aunque también conservan algo de autonomía. Por este motivo, están dispuestos a almacenar sus datos en un mismo repositorio

integrado sólo en el caso que sean aplicados mecanismos de seguridad para los diferentes roles de usuarios.

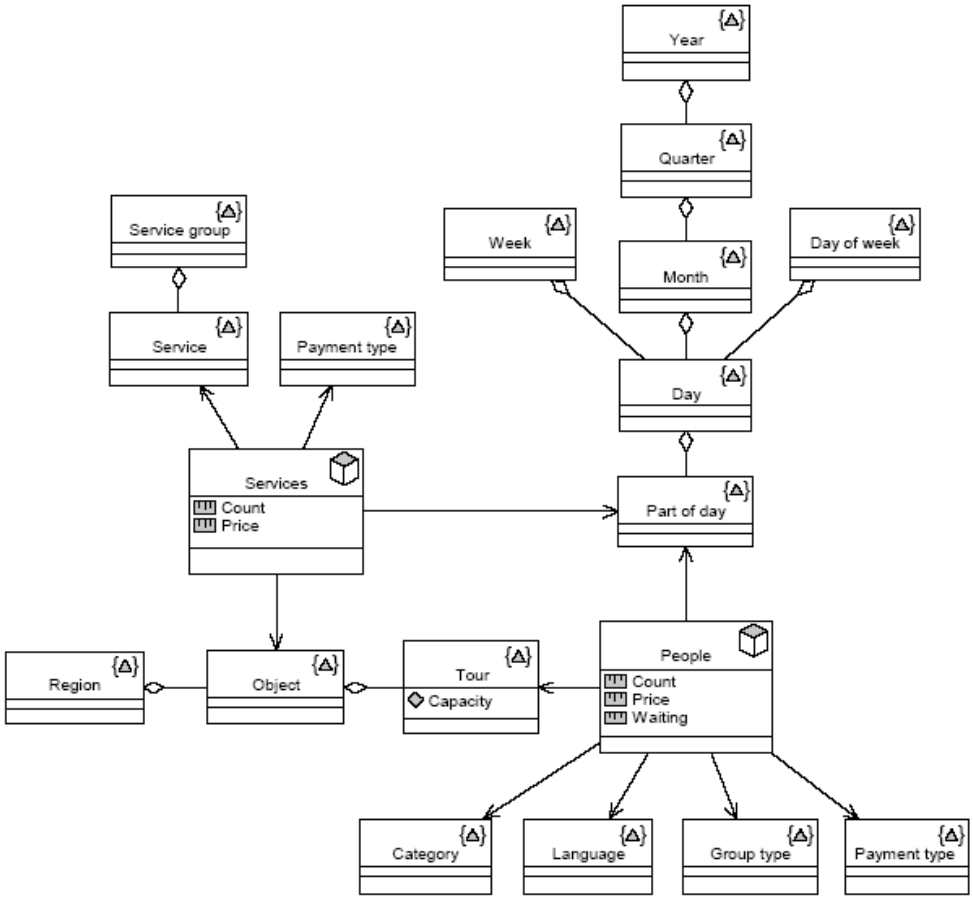


Figura 33: Modelo conceptual del escenario elegido a partir del metamodelo de [PrPe01]

Como se puede observar, hay dos cubos: el de *Personas* que contiene datos sobre la admisión de los visitantes, y el de *Servicios* que contiene toda la información asociada a los consumos realizados por estos visitantes. Tanto la dimensión *Tiempo* como *Geografía* son compartidas por ambos cubos.

Para complementar este modelo conceptual de seguridad, los autores también introducen un *"Lenguaje para Restricción de Seguridad Multidimensional"* (Multidimensional Security Constraint Language o **MDSCL**) que se basa en MDX.

Ejemplo de las sentencias MDSCL de Seguridad que presentan estos autores:

- **<hide cube statement>** ::= HIDE CUBE <cube name> FOR ROLE <role name>

    - Oculta la existencia de un cubo a un Rol determinado.
    - Ej: Ocultar el cubo de *Personas* al *Rol "Proveedor de Servicio"* para que no se quede con datos de los visitantes.

- **<hide measure statement>** ::= HIDE MEASURE <measure name> [WHERE <slicing constraint>] FOR ROLE <role name>.

    - Oculta una medida o ratio a un determinado rol cuando se cumple la condición de *Slice* dada en el *Where*.
    - Ej: Ocultar el ratio *Precio* para el *Rol "Proveedor de Servicio"*.

- **<hide slice_statement>** ::= HIDE SLICE WHERE <slicing constraint> { AND <slicing constraint> } FOR ROLE <role name>.

    - Oculta un slice de un cubo.
    - Ej: Ocultar todos los objetos diferentes de Hulboka al *Rol "Hulboka Manager"* para evitar que el administrador de Hulboka pueda analizar los otros castillos o monumentos que no administra.

- **<hide level statement>** ::= HIDE LEVEL <string> [WHERE <slicing constraint>] FOR ROLE <role name>.

    - Oculta un *Nivel* completo de *Dimensión*, muy usado en general cuando se quiere ocultar un nivel completo de detalle, y que sólo puedan ser visualizados niveles agregados.
    - Ej: Mostrar nivel de dtalle de *Tour* sólo si se trata del ojeto = Hulboka y el *Rol* es *"Hulboka Manager"*. Es en este caso, que puede suceder que las jerarquías queden con niveles de detalles (Ej: *Tour*) para algunos y no para otros. Esto se conoce como *"Frayed Dimensions"*.

Donde la representación gráfica se realiza a través de notas UML con las restricciones de seguridad definidas como se puede apreciar en la **Figura 34**.

Extensiones posibles propuestas por los autores:

- Manejo de Políticas de Seguridad Dinámicas: que se basen no sólo en los objetos multidimensionales definidos sino en el valor real que toma una métrica, o la agregación de la misma en un determinado momento.
- Controlar la Inferencia de datos: Como se mencionó antes, limitar en la medida de lo posible, la inferencia de datos a partir de otros.
- Extender al manejo de Sujetos Jerárquicos.
- Introducción de Restricciones Positivas, como por ejemplo *Show* en combinación con las negativas, como por ejemplo *Hide*. Ampliaría la posibilidad de definir restricciones complejas pero complicaría la resolución en combinación con la jerarquía de roles.

Este trabajo es uno de los primeros que centran su atención en la definición y modelado a nivel conceptual de los requerimientos de seguridad que tienen los datawarehouses. Esta notación gráfica basada en notas UML es complicada de mantener y documentar cuando las políticas de seguridad son voluminosas, existen cantidad de roles y son complejas. Por otra parte el metamodelo que proponen estos autores es complicado de utilizar en la práctica dado que ADAPT no es un lenguaje de modelado estándar como UML y sería complicado integrar estos artefactos en una herramienta estándar.

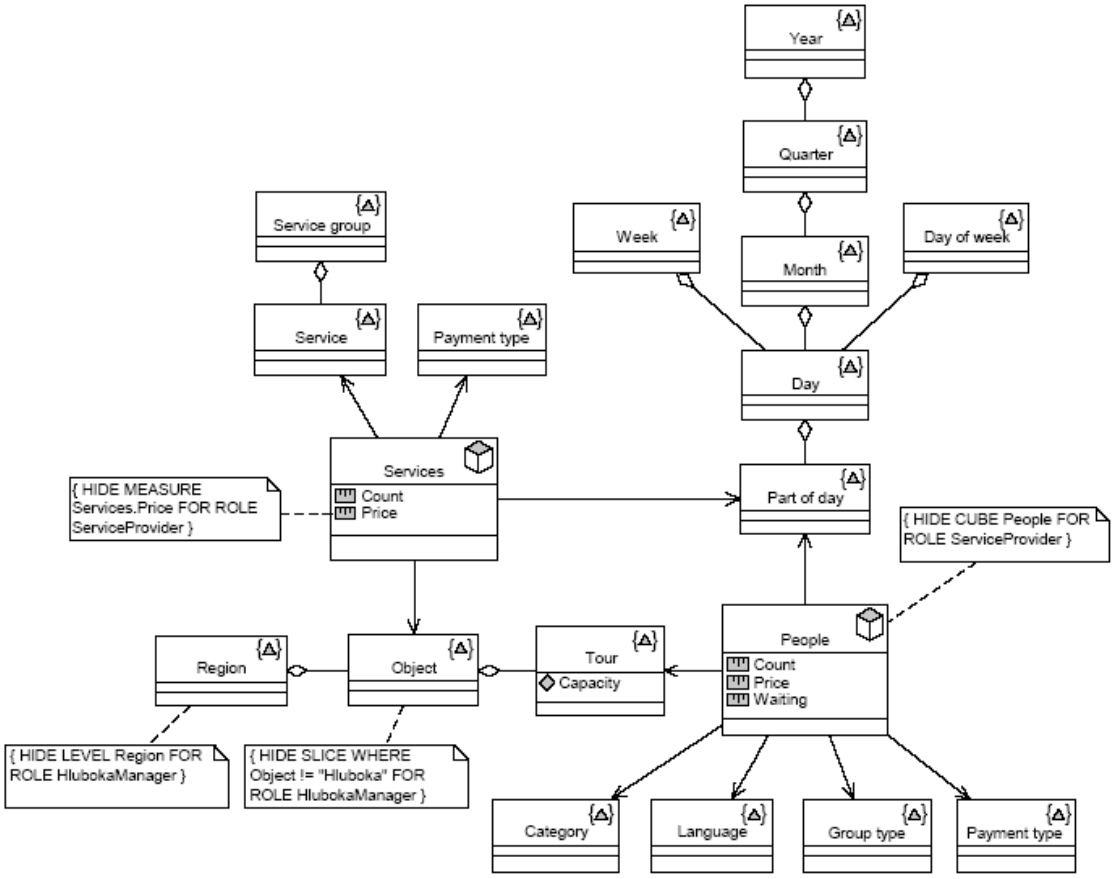


Figura 34: Modelo conceptual de seguridad para el escenario elegido [PrPe01]

Trabajos recientes

Durante nuestra investigación surgieron algunos trabajos posteriores que presentan modelos conceptuales de seguridad en DW, uno de los últimos publicado a principios de 2005 es el trabajo presentado por Villaroel, Medina y Piattini dirigidos por Trujillo [Vil05]: es un metamodelo conceptual para el modelado de seguridad en ambientes multidimensionales, basado en una extensión al metamodelo conceptual presentado por Luján Mora en [Mor05].

El modelo que presentan es muy completo y fácilmente adaptable para su aplicación a una herramienta comercial ya que ofrece la posibilidad de clasificar la información de seguridad en varios niveles, dependiendo del usuario y a qué departamento pertenece. Permite también modelar un conjunto de roles de usuario y restricciones de seguridad utilizando UML 2.0 que es un lenguaje ampliamente conocido y utilizado en herramientas CASE, por lo tanto otra de las bondades de este modelo es que puede ser utilizado y testeado por la comunidad de usuarios que utilizan UML.

En cuanto a su utilización en el modelado de DW vemos como desventaja que para definir cada elemento – o clase - del modelo (*Hecho, Dimensión, Atributo de Hecho*, etc) se debe agregar su información de seguridad utilizando atributos como: una

secuencia de niveles de seguridad, un conjunto de departamentos de usuario, un conjunto de roles de usuario y restricciones de seguridad utilizando OCL.

Teniendo en cuenta que estos atributos deben ser agregados para cada clase del modelo en cada uno de los modelos que se definan resulta ser un tanto incómodo para la persona encargada de definir el modelo además de violar el encapsulamiento.

Esto ocurre porque los autores están pretendiendo modelar un concern transversal, como es la seguridad, mezclándolo y enredándolo con el resto del modelo con lo cual no se diferencia claramente de los otros concerns – *cubos, dimensiones, hechos*, etc. – lo cual se contradice con el modelado conceptual que consiste en identificar conceptos y modelarlos con entidades o clases conceptuales separadas.

Es por esto que en los capítulos subsiguientes nos ocupados de presentar las características de un nuevo paradigma para la separación de concerns – paradigma de aspectos - y explicar su aplicación a nuestro metamodelo multidimensional que comparte características de flexibilidad y fácil aplicación en entornos reales con el modelo de Trujillo y Piattini pero además porque provee una opción no invasiva al modelado de seguridad en DW.

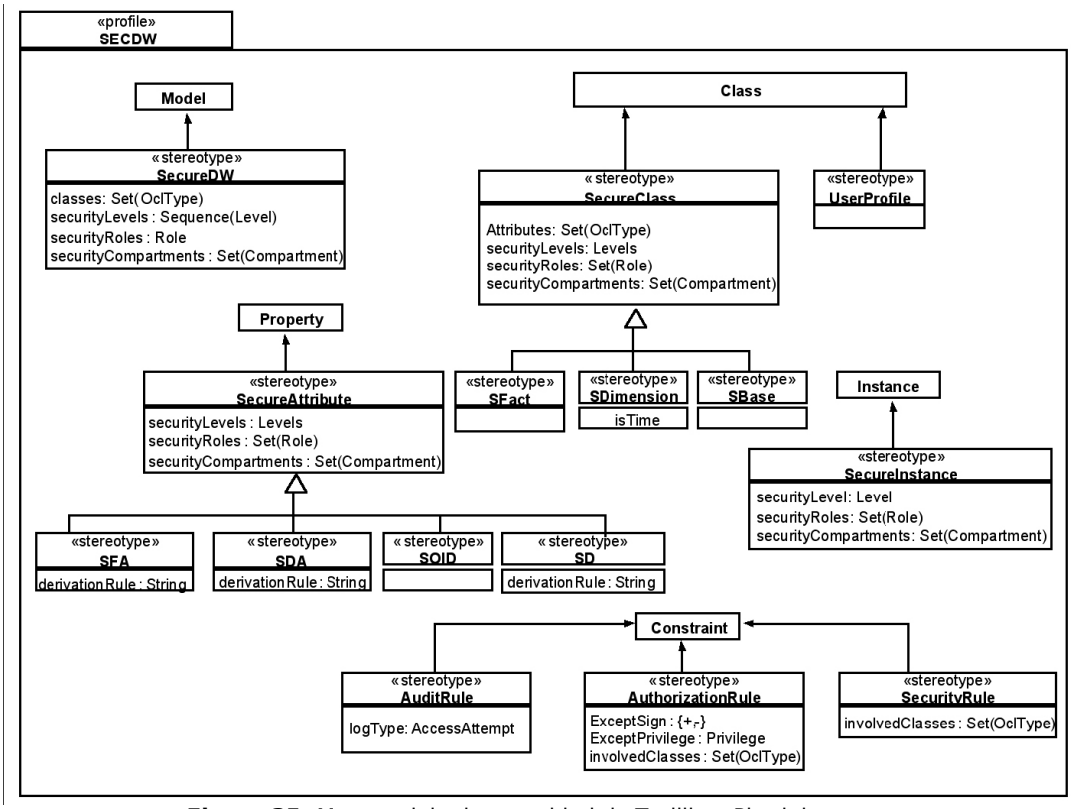


Figura 35: Metamodelo de seguridad de Trujillo y Piattini

Orientación a aspectos – Una introducción

La orientación a aspectos es una nueva tecnología para la separación de funcionalidades de interés, de ahora en adelante *concerns*, en el área de desarrollo de software.

Las técnicas de diseño orientado a aspectos (AOSD) hacen posible modularizar aspectos que cruzan transversalmente un sistema. Como los objetos, los aspectos pueden surgir en cualquier etapa del ciclo de vida del software, incluyendo la especificación de requerimientos. Ejemplos comunes de aspectos transversales son el diseño de restricciones de arquitectura, propiedades o comportamiento sistemático (ej: logging y recuperación de errores).

Los investigadores en AOSD son mayormente guiados por la meta fundamental de la separación de concerns. Así, mientras el cruce transversal tiende a ser un foco significativo en su trabajo, ellos también incorporan otras técnicas, incluyendo orientación a objetos y programación estructurada. Esto se refleja en la manera en que gran parte del trabajo en el campo soporta diferentes tipos de modularidad incluyendo estructuras de bloque, estructuras de objetos, herencia y también cruce transversal.

Motivación

Las ciencias de la computación han experimentado una evolución en lenguajes de programación y sistemas desde el crudo lenguaje ensamblador y código máquina de las primeras computadoras a través de conceptos como la traducción de fórmulas, programación procedural, estructurada, funcional y lógica y programación con tipos abstractos de datos. Cada uno de estos pasos en las tecnologías de programación ha avanzado nuestra habilidad para alcanzar la clara separación de concerns a nivel de código fuente.

Actualmente, el paradigma de programación dominante es la orientación a objetos (**POO**), basado en la idea de que se puede construir un sistema de software descomponiendo un problema en objetos y luego escribiendo el código para esos objetos. Tales objetos abstraen juntos comportamiento y datos dentro de una única entidad conceptual y física. La orientación a objetos es reflejada en el espectro completo de las metodologías y herramientas actuales de desarrollo de software; ha hecho posible escribir aplicaciones complejas como interfaces gráficas (**GUI**s), sistemas operativos y aplicaciones distribuidas y mantener al mismo tiempo código fuente comprensible. Pero el éxito en el desarrollo de sistemas simples lleva a aspirar mayor complejidad e implica mayor necesidad de separación de concerns.

La orientación a objetos aventaja a las otras técnicas de la ingeniería de software en que es una idea inteligente para modelar comportamiento ya que el modelo de objetos subyacente está más cerca del dominio real del problema. Pero en cuanto a la separación de concerns tiene ciertas limitaciones ya que muchos requerimientos no se descomponen con nitidez en comportamiento centrado en un único lugar. La tecnología de objetos así como las técnicas de programación procedural tienen dificultad localizando concerns que involucran restricciones globales y comportamiento pandémico. De este modo hay importantes decisiones de diseño que por cruzar transversalmente la funcionalidad básica del sistema no quedan capturadas

correctamente por estas técnicas. Esta deficiencia en el diseño se traduce en código entrelazado y excesivamente difícil de desarrollar y mantener. Esos concerns transversales son llamados aspectos.

Los problemas de diseño descriptos se conocen generalmente como *"Code Scattering"*: El código correspondiente a un concern, por ejemplo el logging, no es encapsulado en un único módulo, sino que aparece disperso por el programa, y *"Code Tangling"*: dentro de un mismo módulo encontramos código correspondiente a distintos concerns. Estas dos propiedades originan código más complejo y difícil de mantener, con todas las desventajas que esto acarrea.

A raíz de estos problemas de diseño que surgen con la programación orientada a objetos se plantean otras técnicas de diseño y programación que se engloban en los mecanismos Post Object Oriented Programming (**POP**) que buscan aumentar la expresividad del paradigma de orientación a objetos. Ejemplos de tecnología POP son *"Generative Programming"* [CZA00], *"Generic Programming"* [Mus03], *"Constraint Languages"* [OMG04], *"Reflección y Metaprogramación"* [Tan02], *"Feature-Oriented Development"* [Pre97], *"Views/Viewpoints"* [Har93], *"Asynchronic Message Brokering"* y *"Aspect-Oriented Programming"* [Kic97].

La programación orientada a aspectos surge entonces como una nueva evolución en la línea de las tecnologías para separación de concerns, tecnologías que permiten que el diseño y el código sean estructurados para reflejar la manera en que los desarrolladores quieren ver la estructura del sistema. AOP se construye sobre tecnologías existentes y provee mecanismos adicionales que hacen posible afectar la implementación de sistemas cruzando transversalmente el código.

En AOP un único aspecto puede contribuir a la implementación de un número de procedimientos, módulos u objetos. La contribución puede ser homogénea, por ejemplo proveyendo una interfase de logging que puedan utilizar todos los procedimientos, o puede ser heterogénea, por ejemplo implementando dos lados de un protocolo entre dos clases diferentes. Como con el resto de las tecnologías de separación de concerns, la meta de AOP es hacer diseños y código más modular es decir que los concerns estén localizados en vez de estar desparramados por el código; y que tengan interfaces bien definidas lo cual implica modularidad, poder pensar en forma aislada en cada uno de los concerns.

Una clara comparación entre AOP y OOP podría resumirse en el siguiente párrafo:
"Mientras que la tendencia en OOP es encontrar cosas comunes entre clases y llevarlas hacia arriba en el árbol de herencia, AOP intenta concentrar los concerns dispersos como elementos de primera clase, y eyectarlos horizontalmente de la estructura de objetos".

Conceptos fundamentales

A continuación presentamos definiciones genéricas de los conceptos base de AOP

Crosscutting concerns (funcionalidad transversal)

Dos concerns se entrecruzan si los métodos relacionados a los mismos tienen intersección. Los artefactos utilizados para describir, diseñar e implementar un concern dado son llamados métodos. Decimos que un método se relaciona con un concern si

contribuye a la descripción, diseño o implementación del concern. Veamos un ejemplo con un editor de figuras:

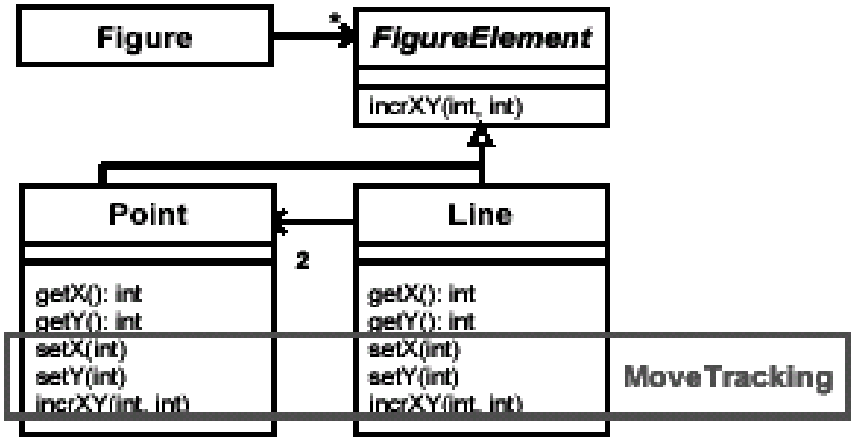


Figura 36: Descripción UML de un simple editor de figuras [Kic01].

Aquí hay dos clases concretas que heredan de *FigureElement*. Estas clases manifiestan buena modularidad en que el código fuente en cada clase tiene mucha cohesión y cada clase tiene una interface clara y bien definida. Considérese ahora el concern relacionado con la notificación del movimiento de un *FigureElement* al manejador de pantalla. Esto requiere que cada método que mueve un *FigureElement* haga la notificación.

La caja que cruza el diseño envuelve cada método que debe implementar este concern tales como las cajas *Point* y *Line* están dibujadas alrededor de cada método que implementa dichos conceptos gráficos. La caja *MoveTracking* no queda dentro ni alrededor de las otras cajas en la figura, en vez de eso esta las cruza transversalmente. Esto es lo que llamamos un concern transversal. Usando sólo programación orientada a objetos la implementación de concerns transversales tiende a dispersarse a través del sistema. Usando los mecanismos de AOP podemos modularizar la implementación del comportamiento de *MoveTracking* en un único aspecto. Como podemos implementar este comportamiento en una única unidad modular, es más fácil para nosotros pensarla como una única unidad de diseño. Teniendo los mecanismos de aspectos en el lenguaje de programación nos permite pensar en términos de aspectos a nivel de diseño también.

El corte transversal es relativo a una descomposición en particular. Los concerns transversales de un diseño pueden no estar claramente separados entre si. Una regla básica de diseño es representar concerns significativos como abstracciones de primera clase en el lenguaje. Pero a veces es difícil diferenciar cuál concern es el más importante, en el ejemplo ocurre eso: el movimiento es tan inherente a las figuras como lo es su posición por lo tanto podríamos haber modelado el movimiento como concern principal y la posición como concern transversal. Es por eso que lo más difícil es determinar cuáles son los concerns transversales y cuales las abstracciones principales. Teniendo en cuenta que vamos a considerar que existe una descomposición dominante.

Los concerns pueden ir desde nociones de alto nivel como ser seguridad y calidad de servicio o nociones de bajo nivel como *"caching"* y *"buffering"*. Pueden ser funcionales, como características o reglas de negocio o no funcionales (sistémicos) como sincronización y manejo de transacciones.

Join Points (puntos de unión)

En AOP las clases se diseñan y se codifican separadamente de los aspectos que encapsulan el código que cruza transversalmente. Los links entre aspectos y clases se expresan por medio de los *"join points"* (puntos de unión). Esto se puede hacer de forma estática como una fase en tiempo de compilación o dinámicamente – puntos en la ejecución del programa - dependiendo del lenguaje que se utilice para implementar los aspectos. En el caso en que se expresen dinámicamente los join points son puntos en la ejecución de un programa en los que pueden entremezclarse aspectos y código base. Algunos ejemplos son las llamadas a métodos, el acceso a atributos de un objeto o el lanzamiento de una excepción.

Se van a utilizar cinco tipos de elementos relacionados con los join points para modularizar concerns transversales cuya implementación variará ampliamente dependiendo del lenguaje de aspectos que se utilice , ellos son:

- Modelo de join points que describe el "enganche" en donde se va a agregar el código del concern.
- Medio para identificar join points.
- Medio para especificar comportamiento en los join points.
- Unidades encapsuladas que combinen especificación de join points y mejora de comportamiento.
- Método para adjuntar unidades a un programa.

Point Cuts (puntos de corte)

Facilita la selección y agrupación de join points y valores en esos puntos que permitirán luego especificar dónde corresponde aplicar un determinado aspecto.

La gran mayoría de las herramientas existentes en la actualidad permite definir dichos conjuntos bien por enumeración o mediante predicados sobre la sintaxis del código. Los *join points* son entonces caracterizados en función de elementos sintácticos. Deben suponerse *convenciones de notación,* que en definitiva no hacen más que dejar implícitos elementos semánticos comunes a todos los integrantes del conjunto.

Advice

Se denomina *"advice"* a la entidad que indica qué aspecto corresponde aplicar en todos los joinpoints de un pointcut. Hay diferentes tipos de advice: *"around", "before", "after"* y, en algunos lenguajes, *"throw"*, que indican respectivamente dónde corresponde aplicar el aspecto: alrededor, antes, después o en caso de excepción en un joinpoint.

Introduction

El agregado de métodos o campos a una clase de un advice. Por ejemplo se puede introducir una nueva interface a un objeto advice: hacer que cualquier objeto implemente la interface *Modificable* para simplificar el caching.

Weaving (entretejido entre aspectos y código)

Más allá de los diferentes aspectos que afecten a un programa, el sistema que queremos obtener es uno solo: hay que definir una relación entre los aspectos y el código base. Será necesario establecer algún tipo de "pegamento" entre ellos. Para ello existe el *"aspect weaver"* (tejedor de aspectos) que es la herramienta que se utiliza para entretejer el código de nuevo, aspectos con clases, ya sea en forma estática, tiempo de compilación, o dinámica, tiempo de ejecución. Un aspect weaver debe procesar el lenguaje de componentes y el lenguaje de aspectos, componiéndolos para generar el sistema completo. El concepto de join point es esencial porque son esos elementos de la semántica del lenguaje de componentes con los cuales coordina el programa hecho en el lenguaje de aspectos. Un ejemplo de join point: invocación de métodos en tiempo de ejecución en el programa de componentes, no son necesariamente construcciones explícitas pero si elementos de la semántica del programa de componentes. El aspect weaver trabaja generando una representación de los join points del programa de componentes y luego ejecutando (o compilando) el programa de aspectos respecto de ese.

La representación de join points puede ser generada en tiempo de ejecución usando métodos reflectivos en el lenguaje de componentes. Aquí el lenguaje de aspectos es implementado como un metaprograma, que se llama en cada invocación de método, que usa la información de los join points y el programa de aspectos, para saber como hacer el *"marshalling"* apropiado de los argumentos.

Aspecto

Y por fin la definición: *Qué es un aspecto?*

Como mencionáramos anteriormente un aspecto es una abstracción usada por la AOP para localizar concerns transversales tales como código que no puede ser encapsulado dentro de una clase pero está mezclado en varias clases.

Pero dependiendo de la etapa de desarrollo de la que se trate y la herramienta que los implementen se pueden presentar diferentes definiciones:

A nivel diseño: representa una abstracción que sirve para modelar concerns transversales.

A nivel de implementación: una construcción de programa que permite que los concerns transversales puedan ser capturados en unidades modulares. Compuesta por todos los elementos anteriores, advices, pointcuts y los elementos básicos de un lenguaje de programación orientado a objetos: declaraciones de campos, métodos y constructores.

AOP aplicado al ciclo de vida del software

La separación de concerns es un principio de central importancia en la ingeniería de software que debe ser aplicado a través del ciclo de desarrollo desde los requerimientos hasta la implementación. Sin embargo, hasta la actualidad la mayor parte del trabajo en esta área se ha concentrado en el nivel de implementación. Mientras que el foco está cambiando a etapas de desarrollo más tempranas tal como

diseño, mucho menos trabajo existe acerca de separación de concerns durante la etapa de ingeniería de requerimientos.

Por ello nuestro objetivo es hacer a continuación un estudio detallado de las técnicas de modelado de aspectos existentes hasta el momento en las diferentes etapas del desarrollo del software haciendo hincapié en el diseño conceptual y las herramientas que permiten definirlo para poder luego aplicarlo a nuestro metamodelo conceptual multidimensional.

Arquitectura

Según [Nav2] la arquitectura de software permite al diseñador especificar la estructura del sistema en términos de componentes y conectores. Los componentes especifican la funcionalidad del sistema mientras que los conectores determinan la interacción entre los componentes.

Dos de las herramientas más útiles para los arquitectos de software son los estilos arquitecturales y los *"Lenguajes de Descripción de Arquitectura"* (**ADL**s). Los estilos de arquitectura nos dan reglas para construir familias de sistemas con características similares como por ejemplo sistemas *"batch"*, *"pipes and filters"*, *"call and return"*, *"virtual machines"* y *"repositories"*. ADL son lenguajes que proveen primitivas para especificar componentes y conectores ejemplos son: *"Rapide", "Darwin", "Wrigh"* y *"Acme"*.

Examinando la separación de concerns transversales desde un punto de vista estructural, para introducir conceptos de orientación a aspectos en el diseño de arquitectura sería necesario adaptar las herramientas de arquitectura de software actuales lo cual no es una tarea trivial. Con respecto a estilos de arquitectura se podría proponer un estilo en el cual los aspectos sean manejados por separado, esto no es fácil por las siguientes razones:

- La diferente naturaleza de los aspectos que intervienen en un sistema hace difícil manejarlos de una manera simple y uniforme usando un único estilo.

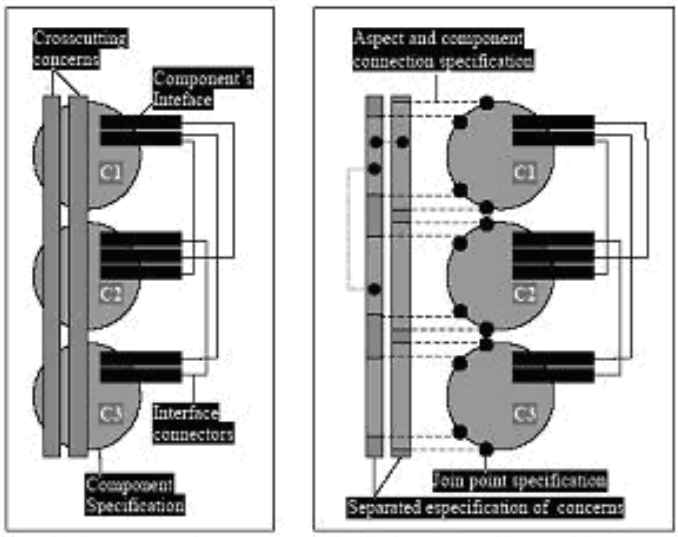
- Los mismos aspectos en diferentes sistemas pueden requerir diferente tratamiento. Por ejemplo una aplicación con restricciones de tiempo real y distribución puede requerir tratar la restricción de tiempo real primero y otra al revés: primero la de distribución.

Para manejar la separación de concerns a un nivel de arquitectura el ADL actual tiene que ser extendido para proveer la siguiente funcionalidad:

- Para especificar componentes funcionales: poder especificar join points en dichos componentes. Se deben proveer nuevas primitivas que soporten la especificación de todo tipo de join points.

- Para especificar aspectos: que tienen una morfología diferente a la de los componentes funcionales en el sentido de que no proveen servicios ni interfaces. Deberían ser especificados como un tipo especial de componentes descriptos por nuevas primitivas.

- Para especificar conectores entre aspectos y join points. Este problema estaba ya resuelto por modelos de coordinación entonces se debe estudiar la posibilidad de importar soluciones de esta área. En este caso los conectores pueden ser los componentes de coordinación.



Figuras 37a y 37b: Especificación de arquitecturas de software tradicional e incluyendo aspectos [Nav02]

Supongamos que las **Figuras 37a** y **37b** especifican la arquitectura de software del mismo sistema. La **Figura 37a** muestra cómo se especifican actualmente la arquitectura de software. La especificación de los concerns que intervienen en el sistema está cruzando transversalmente la especificación de los componentes.

En la **Figura 37b** la especificación de los aspectos ha sido separada. Para producir un diseño equivalente al primero los arquitectos de software deben:

- Marcar los join points en la especificación de los componentes.

- Especificar las conexiones entre aspectos y join points.

Estas conexiones tienen una naturaleza diferente que la conexión entre interfaces y su objetivo es mantener la coherencia del sistema original preservando su semántica. Tales conectores tienen que especificar dónde y cómo cada aspecto va a ser tratado. Según los autores este es un problema típico de coordinación que fue resuelto con modelos de coordinación y lenguajes.

Otros como [Kat03] proponen modelar aspectos de manera genérica planteando que no existe soporte en un lenguaje de diseño como para diseñar aspectos en forma incremental y que la relación entre aspectos es difícil de modelar cuando éstos no son ortogonales. Vemos entonces que los aspectos solapados introducen un nuevo tipo de problema porque no se puede dejar claro a qué módulo pertenece cada sección del lenguaje.

La solución propuesta en este caso es plantear que un aspecto puede ser visto como un mapeo que aumenta un artefacto de diseño preexistente a uno nuevo con más

detalle. Lo nuevo que introduce el aspecto puede cortar transversalmente las divisiones existentes en módulos.

En este enfoque se considera a los concerns como cuestiones conceptuales que pueden ser tratadas por uno o más aspectos. Propone modelar los aspectos de manera genérica para que puedan ser reutilizados instanciándolos varias veces.

Definición

- Usa: describe los join points a los cuales el aspecto va a ser aplicado para aplicar nuevo comportamiento y para restringir el contexto en el cual ese aspecto es aplicable.

- Join Points: pueden ser vistos como elementos paramétricos de artefactos de diseño.

- Define: que introduce los elementos adicionales.

- Composición de aspectos: se basa en el principio de superimposición que es una operación asimétrica en la cual un aspecto es aplicado sobre otro. A la superimposición del aspecto B sobre el A se la denota B/A.

Cada concern es relacionado en forma modular por una colección de aspectos ordenados por la relación *depend*.

El modelo de combinación de aspectos es abstracto así que para usarlo tiene que ser instanciado en algún lenguaje de diseño.

Para instanciarlo se introduce un nuevo tipo de diagrama llamado diagrama de concerns.

Un aspecto de diseño es un estereotipo de un paquete por lo tanto pueden contener todos los tipos de diagrama. La dependencia entre aspectos, denotada por una flecha, define un orden parcial de cómo van a ser aplicados.

Veamos un ejemplo de modelado de un aspecto de manejo de memoria en un sistema controlador de audio en la **Figura 38**.

Los concerns se dibujan como líneas irregulares circulares que encierran la correspondiente colección de aspectos. En la **Figura 39** podemos apreciar un ejemplo de modelado de todo el sistema controlador de audio.

Trabajos recientes

A la fecha debemos destacar que han surgido nuevas propuestas en modelos de componentes orientados a aspectos en donde los aspectos no se diferencian de los componentes a nivel semántico como es el caso de *"AspectLagoona"* [Gal03], *"FuseJ"* [Suv03] y *"CoCompose"* [Wag03] – este último propone un modelo de arquitectura mientras que los anteriores son lenguajes de programación -. Son modelos de arquitectura en donde la funcionalidad transversal se delega al nivel de conectores de componentes – nivel de comunicación – como el modelo de componentes que propone FuseJ. Las desventajas de estos modelos es que la funcionalidad transversal es

expresada a muy alto nivel – nivel de componente – diferenciándose de otros lenguajes de aspectos donde la granularidad de la funcionalidad transversal es mucho menor: nivel de clase o método.

En cuanto a AspectLagoona, la ventaja es que la funcionalidad transversal: advice, se especifica a nivel de método -mensaje-.

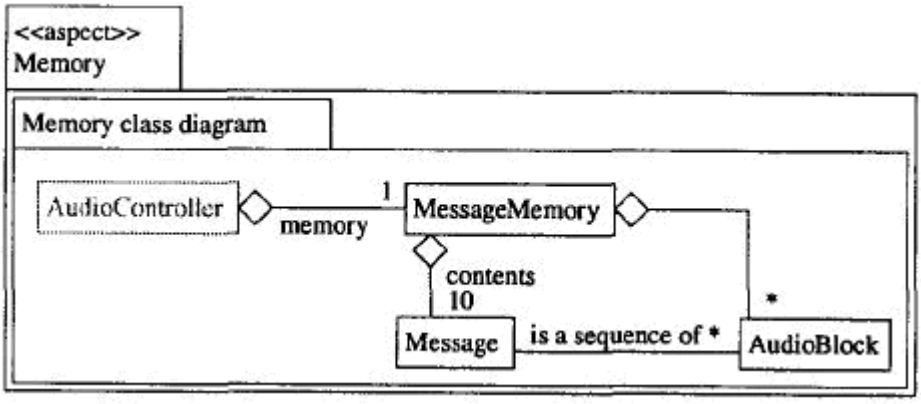


Figura 38: Aspecto de memoria

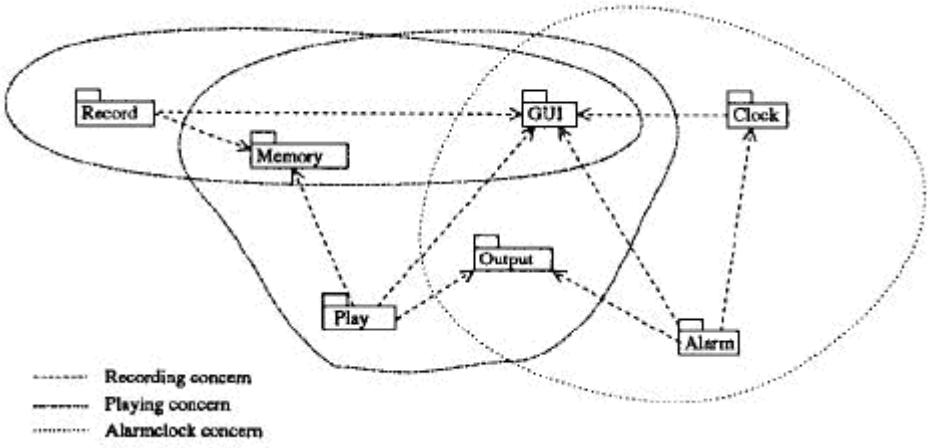


Figura 39: Diagrama de concerns – características principales

Análisis

Aunque la aplicación de aspectos en el análisis está todavía en etapa de investigación se han propuesto varias soluciones para comenzar con esta práctica ya que la aplicación temprana de aspectos haría posible establecer rápidamente la relación costo-beneficio entre requerimientos funcionales y aspectuales proveyendo por lo tanto soporte para negociación y subsiguiente toma de decisiones entre los accionistas, además esto facilitaría la determinación del mapeo e influencia de requerimientos

transversales en artefactos en etapas posteriores del desarrollo lo cual ahorraría significativamente los costos.

Los concerns que se manejan en la etapa de requerimientos comprenden todas aquellas propiedades no funcionales que requerimos del sistema y que restringen los requerimientos funcionales como por ejemplo: tiempo de respuesta y seguridad. En las etapas de diseño e implementación pueden surgir otros concerns transversales a partir de las limitaciones impuestas por la tecnología elegida como por ejemplo: manejo de excepciones y sincronización.

En cuanto a modelos propuestos podemos destacar las que se presentan en [Ras03] en el que se plantea una realización basada en *"Viewpoints"* y XML y en [Ras02] en donde se presenta la misma técnica utilizando casos de uso.

A continuación en la **Figura 40**, una vista simplista del proceso de ingeniería de requerimientos orientado a aspectos que estos modelos plantean.

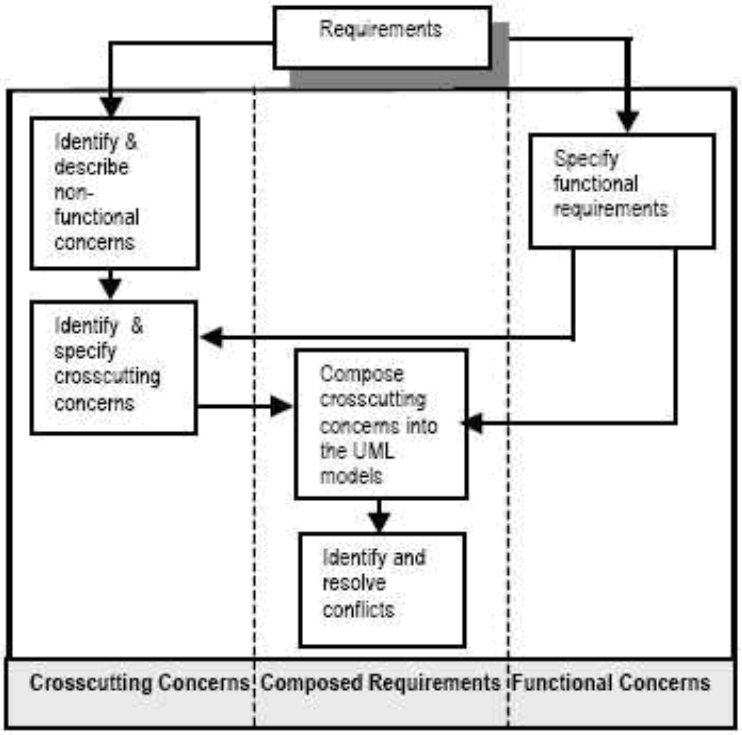


Figura 40: Un modelo de requerimientos orientado a aspectos

El proceso es partido verticalmente en tres partes principales:

- Concerns transversales: identificación de requerimientos no funcionales a partir de los cuales se discriminan los transversales que van a ser candidatos a aspectos.

- Concerns funcionales: Especificación tradicional de requerimientos funcionales utilizando diferentes técnicas como Viewpoints o modelado de casos de uso con UML.

- Requerimientos compuestos:

- Se relacionan concerns transversales con requerimientos por medio de una matriz de manera que se evidencie qué concerns cruzan transversalmente los módulos que encapsulan requerimientos de los usuarios y se los califica como candidatos a aspectos.

- Se definen reglas de composición detalladas. Estas reglas operan a la granularidad de requerimientos individuales lo cual facilita la identificación de requerimientos aspectuales individuales y conflictivos.

- Se identifican y resuelven conflictos entre aspectos candidatos. Para ello:

    - Se construye una matriz de composición donde cada aspecto puede contribuir negativamente o positivamente a los otros.

    - Se atribuyen pesos a esos aspectos que contribuyen negativamente a los otros en relación a un conjunto de requerimientos. Cada peso es un número real en el intervalo [0...1] y representa prioridad de un aspecto en relación a un conjunto de requerimientos.

- Se resuelven los conflictos con los stakeholders utilizando la priorización anterior. La resolución de conflictos puede llevar a una revisión de la especificación. Si esto pasa los requerimientos son recompuestos y se van resolviendo conflictos posteriores. El ciclo se repite hasta que todos los conflictos han sido resueltos.

- Luego se adoptan los conceptos de sobreposición, sobreescritura y envoltorio para definir la parte compuesta del modelo:

    - Sobreposición: Los requerimientos del aspecto modifican los requerimientos funcionales que atraviesan. En este caso los requerimientos aspectuales pueden ser requeridos antes de los funcionales, o después.

    - Sobreescritura: Los requerimientos del aspecto se superponen a los requerimientos funcionales que atraviesan. En este caso el comportamiento descripto por el requerimiento aspectual sustituye el comportamiento de los requerimientos funcionales.

    - Envoltorio: Los requerimientos del aspecto "encapsulan" los requerimientos funcionales que atraviesan. El comportamiento descripto por el requerimiento funcional es "envuelto" por el comportamiento descripto por el requerimiento aspectual.

- La última actividad es la identificación de las dimensiones de un aspecto:

    - Mapeo: Un aspecto puede mapearse en una función/característica del sistema por eso es que se los llama candidatos a aspecto porque pueden mapearse o no en un aspecto en etapas posteriores.

    - Influencia: un aspecto puede influenciar diferentes puntos en el ciclo de desarrollo (análisis, diseño, implementación).

A continuación veamos un ejemplo de cómo sería modularizar un concern transversal utilizando Viewpoints con xml, el siguiente concern especifica que un sistema de cobro de peaje debe ser compatible con sistemas usados para activar y reactivar.

```
<?xml version="1.0" ?>
- <Concern name="Compatibility">
- <Requirement id="1">The system must be compatible with systems used to:
        <Requirement id="1.1">activate and reactivate dispositivo;</Requirement>
        <Requirement id="1.2">deal with infraction incidents;</Requirement>
        <Requirement id="1.3">charge for usage.</Requirement>
- </Requirement>
</Concern>
```

Diseño

Es importante tener una buena navegabilidad entre el modelo de análisis y el de diseño y entre éste y la implementación en un lenguaje de programación. En particular, en lo que al diseño se refiere, un proceso de diseño y un lenguaje de programación cumplen con esta propiedad cuando este último provee abstracciones y mecanismos de composición que soportan de manera clara las unidades que surgen del proceso de diseño que dividen al sistema. Y viceversa, al aplicar aspectos en un lenguaje de programación, es necesario poder mapearlos con unidades del modelo de diseño.

Los lenguajes de diseño orientado a objetos no son suficientes para modelar el encapsulamiento de artefactos que cruzan transversalmente la arquitectura por lo tanto es necesario introducir nuevos elementos para permitir el modelado de aspectos.

Entre las nuevas técnicas de modelado de aspectos que existen hoy en día podemos mencionar "Patrones de Composición", "Estereotipos sobre UML" y "AML".

La propuesta de *"Patrones de Composición"* está basada en la combinación del paradigma de diseño orientado a sujeto [Clar01] y *"templates"* (plantillas) que extienden el lenguaje de modelado UML. Se sustenta en la idea de que existen patrones en la manera en que los concerns transversales se comportan con respecto al diseño base. Por lo tanto se pueden diseñar sujetos reusables y concerns en forma independiente. Luego se usan reglas de composición para combinarlos.

Tiene la ventaja de poder mapearse a un modelo de programación que provee una solución para separar requerimientos transversales en el código. Este mapeo sirve para ilustrar que la separación de aspectos puede ser mantenida a través del ciclo de vida del software.

Permite descomponer el diseño en diferentes modelos llamados "sujetos de diseño" que representan vistas independientes y cada uno se mapea con un requerimiento. Un patrón de composición es un sujeto de diseño con potencialmente múltiples patrones de clases. En UML es representado como un template utilizando el artefacto *"paquete"* con estereotipo *"subject"*. Contiene templates de clases con sus métodos que son también templates que van a ser reemplazados luego por elementos reales.

En el siguiente ejemplo se muestra un patrón de composición conformado por una clase *Trace* y una clase template *TracedClass* que puede ser reemplazada por cualquier clase que necesite la funcionalidad de *"tracing"* (rastreo) en alguno/s de su/s métodos. Y cualquier operación de esa clase que requiera comportamiento de tracing reemplazará al método *tracedOp(..)*.

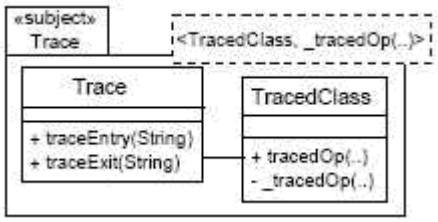


Figura 41: Especificación de templates en un patrón de composición Trace [Clar01].

El modelo de diseño orientado a sujeto define una relación de composición para especificar cómo sujetos diferentes pueden ser integrados a una salida compuesta. UML define una relación de *"binding"* (vinculación) entre plantillas de especificación y los elementos que van a reemplazar dichas plantillas.

Figura 42: Especificación de vinculación para composición [Clar01]

Patrones de composición no provee una notación explícita para especificar advices por lo cual éstos se expresan en diagramas de estado. Un diseñador debe proveer un diagrama de estados adicional para cada punto de ejecución. En este caso la notación se complica y se hace difícil de leer para sistemas complejos.

En la **Figura 43** se puede apreciar un ejemplo de diagrama de interacción.

Veamos ahora cómo los conceptos de patrones de composición se mapean con los conceptos que hemos definido previamente para la teoría de aspectos:

- Aspecto: Un sujeto de patrón de composición es el equivalente en términos de diseño a un aspecto.

- Punto de corte (pointcut): Los parámetros template de una operación pueden ser definidos y referenciados dentro de especificaciones de interacción, denotando que son puntos de junta (joinpoints) para comportamiento transversal. Estos templates pueden ser reemplazados por las operaciones reales múltiples veces, y, por lo tanto son equivalentes a los puntos de corte.

- Advice: Dentro de un diagrama de interacción el comportamiento transversal puede ser especificado para ejecutar cuando se llama a una operación template. Este comportamiento es equivalente al código del advice.

- Introduction: Los elementos de diseño que no son elementos template pueden ser definidos dentro de patrones de composición. Estos pueden ser clases, atributos, operaciones o relaciones y son equivalentes a una introducción.

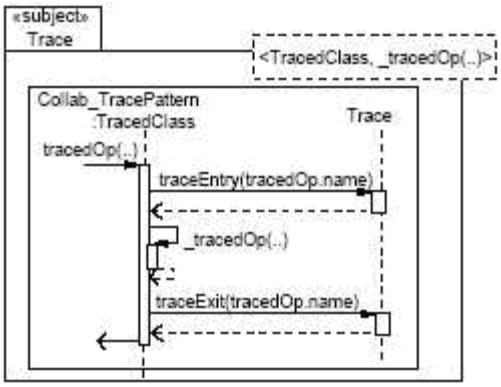


Figura 43: Especificación de patrones de comportamiento transversal [Clar01].

En cuanto a la propuesta de utilizar **estereotipos con UML** para modelar aspectos [Wai02]: es un framework que provee una herramienta para entretejer modelos UML de alto nivel en modelos de diseño capaces de ser implementados, simulados o validados. Está basado en agregar estereotipos, valores etiquetados y patrones de diseño a un modelo UML para denotar los aspectos que afectan al modelo.

El diseñador puede agregar gran cantidad de información no funcional a un modelo agregando anotaciones a los elementos del modelo. Los mecanismos de anotación más usados son:

- Estereotipos

Para subtipar un tipo de elemento del modelo. Por ej: si quiero especificar que las instancias de la clase *History* deberían ser persistentes. Luego herramientas automáticas pueden identificar este elemento y procesarlo específicamente.

- Patrones de diseño

Para especificar que un patrón de diseño dado debe ser aplicado en un lugar específico en el modelo. Por ejemplo podríamos querer indicar que ciertos métodos de una clase deben participar en el *"Command Pattern"* (Patrón de diseño **Command**) como en el caso de *ServiceProvider*. En base a esta notación se puede construir un tejedor.

- Valores etiquetados

Son pares clave-valor proveen información adicional para guiar el proceso de entretejido. Pueden ser puestos en una clase para especificar la elección entre varias implementaciones existentes del patrón *Command*.

Un ejemplo de estas anotaciones se puede ver en la **Figura 44**. En este diagrama de clases se modelan las entidades del dominio así como también las anotaciones para generar implementaciones eficientes en la plataforma en la que se instancia el modelo.

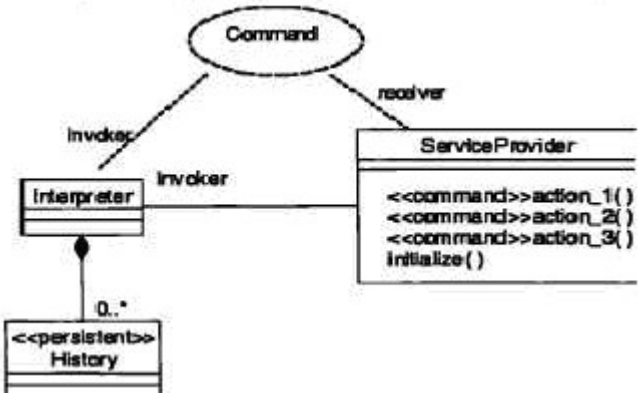


Figura 44: Diagrama de clases de UML con estereotipos [Wai02].

La decisión final de implementación consiste entonces en decirle al tejedor qué grupo de aspectos componer y cómo deben ser compuestos de acuerdo a esta información no funcional agregada en forma de anotaciones.

Por último hemos considerado *"Aspect Modeling Language"* (**AML**) [Gro04] basado en UML estándar. Esta propuesta es una de las primeras expresiones de los avances descriptos en cuanto a lenguajes de arquitectura de componentes para describir aspectos pero a nivel de diseño detallado, en donde el comportamiento aspectual se delega a un artefacto especial que representa el conector entre componentes. Esta nueva técnica ayuda a que los desarrolladores se familiaricen rápidamente con el modelado orientado a aspectos cuando ya están familiarizados con el modelado orientado a objetos en UML.

Expresa los diseños a través de artefactos transversales encapsulados y mantenidos aparte de la lógica de negocios lo cual permite soporte de varios lenguajes de aspectos. Ambos son reusables y al mismo tiempo independientes de la tecnología de implementación. Respeta los requerimientos para desarrollar modelos de aspectos en aplicaciones del mundo real que consisten en una notación simple soportada por herramientas CASE conocidas como para mejorar la productividad de los desarrolladores y para asegurar la precisión sintáctica del modelo de aspectos. Provee modelos fáciles de leer y además un mapeo directo entre la notación y los lenguajes de implementación soportados que deberían permitir generación automática de código basada en el modelo de diseño.

Como los concerns transversales tienden a afectar múltiples clases en un sistema y un concern puede consistir de varias clases y como todas estas clases pueden estar asociadas con la clase que el concern cruza, la construcción modular para un concern debe ser de mayor nivel que una clase, por lo tanto para modelar concerns se utiliza el artefacto paquete de UML. En la **Figura 45** se puede apreciar un ejemplo de cómo quedaría un modelo orientado a aspectos en AML.

El conector encapsula la tecnología de implementación subyacente (por ejemplo aspectos). El aspecto puede ser modelado independientemente de cualquier diseño que afecte. El paquete de aspecto provee una representación gráfica (diagrama de clases) de la vista estática de un concern transversal particular. El paquete base contiene la lógica de negocio del sistema y puede ser modelado independientemente de los concerns. La relación entre ambos paquetes está dada por el conector, el cual puede contener las siguientes clases:

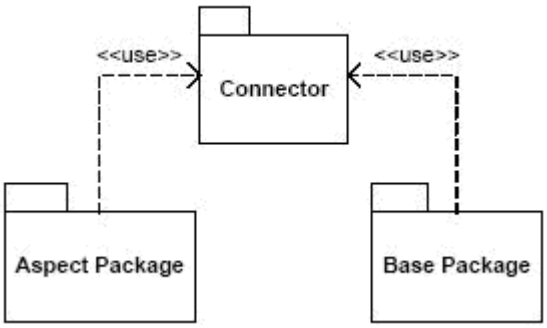


Figura 45: Descomposición a nivel de paquete [Gro04].

- Introduction que define las reglas para mecanismos de introduction de aspecto.
- Pointcut que define puntos de ejecución en el flujo de control del programa.
- Advice que define el código a ser ejecutado en la clase *Pointcut*.

Un ejemplo con aplicación de *tracing* se puede apreciar en la **Figura 46**.

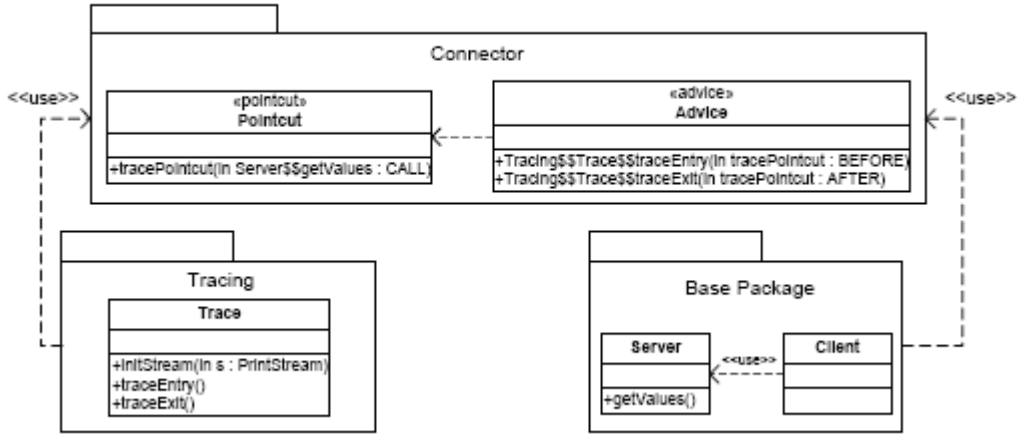


Figura 46: Ejemplo de diseño del aspecto Tracing con AML [Gro04].

Tracing (rastreo): cada vez que el usuario invoca a un método en el servidor la acción debe ser rastreada. El pointcut se ejecuta cada vez que el cliente invoca el método *Server.getValues()*.

Para generar código se puede utilizar "*Together*" (http://www.borland.com/resources/en/pdf/products/together/together_datasheet.pdf): una herramienta que valida y genera automáticamente las partes OO del modelo. La validación y la generación de código de las partes AO (los elementos del conector) es implementada como módulos que se "enchufan" en la plataforma Together.

A continuación listamos una serie de criterios de buen diseño que nos ayudarán a comparar las propuestas anteriores para modelar aspectos:

- Trazabilidad

Consiste en la facilidad de navegar de los requerimientos transversales al diseño, del diseño a partes del modelo de diseño y finalmente al código. Este criterio es para asegurar que la propuesta de diseño provee construcciones aptas para separación de requerimientos relevantes. Para asegurar esta propiedad es menester que las notaciones de diseño de arquitectura soporten modelado de acuerdo a las técnicas mas comunes del paradigma AOP. Sería deseable también poder contar con un mapeo directo entre la notación de diseño a pequeña escala y los lenguajes de implementación soportados para permitir generación automática de código basada en el modelo de diseño.

- Propagación de cambio

Facilidad de adaptación del diseño completo cuando un cambio es introducido a una parte del modelo. Si un concern transversal está bien separado, debería ser fácilmente modificable sin afectar otros concerns y debería no ser afectado por modificaciones de otros concerns.

- Reusabilidad

Facilidad de reuso de diseño para otros sistemas. Los mismos requerimientos transversales o similares surgen a menudo en diferentes sistemas. Los diseños producidos para satisfacer estos requerimientos para uno de estos sistemas debe estar listo para reusar en otros sistemas. Esto implica que los concerns transversales de naturaleza general capturados en las construcciones de diseño deben ser independientes de contexto, mientras que la información de contexto misma debería ser encapsulada en unidades separadas.

- Comprensibilidad y facilidad de uso

Facilidad de entender el sistema de software a través de los diseños. Los diseños de concerns transversales deben ser auto contenidos y entendibles sin referenciar otros concerns. Una notación conocida y fácil de usar por los desarrolladores como UML ayudaría a este propósito. Por eso el diseño debe ser soportado por herramientas CASE poderosas para mejorar la productividad de los desarrolladores y para asegurar correctitud sintáctica del modelo AO.

- Flexibilidad

Habilidad de la técnica de diseño para adaptarse a diferentes técnicas de separación de concerns, lenguajes de implementación, herramientas, etc. Una técnica de diseño flexible podría acercar diferentes propuestas para separación de concerns transversales, así como permitir reuso de artefactos de diseño a través de diferentes propuestas, lenguajes y herramientas.

- Desarrollo paralelo

De partes del diseño del sistema. Cuando los concerns de base y transversales pueden ser modelados separadamente, diferentes tipos de concerns pueden ser diseñados simultáneamente por gente diferente. Este criterio contribuye a aumentar la productividad de los equipos de diseño.

Comparación de las propuestas

Todas extienden la misma notación de diseño base en UML. En la siguiente tabla se puede apreciar el mapeo entre artefactos de las diferentes propuestas:

Tabla 2: Comparación de diferentes propuestas para diseño de aspectos			
Concepto aspectual	**Patrones de composición**	**AML**	**Estereotipos con UML**
Puntos de corte y advices	Declaración de variables de instancia, métodos, etc.	Pointcut	Valores etiquetados
Introducción	Clases, aspectos, interfaces, paquetes	Paquete base y paquete aspectual	Clases
Aspecto	Patrones de composición y sujetos	Paquete aspectual y paquete conector	Etiquetas de estereotipo, patrones de diseño y valores etiquetados
Aspectos compuestos o entretejidos con el modelo	Sujetos, patrones de composición con sus reglas de composición.	Aspectos con reglas para relacionar –tejer- componentes con aspectos a través de los conectores.	Clases etiquetadas. Se ven efectivamente en la implementación luego de que el tejedor generó el código.

Y a continuación presentamos una comparación en cuanto a las características de "buen diseño" enumeradas previamente:

- Trazabilidad

Patrones de composición tiene buena trazabilidad de requerimientos a diseño. Si cada unidad de diseño se implementa en forma independiente antes de la composición los diseños van a ser fácilmente trazables a módulos en el código pero si son compuestos en etapa de diseño van a re-introducir enredamiento en el código.

AML no necesita mezclar los concerns transversales con los otros componentes, estos apenas tienen referencias entre ellos por eso el diseño es fácilmente trazable a lo largo de todo el ciclo de desarrollo. Permite así a los desarrolladores razonar sobre la validez de configuración de los componentes.

En "Estereotipos con UML" la separación de concerns transversales en estructura de diseño de aspectos asiste en trazar requerimientos, aquí los aspectos tienen referencias directas a los elementos base por lo tanto no tiene reglas claras de composición a nivel diseño, y la composición es fija en cada aspecto, en cada caso, por lo tanto la trazabilidad es impedida por referencia a objetos base.

- Propagación de cambio

Patrones de composición tiene habilidad para introducir cambios en cualquier sujeto sin invadir otros pero igual las relaciones de composición deben ser revisadas y actualizadas. Cuando el diseño tiene varios niveles de composición necesita ser revisto por completo para que composiciones inválidas no sean pasadas a otros módulos.

En AML los aspectos de diseño están separados modularmente de los servicios que afectan, sólo unidos por los conectores permitiendo a los diseñadores analizar posibles propagaciones de cambios usando estos links de dependencias.

En el modelo de *"Estereotipos con UML"* los cambios a un aspecto no siempre se pueden hacer en forma independiente porque los elementos de los objetos están fijados en las operaciones con aspectos.

- Reusabilidad

Patrones de composición producen diseños reusables que pueden ser compuestos con otros diseños base.

AML provee un mecanismo de interacción desacoplado aumentando la reusabilidad de los componentes debido a la información acerca de los servicios requeridos/provistos. Mientras que el diseño de componentes puede ser reusado a través de diferentes tecnologías de implementación, algunos diseños podrían requerir adaptación.

El modelo de *"Estereotipos con UML"* permite reuso de aspectos y objetos de diseño, aunque las firmas de las operaciones de aspectos deben ser siempre actualizadas. Provee la habilidad de intercambiar descripción de aspectos entre herramientas UXF/a compatibles a través del ciclo de desarrollo de software.

- Comprensibilidad

Patrones de composición produce diseños claramente segregados para concerns base y transversales. Pero cuando los patrones de composición y el diseño base son compuestos se reintroduce el enredamiento. Consecuentemente el diseño integrado se vuelve complicado de entender. Además no provee notación explícita para especificar advices por eso éstos se expresan a través de diagramas de estados. Un diseñador está forzado a proveer un diagrama de estados adicional para cada punto de ejecución. Mientras uno está modelando la notación requiere cambiar entre diagramas de objeto y de estados. Esta notación puede ser simple para diseños chicos pero se complica y se hace difícil de leer para sistemas más complejos.

AML, al separar el diseño en módulos de colaboración aspectual y todas las reglas de relación en un paquete *connector* separado asiste a clarificar la relación entre componentes.

Luego de este estudio exhaustivo de las técnicas de modelado de aspectos existentes hasta el momento consideramos que la más adecuada para representar el modelo conceptual multidimensional es AML porque es el que mejor cumple con los criterios de buen diseño enunciados anteriormente.

Otras notaciones han sido consideradas y estudiadas como por ejemplo: *"El Modelo de Suzuki y Yamamoto"* [Suz99], *"Introducing Separation of Aspects at Design Time"* [Her00] y *"A UML Notation for Aspect-Oriented Software Design"* [Paw02]. Estas están basadas en descomposición a nivel clase, lo cual no parece ideal porque varias clases están involucradas en un concern transversal. [Paw02] respeta UML estándar pero el alto acoplamiento de notaciones específicas a conceptos aspectuales haría difícil soportar otros lenguajes AO (como por ejemplo "*HyperJ"*); [Suz99] define correctamente la semántica pero no considera a los aspectos como abstracciones de primera clase: hace que el paquete de aspecto herede de la metaclase *Classifier* lo que provoca que esté al mismo nivel de abstracción que la metaclase *Class* de UML, esto dificulta el reuso de los aspectos a nivel de diseño. Por otra parte, peca de demasiado

abstracto al ser inespecífico en cuanto a cómo un advice o pointcut pueden ser modelados, provee principalmente conceptos para cruce transversal estático de operaciones. Por último [Her00] provee capacidades limitadas de modelado para operaciones de cruce transversal y los advices pueden sólo ser expresados a través de diagramas *"Statecharts"*.

En AML en cambio los aspectos y los elementos base se mantienen completamente aparte y son conectados por un conector específico del lenguaje que encapsula la implementación subyacente, y por lo tanto cualquier tecnología AO puede ser soportada. Ambos aspectos y elementos base se pueden reusar separadamente porque el conector es la única parte dependiente del lenguaje. Este tipo de encapsulamiento ofrece un agrupamiento lógico de todas las clases que pertenecen a un concern y facilita legibilidad de modelos de diseño evitando enredamiento gráfico. Además permite generar a partir del modelo planteado código Aspectual por lo tanto podemos, con un trabajo previo de refinamiento del modelo o contando con una ejemplificación particular, contar inmediatamente con un prototipo para testear su funcionalidad.

Como única desventaja podemos nombrar que la notación UML de esta propuesta introduce dos nuevas relaciones a nivel de paquete que son herencia y adaptación que no cumplen UML estándar por lo tanto sería útil definir un perfil de UML para soportarlas.

Trabajo reciente

Durante el curso de nuestra investigación hubo grandes avances en cuanto a técnicas de diseño de aspectos que consideramos importante estudiar y comparar con las técnicas previamente expuestas. Es por eso que en este apartado presentamos un resumen de las mismas y las evaluamos de manera sucinta con respecto a las propiedades de diseño previamente mencionadas. También las hemos considerado como posibles alternativas para el modelado de la extensión del metamodelo multidimensional expuesto en esta tesis.

Composable Designs with UFA [Her02]

Este enfoque es parecido a AML pero modela aspectos como paquetes abstractos que son simétricos con los paquetes que modelan funcionalidad común del sistema, por lo tanto los paquetes de aspectos representan abstracciones de primer orden, no así en AML donde el paquete de aspecto hereda de la metaclase *Classifier* de UML y no de *Class*.

Se basa en el modelo de colaboración aspectual [Lie99] y [Her01] que permite el desarrollo separado de módulos para posterior integración. Esta integración se realiza por medio de una relación de refinamiento y adaptación. Un paquete conector refina un paquete abstracto para inicializar un aspecto específico de la aplicación, luego el paquete conector adapta un paquete funcional. La relación de composición es directa, el paquete adaptado no conoce al paquete adaptador (el conector).

Los paquetes conectores son abstractos y son representados por un estereotipo de UML <<*connector*>>, del cual también definen la semántica. Al ser abstractos el lenguaje es inespecífico en cuanto a la especificación del cruce transversal, no define mecanismos de pointcut o advice lo cual si se hace en AML, por lo tanto, para que este modelo sea completo, es necesario que se le agreguen estos mecanismos u otros equivalentes para definir claramente las interacciones entre los aspectos y las clases base.

Se pueden eventualmente generar los perfiles de UML que permitan definir en este lenguaje de modelado los advices y los pointcuts, una aproximación a esto fue realizada aparentemente en [Gro03] en donde se definen los estereotipos para generar la implementación del modelo en *AspectJ*.

Los aspectos a diferencia de AML soportan abstracción y parametrización (se pueden generar aspectos template).

En cuanto al alineamiento a fases del desarrollo esta propuesta empieza por una descripción de la arquitectura del sistema, donde se identifican aspectos y funcionalidad principal. Los aspectos se refinan a través de un paquete conector a nivel de clase y método. Lo cual indica que esta propuesta está alineada a las fases de diseño e implementación.

Soporta trazabilidad de arquitectura a diseño y de éste a implementación, también soporta trazabilidad interna por medio de refinamiento a nivel de paquete bien definido para aspectos, clases y métodos.

Theme/UML [Clar01b]

Presenta un proceso para el desarrollo de software orientado a aspectos por lo tanto provee guías y artefactos para modelar el sistema desde los requerimientos hasta la implementación. Lo interesante a los efectos de este trabajo es cómo *Themes/UML* modela aspectos en la etapa de diseño y, eventualmente, como los integra.

En la **Figura 47** se puede apreciar como se desglosa un modelo de diseño con *Themes/UML*.

La cualidad principal de esta propuesta es que permite que un diseñador trabaje con *"themes"* (temas) individuales en forma separada.

Los temas son más generales que aspectos y elementos base en el metamodelo de UML. Un tema representa un concern y modela tanto estructura como comportamiento dentro de un paquete UML.

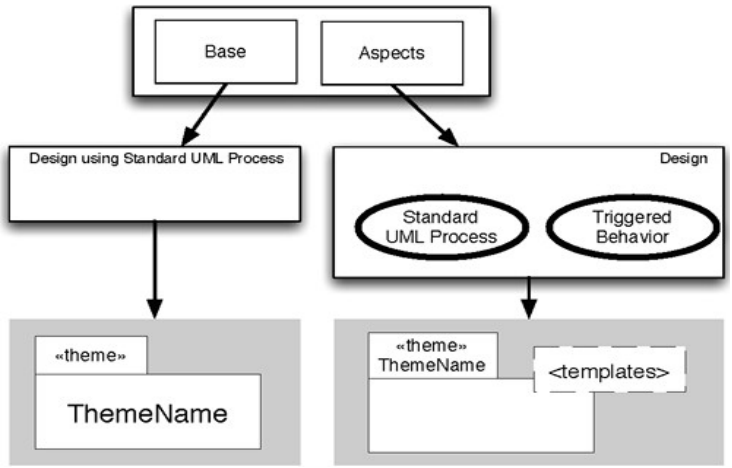


Figura 47: Modelo de diseño con Themes/UML **[Clar01b]**

La especificación de composición se modela como una relación entre temas. Los temas que son transversales se llaman temas aspectuales. Estos temas son plantillas que necesitan ser instanciadas vinculando sus parámetros, permitiendo que se apliquen en diferentes contextos.

En tiempo de composición los elementos reales a ser cruzados transversalmente reemplazan a las plantillas - sigue el modelo de patrones de composición para el lenguaje de diseño - [Clar01].

Soporta dos tipos de integración: mezclar y sobrescribir. La integración es utilizada para mezclar o sobrescribir un tema específico con otro.

La reconciliación resuelve problemas de compatibilidad reconectando elementos de temas diferentes que representan el mismo concepto o desconectando elementos que se corresponden accidentalmente cuando son usados. Este proceso compone los temas que forman el sistema. Para evitar conflictos en la composición se establece un orden de precedencia de temas.

La **Figura 48** muestra los pasos a seguir cuando se están componiendo temas.

En cuanto al alineamiento a fases del sistema usa temas independientes que mapean a concerns identificados durante los requerimientos. Cada tema puede ser definido separadamente y refinado a un diseño de bajo nivel detallado. Luego de la integración se puede implementar en una plataforma específica. Por lo tanto tiene alineamiento a fase de análisis, diseño e implementación.

Soporta trazabilidad desde los requerimientos hasta implementación. Presenta también trazabilidad interna porque los paquetes UML son refinados internamente por medio de varios tipos de diagrama UML. Usa semánticas de composición bien definidas para combinar temas en temas compuestos. Estos temas forman el diseño refinado del sistema que puede ser implementado.

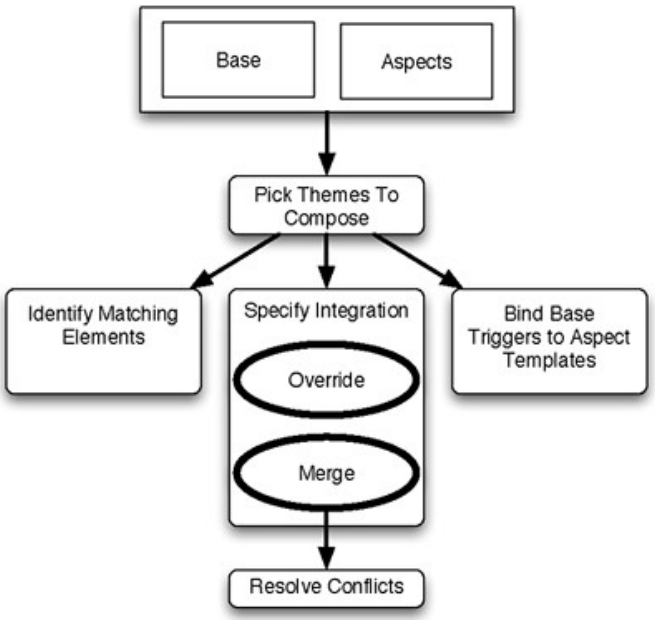


Figura 48: Vista genérica del proceso de composición de temas [Clar01b]

Soporta mapeo a implementación en diferentes lenguajes AO. Como por ejemplo AspectJ que agrega cuatro tipos de elementos de programa a Java: aspectos, pointcuts, advices y declaraciones adentro de tipos.

En la **Tabla 3** se describen estos elementos y se mapean a los elementos de diseño correspondientes en Theme/UML.

Tabla 3: Mapeo de alto nivel de los elementos de programación de AspectJ a Theme/UML

Elemento	AspectJ	Theme/UML
Aspect	Un aspecto es un tipo que contiene elementos de lenguaje que especifican estructura transversal y comportamiento. Puede ser instanciado en tiempo de ejecución al igual que una clase.	Un tema transversal es un diseño equivalente a un aspecto, aunque este podría también contener comportamiento que no es transversal.
Pointcut	Durante la ejecución de un programa, y como parte del alcance de esa ejecución, hay puntos en la ejecución donde el comportamiento puede ser juntado. Estos puntos son joinpoints. Un pointcut es un predicado que puede determinar, para un joinpoint dado, si se corresponde o no con dicho predicado.	Los parámetros de un template de operación pueden ser definidos y referenciados dentro de diagramas de secuencia que representen puntos en la ejecución de la secuencia de comportamiento donde éste puede ser juntado. Estos templates son reemplazados por (posiblemente múltiples) operaciones actuales como se especifica en el adjunto bind[] en el modelo de clases. Las especificación de templates y su especificación bind[] correspondiente son por lo tanto equivalentes a point-cuts.
Advice	Un advice es código que ejecuta en un joinpoint, dentro del alcance del contexto de ejecución.	Dentro de un diagrama de secuencia, el comportamiento transversal puede ser especificado para ejecutar cuando se llama a una operación template. Este comportamiento es equivalente al código de un advice.
Intertype declarations	Una declaración intertipo es un elemento de programa como un atributo, un método que es agregado a un tipo que puede	Los elementos de diseño que no son elementos template pueden ser definidos dentro de temas transversales. Estos pueden ser

Tabla 3: Mapeo de alto nivel de los elementos de programación de AspectJ a Theme/UML

Elemento	AspectJ	Theme/UML
	agregar o extender la estructura del tipo.	clases, atributos, operaciones o relaciones y son equivalentes a elementos que son agregados a los temas (o mezclados con estos).

AOD - Initial Version of Aspect Oriented Design Approach [Jac06]

Propone no sólo un lenguaje de diseño sino también un proceso completo de desarrollo de software aplicando aspectos siguiendo para cada fase los principios de MDA.

El lenguaje que propone es puramente orientado a diseño, totalmente independiente de plataforma, aunque también puede ser adaptado a una implementación particular. Este lenguaje está basado en el lenguaje Themes/UML que vimos anteriormente **[Clar01b]** cuyas principales características radican en el diseño de concerns funcionales y transversales como abstracciones de primera clase.

En cuanto al proceso de diseño incluye la habilidad de refinar módulos de concerns en sub-módulos hasta llegar al nivel de implementación.

En lo referente a la aplicación a nuestro metamodelo vemos que esta propuesta cumple con la mayoría de las propiedades enunciadas en el capítulo anterior pero sin embargo resulta ser una propuesta demasiado amplia y, a nuestro criterio, demasiada abstracción no ayudaría a clarificar la extensión del metamodelo que deseamos exponer.

UML Profile for Aspect-Oriented Software Development [Ald03]

UML soporta mecanismos de extensión como estereotipos, valores etiquetados y restricciones. Un conjunto predefinido de mecanismos de extensión es un perfil. El trabajo citado propone un perfil de UML para modelar aspectos en etapa de diseño.

La forma tradicional de representar sistemas AOSD vía notación UML es modelar aspectos como subclases de la metaclase *Classifier* [Suz99] del metamodelo de UML y luego estereotipar estos *Classifiers* para indicar si representan un concern transversal, una composición o una operación de vinculación (binding). Esta forma tiene la contra de que no hay un estándar establecido para extensiones de UML para acomodar las necesidades de AOSD.

Los perfiles propuestos en este trabajo modelan aspectos como abstracciones de primera clase para promover la reusabilidad en la fase de diseño. Esta extensión es soportada por UML estándar lo cual tiene la ventaja, frente a las otras propuestas estudiadas, que puede ser integrada fácilmente en herramientas CASE preexistentes que soporten UML.

El perfil propuesto va a ser capaz de expresar aspectos, clases funcionales base, y las asociaciones entre aspectos y entre aspectos y clases. Para ello introduce un nuevo estereotipo <<*aspect*>> para la clase base <<*class*>> que hereda de <<*classifier*>> element, de esta manera se garantiza que <<*aspect*>> se va a comportar igual que una instancia de <<*class*>>. Este formalismo habilita los aspectos a comportarse como "abstracciones de primera clase" que extiende automáticamente la reusabilidad de los aspectos.

Para cada aspecto define dos operaciones: *Preactivation* y *Postactivation*.

También estereotipa las relaciones de dependencia y asociación para permitir vincular las vistas de aspectos entre sí y con la funcional. El estereotipo genérico que se agrega para las relaciones entre aspectos y clases funcionales y para aspectos entre sí es <<*crosscut*>>.

A continuación en la **Figura 49** se presenta un ejemplo de modelado de un Buffer con este perfil. Este es un ejemplo de aplicación del estereotipo para modelar aspectos de "scheduling" (agendado), sincronización y manejo de errores en un *"Bounded Buffer"*. En el se puede apreciar el uso del paquete AOSD y de los conectores con el estereotipo *crosscut*.

Como la necesidad que plantean es crear un perfil para diseño y no para una implementación particular no se adentran en la especificación de cómo es la interacción en el cruce transversal. Sería útil en este caso extender el perfil para permitir incluir en el diseño los artefactos involucrados en esta interacción como point-cuts y advices (como es el caso de AML).

En conclusión tenemos aquí dos propuestas que proponen procesos completos de desarrollo de software modelados con aspectos como es el caso de Themes/UML y AOD – también basado en Themes/UML -. En general estos enfoques resultan demasiado amplios a la hora de elegirlos para extender nuestra propuesta de metamodelo y, en particular la utilización de patrones de composición, que fue analizada oportunamente, no es considerada la mejor en términos de los criterios de diseño que utilizamos para la selección de la herramienta.

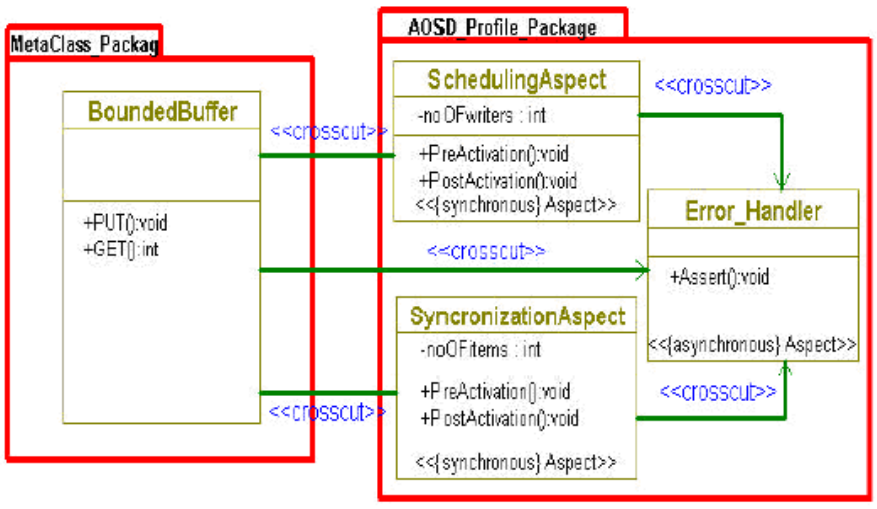


Figura 49: Diagrama de objetos del "Bounded-Buffer" usando el perfil AOSD [Ald03]

Por otro lado hemos analizado otros dos casos más: uno es UFA, un lenguaje de modelado muy similar a AML con la ventaja de que define la semántica de todos los elementos que utiliza y con la desventaja de que no define semánticas para diseñar point-cuts y advices, artefactos sí utilizados en AML. De esto concluimos que lo ideal en este caso sería proveer un perfil de UML como el propuesto en [Ald03] y extenderlo para modelar más específicamente el comportamiento transversal, [Gro03] ha definido algo de ese estilo.

Luego de haber recorrido las diferentes opciones y por razones de simplicidad, claridad y especificidad en el diseño hemos decidido insistir con la utilización de los artefactos de AML para el modelado de la extensión de AODW – el metamodelo conceptual aquí presentado - y quedará como trabajo futuro para aquellos tesistas que tomen como punto de partida este trabajo, la integración de todos los estereotipos y restricciones necesarias en un perfil UML para el modelado de aspectos que reúna las mejores características de AML, UFA, el profile UML propuesto [Ald03] y el perfil para mapeo de implementación [Gro03] de implementación.

AODW - Modelado de la seguridad en DW con aspectos

Como ya anteriormente mencionáramos la seguridad y el manejo de perfiles son concerns transversales en DW pero no menos importantes para modelar que el resto de las entidades. Es por eso que hemos decidido, para completar el metamodelo multidimensional, agregar el modelado de estos dos concerns tan importantes. Para ello utilizamos modelado de aspectos aplicando los estereotipos de AML, el lenguaje de modelado –extensión de UML para modelar aspectos- elegido en el capítulo anterior por sus bondades para permitir modelar un buen diseño orientado a aspectos. Así es como surge al fin, nuestro metamodelo multidimensional orientado a aspectos: AODW.

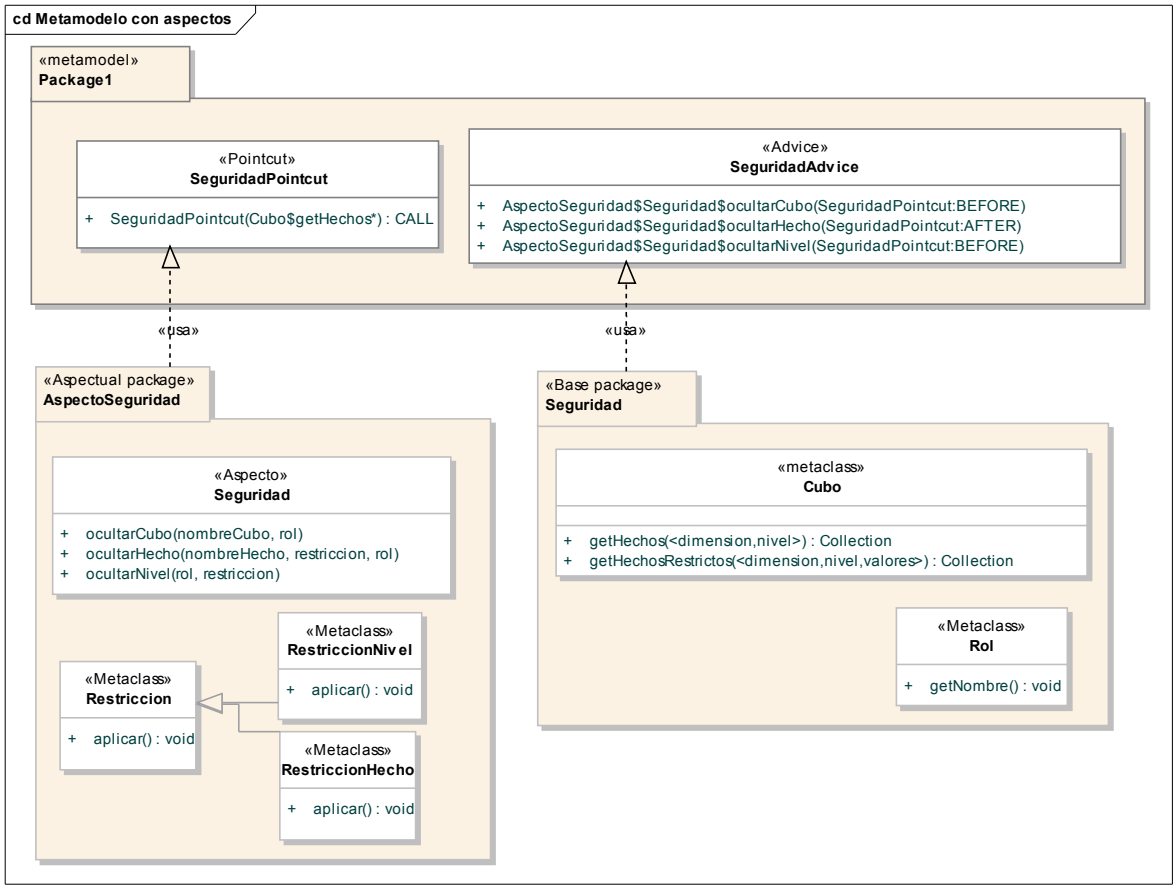


Figura 50: AODW - Metamodelo de seguridad extendido con aspectos

Como se puede apreciar en el diagrama, hemos definido para ello un paquete con la funcionalidad aspectual que contiene un aspecto llamado *Seguridad* con tres métodos definidos en él a modo de ejemplo: *ocultarCubo*, *ocultarHecho*, *ocultarNivel*. El tipo de política de seguridad que permite aplicar este aspecto es de "Mundo Abierto" –todo está permitido a menos que sea expresamente prohibido-. En el paquete *connector* reside realmente la funcionalidad aspectual que consiste de los pointcuts y los advices – en este caso un *"around advice"* - que definen dónde los métodos del aspecto de seguridad interceptarán a los mensajes enviados a la clase *Cubo* – punto de entrada al metamodelo – para restringir el acceso de un determinado rol a los datos almacenados

en él, permitiendo restringir a nivel de *Cubo*, *Hecho* y *Nivel* (se pueden definir métodos más complejos para interceptar otros accesos que aquí por motivos de simplicidad y alcance no son expuestos ni explicados).

Con este modelo pueden definirse **políticas de seguridad dinámicas** que son las que fueron expuestas como los requerimientos más avanzados de seguridad en el trabajo de Priebe y Pernul [Pri00]. En muchos casos estos requerimientos no han podido ser implementados en herramientas comerciales debido a su complejidad o posibilidad de generar resultados erróneos, imperceptibles al analista, como consecuencia de las restricciones sobre algunos datos.

Pero con AODW se podría ir más allá de la información estructural y definir políticas sensibles a los valores contenidos en ciertas celdas del cubo para determinadas medidas. Esto se lograría simplemente extendiendo la clase *RestricciónHecho* o creando una nueva subclase de *Restriccion*, por ejemplo: *RestricciónMedida* que en su método *aplicar()* implemente las reglas para aplicar una determinada política de seguridad sobre un valor determinado de una medida. Y aun cuando ésta sea obtenida por medio de un cálculo se podría definir otro "around advice" que se ejecute con la llamada al método que calcula el valor de la medida, luego, se analizaría en forma dinámica en tiempo de ejecución si esta cumple con las propiedades requeridas para mostrarla.

AODW puede también contribuir a solucionar fácilmente el problema de **inferencia de información** ya que utilizando los aspecto de *Seguridad*, *Restricción* y sus subclases se pueden definir restricciones que detecten cuando un usuario con un determinado rol intente acceder a una dimensión, nivel o medida determinada con precisión y sin necesidad de agregar atributos adicionales que "ensucien" las clases del metamodelo.

Por ultimo AODW ataca de una manera simple, transparente y eficiente el problema de ocultar la existencia de información sensible (**Hiding the existente**): En aplicaciones de seguridad crítica puede ser necesario ocultar la existencia de datos sensibles, ya no es suficiente devolver valores nulos a los usuarios no autorizados a ver ciertos hechos. No se trata ya sólo de un problema de seguridad, las propiedades de transparencia y facilidad para el usuario final son también afectadas. En general, en las herramientas comerciales, la seguridad en OLAP es implementada en la metadata, en miembros de dimensiones, o a nivel de datos de hechos. Sin embargo, los mecanismos de los diferentes sistemas difieren significativamente, también en sus capacidades de ocultar la existencia de datos sensibles. Esto por lo tanto termina convirtiéndose en una cuestión a resolver durante la implementación causada por una falencia en los modelos conceptuales utilizados – o la completa falta de tales modelos en la mayoría de los casos -. La solución que propone AODW para este caso es muy simple y está a la vista en el diagrama presentado: especializando las restricciones se puede esconder fácilmente – con los métodos *ocultar<nombre objeto>*- la existencia de determinadas medidas, dimensiones, niveles o hechos para determinados roles y lo más atractivo de esto es que resulta ser totalmente configurable por fuera del metamodelo.

Como conclusión podemos observar que en AODW, el metamodelo extendido que acabamos de presentar, no es necesario repetir los atributos de seguridad en cada clase del metamodelo como propone Trujillo y Piattini en [Vil05], sólo basta con agregarlos al aspecto de seguridad creado para tal fin y tampoco es necesario definir un perfil de seguridad, solamente con utilizar la extensión propuesta por AML – expresada en un perfil de UML por ejemplo extendiendo [Ald03]- basta para modelar

la seguridad utilizando el metamodelo. A partir del perfil de implementación que mencionáramos anteriormente [Gro03], el cual quedaría pendiente refinar, puede generarse código en lenguajes orientados a aspectos y también en lenguajes orientados a objetos en donde, obviamente, los conceptos transversales van a volver a mezclarse con los conceptos base, pero es una opción viable y da mayor apertura al espectro de posibles lenguajes para la implementación del modelado de seguridad en datawarehousing.

Los conceptos que hemos utilizado en este trabajo no son nuevos ni mucho menos, simplemente hemos logrado combinar en AODW la inteligencia proveniente de campos disímiles de investigación como lo son las áreas de datawarehousing, orientación a objetos y el conocimiento pragmático que suele adquirirse con la experiencia laboral.

El combinado AODW utiliza entonces conceptos provenientes de las áreas del datawarehousing obviamente, orientación a objetos (patrones de diseño), y orientación a aspectos más específicamente, de hecho, se podría decir que en AODW se utiliza fuertemente el concepto de *"adaptee unawareness"* que implica que el objeto adaptado, del metamodelo, no conoce al objeto que lo adapta, en este caso la restricción de seguridad modelada como un aspecto. Este principio está siendo también masivamente utilizado en frameworks orientados a objetos que han surgido en los últimos años, particularmente en el mundo *"open-source"*, con el objetivo de ensamblar componentes de diferente naturaleza de una manera cohesiva y liviana para lo cual utilizan el concepto de *"Inversion of Control"* (**IOC**), que asegura la propiedad de *"adaptee unawareness"* como por ejemplo *"Spring", "PicoContainer"* o *"HiveMind"* que implementan contenedores J2EE livianos.

Con AODW entonces, damos por terminado el alcance de esta tesis.

SECCION IV - Conclusiones y Trabajo Futuro

El siguiente paso en la búsqueda del modelado conceptual multidimensional estará dado naturalmente por el desarrollo de herramientas que soporten el metamodelo conceptual presentado. Con dicho fin se pueden utilizar los modelos de construcción y consulta expuestos en la sección II

Conclusiones

El concepto de multidimensionalidad en sí mismo, no nació de la comunidad de investigación, sino, por el contrario, como respuesta de las empresas de datawarehousing a las necesidades de gestión de los usuarios por medio de herramientas que fueron introducidas en el mercado para tal fin. A raíz de esto, carecían de una base teórica fuerte como en el caso de las bases de datos relacionales, y por ende los conceptos no estaban claramente definidos ni estandarizados. Sumado a esto los primeros esfuerzos de investigación se orientaron a mejorar la performance y la presentación al usuario final.

Frente a este escenario y a pesar de que la comunidad de investigación comenzó a tomar cartas en el asunto para brindar al modelado multidimensional la formalización necesaria, todavía no se ha llegado a la estandarización de un modelo conceptual multidimensional.

Es por eso que nuestra investigación se basó en un estudio detallado y exhaustivo de propuestas de modelado multidimensional, analizando para cada una sus fortalezas y debilidades sosteniendo siempre el paradigma de orientación a objetos como bandera de forma tal de poder explotar al máximo su fortaleza semántica en esta área. Esto nos dio las herramientas necesarias para definir el conjunto de propiedades deseables, que a nuestro entender, debería tener un *"Metamodelo Multidimensional Conceptual"*. Paralelamente, un estudio pormenorizado de los avances en el área de orientación a aspectos nos decidió a aplicarlo al modelado de seguridad para obtener una solución básicamente flexible y reutilizable ante un problema poco tratado en este área como el del modelado de políticas y mecanismos de seguridad en el área del datawarehousing.

UML fue seleccionado como lenguaje de modelado tanto para el metamodelo multidimensional que propusimos como para su extensión con aspectos. La formalización de la inclusión de OCL en UML 2.0 nos permitió utilizarlo inclusive para el modelado de las restricciones de integridad. Consideramos que fue la mejor opción en lenguajes de modelado por el nivel de estandarización alcanzado y por su flexibilidad para extenderlo a través de estereotipos lo cual nos ayudó a cumplir con nuestros objetivos: formalizar un metamodelo multidimensional orientado a objetos flexible y extensible y tener la posibilidad de hacerlo llegar a una amplia comunidad de usuarios que no sólo pueda utilizarlo sino también enriquecerlo.

Por último, ejemplificamos la instanciación de OODW basándonos en el caso de estudio presentado en esta tesis. Con esto esperamos haber mostrado la factibilidad, pero también la facilidad con la que se pueden generar modelos a partir de este metamodelo y su capacidad de extensión con los artefactos que extienden el metamodelo con aspectos.

Por lo expresado anteriormente, y por lo desarrollado a lo largo del presente trabajo, podemos afirmar que nuestra contribución más importante es en sí misma AODW, el "Metamodelo Multidimensional Orientado a Aspectos" que hemos presentado y que tiene la particularidad de combinar conceptos de orientación a aspectos aplicado al modelado de seguridad y perfiles, un tema también largamente postergado por la comunidad científica y superficialmente abordado por las herramientas existentes en el mercado. Con este metamodelo hemos encontrado una manera muy simple y muy

directa de generar modelos multidimensionales OO que contribuirá ampliamente a simplificar las tareas de diseño, desarrollo e implementación de datawarehouses.

Confiamos que los resultados de este trabajo sean adoptados y explotados por la comunidad de usuarios y eventualmente comercial.

Trabajo Futuro

Desde ya, este trabajo tiene varias líneas de investigación por las cuales se puede continuar.

Una de ellas comprende la generación los *"Profiles UML"* ("Perfiles UML") para OODW y AODW. Así estos metamodelos van a poder ser validados por herramientas CASE que soporten UML 2.0 y también existiría la posibilidad de generar de manera muy directa código para prototipos de sus instanciaciones. De esta manera OODW y AODW quedarían alineados al paradigma MDA.

Otra posible aplicación de este trabajo podría ser la extensión de herramientas del mercado por medio de desarrollo de plugins que, basados en los metamodelos y a partir de perfiles UML, puedan generar modelos multidimensionales e inclusive, con un poco más de elaboración en el mapeo a la capa de persistencia subyacente, herramientas de consulta multidimensional. Como se puede observar, el abanico de posibilidades de extensión es atractivamente amplio.

Otra línea de investigación que se podría seguir en particular y donde realmente queda mucho trabajo por hacer es en el área de seguridad en datawarehouses. Poco hay hasta el momento y creciente su necesidad de utilización. Cada día, las necesidad de seguridad son más complejas combinando roles de accesos, con permisos particulares a objetos, o a partes de ellos (como ser un *slice* de un cubo o un nivel de un cubo) como así también, permisos a diferentes niveles de agregación en modo dinámico como se explicó oportunamente al presentarse AODW en la Sección III de este trabajo, en lo referente al modelado e implementación de las **políticas de seguridad dinámicas**, consideradas requerimientos avanzados de seguridad según [Pri00].

Desde le punto de vista de la combinación de conceptos de orientación a aspectos en el modelado conceptual multidimensional podemos destacar otras líneas de investigación abiertas aparte de la seguridad y control de acceso:

Restricciones de diseño

Los aspectos pueden resultar útiles también para expresar aquellas restricciones en los estados del modelo de diseño. Restricciones operacionales que expresan relaciones entre alternativas del espacio de diseño y modos de operación del sistema. Por ejemplo en nuestro metamodelo con UML no podemos expresar que los objetos dimensionales de un nivel sólo pueden relacionarse con objetos dimensionales de otro nivel diferente. Si contáramos con un lenguaje de especificación de restricciones o una extensión de UML que pudiera expresar esto podríamos reflejar estas restricciones en etapa de diseño. Se puede utilizar para eso lo propuesto en [Gra01]. Otra opción más simple y comúnmente adaptada en los modelos actuales sería expresar las restricciones con OCL al construir perfiles en UML 2.0 para los metamodelos previamente presentados.

Tiempo de respuesta

Este es un aspecto relacionado con un requerimiento no funcional muy importante en el modelado de bases de datos multidimensionales ya que a la gran cantidad de los datos almacenados es importante optimizar el tiempo de respuesta de las aplicaciones. Como una base de datos multidimensional no se actualiza muy a menudo lo que se hace actualmente es tener la mayoría de las consultas precalculadas de antemano. Pero puede suceder que se desee ejecutar una consulta que no fue calculada antes o que esté precalculada pero se desee un resultado más actualizado, en estos casos sería bueno tener la libertad de poder elegir qué tipo de consulta queremos hacer. Un aspecto que permita elegir el camino para calcular la consulta en tiempo de ejecución sería muy útil en este caso.

Bibliografía

[Ac04] Rodolfo Villarroel Acevedo, "Modelado Conceptual de la Seguridad en Data Warehouses", Abril 2004.

[Agr95] Agrawal, R; Gupta, A, Sarawagi, S. "Modeling Multidimensional Databases", 1995.

[AGR97] R. Agrawal, A.Gupta, A. Sarawagi. "Modeling Multidimensional Databases" – ICDE'97. 1997

[Alb00] Abelló, A; Samos, J; Saltor,F. "A Data Warehouse Multidimensional Data Models Classification", 2000.

[Alb01] Alberto Albelló, José Samos, Felix Saltor (2001). "Benefits of an Object-Oriented Multidimensional Data Model". In Objects and Databases – Int. Symposium-, Volume 1944 of LNCS, pages 141-152. ECOOP'00, Sophia Antipolis – France. 2001.

[Alb01b] Alberto Albelló, José Samos, Felix Saltor (2001). "Understanding Analysis Dimensions in a Multidimensional Object-Oriented Model". International Workshop on Design and Management of Data Warehouses (DMDW'2001) – Interlaken, Switzerland. 2001.

[Alb01c] Alberto Albelló, José Samos, Felix Saltor (2001). "Understanding Facts in a Multidimensional Object-Oriented Model". DOLAP'01 - Atlanta, Georgia, USA. 2001.

[Alb02] Abelló, A; Samos, J; Saltor, F. "YAM2 : A Multidimensional Conceptual Model". Phd Thesis, Universidad Politécnica de Cataluña, 2002.

[Arbo96] Arbor Software Corporation. Arbor Essbase. http://www.arborsoft.com/essbase.html. 1996.

[Buz98] J. W. Buzydlowski, I. Song, L. Hassell. "Framework for Object-Oriented On-line Analytical Processing". Workshop on Data Warehousing and OLAP –DOLAP'98 – Washington DC USA. 1998.

[CAB98] L. Cabibbo and R. Torlone (1998). "A logical Approach to Multidimensional Databases". In advances in Database Technology. 1998

[COD93] E.F. Codd, S.B. Codd y C.T. Salley. "Providing On-Line Analytical Processing (OLAP) to User-Analysts: An IT Mandate". Internet: http://www.arborsoft.com/OLAP.html. 1993

[Cut03] Cutter Consortium. 41% HAVE EXPERIENCED DATA WAREHOUSE PROJECT FAILURES. http://www.cutter.com/research/2003/ Edge030218.html, 2003

[CWM02] Object Management Group (OMG). "Common Datawarehouse Metamodel (CWM)". Specification 1.1. Internet: http://www.omg.org/cgi-bin/doc?ad/2002-03-03. 2002.

[CZA00] Krzysztof Czarnecki, Ulrich Eisenecker, Krzysztof Czarnecki. "Generative Programming – Methods, Tools, and Applications". Addison-Wesley. 2000.

[Elr01] Elrad, T., Filman, R. et al; "Aspect Oriented Programming"; Communications of the ACM, October 2001

[Eng00] Gregor Engels & Luuk Groenewegen. "Object-Oriented Modeling: A Roadmap" Gregor Engels & Luuk Groenewegen. 2000

[Fra03] Frankel D. S. "Model Driven Architecture. Applying MDA to Enterprise Computing". Indianapolis, Indiana. Wiley 2003.

[Gam95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides; "Design Patterns. Elements of Reusable Object-Oriented Software". 1995

[Gol98] Matteo Golfarelli, Darío Maio, Stefano Rizzi. "The Dimensional Fact Model: A Conceptual Model for Data Warehouses". In International Journal of Cooperative Information Systems, vol. 7, number 2-3. 1998.

[Gol99] Matteo Golfarelli, Stefano Rizzi. "A Methodological Framework for Data Warehouse Design". DOLAP'98 – Washington DC, USA. 1999.

[Gop99] V. Gopalkrishnan, Q. Li, K. Karlapalem. "Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies". 1999.

[Gra01] Jeff Gray, Ted Bapty, Sandeep Neema, James TUC. "Handling Crosscutting Constraints in Domain-Specific Modeling". 2001

[Gre02] Robin Green, Awais Rashid. "An Aspect-Oriented Framework for Schema Evolution in Object-Oriented Databases". 2002.

[HAC97] M.S. Hacid, U. Sattler. "An Object-Centered Multi-Dimensional Data Model with Hierarchically Structured Dimensions". Proc. of the IEEE Knowledge and Data Engineering Workshop. 1997.

[Har93] William Harrison, Harold Ossher. "Subject-oriented programming: a critique of pure objects". Conference on Object Oriented Programming Systems Languages and Applications,Washington, D.C., United States. Pages: 411-428. ACM Press; 1993.

[Hyp01] Hyperion. "An Overview of Hyperion's Data Warehousing Methodology". dev.hyperion.com/resource_library/white_papers/data_warehousing_methodology.pdf; 2001.

[Info97] Informix, Inc.: "The INFORMIXMetaCube Product Suite". http://www.informix.com/informix/products/new_plo/metabro/metabro2.htm, 1997.

[Inm92] W.H. Inmon. "Building the Data Warehouse". QED Press/John Wiley, 1992.

[JAC01] Arno Jacobsen, H.; "Middleware architecture design based on aspects, the open implementation metaphor and modularity"; Aspect Oriented Programming and Separation of Concerns – Proceedings of the International Workshop – August 2001.

[JOLAP02] JSR-69 (JOLAP) Expert Group - JavaTM 2 Enterprise Edition - JavaTM OLAP Interface (JOLAP) - Version 0.85, Initial Public Review Draft,; 2002.

[Kat03] Mika Katara, Shmuel Katz. "Architectural views of aspects". Proceedings of the 2nd international conference on Aspect-oriented software development; Boston, Massachusetts; Pages: 1 – 10. 2003.

[KEL00] Keller, D. "Data Warehousing: El Ciclo de Vida Dimensional Orientado a Objetos". 2000.

[Kic97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. "Aspect Oriented Programming". Published in proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. 1997.

[Kic01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, William G. Griswold. "An Overview of AspectJ". Lecture Notes In Computer Science; Vol. 2072. Proceedings of the 15th European Conference on Object-Oriented Programming; pages 327-353. 2001.

[Kim96] Ralph Kimball et al. "The Data Warehouse Toolkit: The Complete Guide To Dimensional Modeling", John Wiley & Sons Computer Publishing. 1996.

[Kim98] Ralph Kimball et al. "The Data Warehouse Lifecycle Toolkit", John Wiley & Sons Computer Publishing. 1998.

[Klep03] Kleppe A., Warmer J, Bast W. "MDA Explained. The Practice and Promise of The Model Driven Architecture". Addison Wesley, 2003.

[Lar99] Craig Larman. "UML y Patrones – Introducción al análisis y diseño orientado a objetos", Prentice Hall. 1999.

[Lem03] Daniel Lemire, "A Research-Oriented Bibliography". http://www.cs.toronto.edu/~mendel/dwbib.html; 2003.

[Maz05] Jose-Norberto Mazón, Juan Trujillo, Manuel Serrano, Mario Piattini . "Applying MDA to the Development of Data Warehouses". 2005.

[Maz05b] Mazón J-N, Trujillo J., Serrano M., Piattini M. "Designing data warehouses: from business requirement analysis to multidimensional modeling". 13th IEEE International Requirements Engineering Conference Workshop on Requirements Engineering for Business Needs and IT Alignment (REBNITA). Paris. August, 2005. http://homepage.mac.com/karlalancox/documents/MazonCRC_000.pdf

[MDA04] Object Management Group (OMG). "Model Driven Architecture (MDA)". http://www.omg.org/mda. 2004.

[Med02] E. Medina, S. Luján Mora, J. Trujillo. "Handling Conceptual Multidimensional Models using XML through DTDs". 2002.

[Med05] Medina E., Trujillo J. "A Standard for Representing Multidimensional Properties: The Common Warehouse Metamodel (CWM)". In proceedings of the 6th East-European Conference on Advances in Databases and Information Systems (ADBIS'02), volume 2435 of Lecture Notes in Computer Science, pages 232-247, Bratislava, Slovakia; Springer-Verlag. 2005.

[Mel04] Mellor S, Scott K., Uhl A., Weise D. "MDA distilled: principles of Model-Driven Architecture"; Addison-Wesley. 2004.

[Mor02] S. Luján Mora. "Multidimensional Modeling using UML and XML", 2002.

[Mor02b] S. Luján Mora, J. Trujillo, I. Song. "Extending UML for Multidimensional Modeling". 2002.

[Mor02c] S. Luján Mora, J. Trujillo, I. Song. "Multidimensional Modeling with UML Package Diagrams". 2002.

[Mor03] S. Luján Mora, J. Trujillo, I. Song. "Applying UML for designing multidimensional databases and OLAP applications". 2003

[Mor04] Luján-Mora S., Trujillo J. "A Data Warehouse Engineering Process". In Proceedings of the 3rd Biennial International Conference on Advances in Information Systems (ADVIS'04). Lecture Notes in Computer Science, Izmir, Turkey; Springer-Verlag. 2004.

[Mor04b] S. Luján Mora, J. Trujillo, P. Vassiliadis. Advantages of UML for Multidimensional Modeling. Abril 2004.

[Mor04c] S. Luján Mora, J. Trujillo, I. Song. "Applying UML and XML for designing and interchanging information for data warehouses and OLAP applications". 2004

[Mor05] Luján-Mora S., Trujillo J., Song I-Y. "Extending UML for Multidimensional Modeling". 5th International Conference on the Unified Modeling Language (UML 2002), LNCS 2460, 290-304. 2005.

[MStr95] MicroStrategy, Inc. Relational OLAP:An Enterprise-Wide Data Delivery Architecture. White Paper: http://www.strategy.com/wp_a_i1.htm. 1995.

[MStr97] MicroStrategy, Inc. MicroStrategy's 4.0 Product Line. http://www.strategy.com/launch/4_0_ar msc1.htm. 1997.

[Mus03] David R. Musser. "Generic Programming". http://www.cs.rpi.edu/~musser/gp/, 2003.

[Nav02] A. Navasa, M. A. Perez, J. M. Murillo, and J. Hernandez, "Aspect-Oriented Software Architecture: a Structural Perspective", presented at Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, junto con AOSD Conference, 2002.

[Ngu00] Thanh Nguyen, A Min Tjoa, Roland Wager. "An Object Oriented Multidimensional Data Model for OLAP". 2000.

[Ngu01] Thanh Nguyen, A Min Tjoa (2001). "Conceptual Multidimensional Data Model Based on Object Oriented Metacube". SAC 2001, Las Vegas, NV. 2001.

[OMG04] "UML 2.0 OCL Specification". OMG, 2004.

[Pre97] Christian Prehofer. "Feature Oriented Programming: A Fresh Look at Objects". Institut fur Informatik Technische Universitat Munchen; Munchen, Germany. 1997.

[Pri00] Torsten Priebe, Günther Pernul, "Towards OLAP Security Design – Survey and Research Issues", 2000.

[Pri02] Torsten Priebe, Günther Pernul, "A Pragmatic Approach to Conceptual Modeling of OLAP Security", 2002.

[QVT] OMG 2nd Revised Submission: MOF 2.0 Query/Views/Transformation. http://www.omg.org/cgi-bin/doc?ad. 2002.

[Ras01] Awais Rashid. "A hybrid Approach to Separation of Concerns: The Story of SADES". 2001.

[Ras01b] Rashid, A., Green, R. "An Aspect-Oriented Framework for Schema Evolution in Object-Oriented Databases". 2001.

[Ras01c] Awais Rashid, Peter Sawyer. "Object Database Evolution using Separation of Concerns". 2001

[Ras02] J. Araújo, A. Moreira, I. Brito, A. Rashid. "Aspect-Oriented Requirements with UML". Workshop on Aspect-oriented Modeling with UML, UML 2002, Dresden, Germany, 2002.

[Ras03] Awais Rashid, Ana Moreira, João Araújo. "Modularisation and composition of aspectual requirements". Proceedings of the 2nd international conference on Aspect-oriented software development; Boston, Massachussets. Pages 11-20. 2003.

[Rav99] Franck Ravat, Olivier Teste, Gilles Zurfluh. "Towards Data Warehouse Design". CIKM'99 – Kansas, USA. 1999

[Rav00] Franck Ravat, Olivier Teste. "Object-Oriented Decision Support System". Université Paul Sabatier – IRIT/SIG. 2000

[RedB97] Red Brick Systems, Inc.. Red Brick Warehouse 5.0. http://www.redbrick.com/rbsg/html/whouse50.html, 1997.

[Suv03] Davy Suvée, Wim Vanderperren, Viviane Jonckers. "JasCo: an Aspect-Oriented approach tailored for Component Based Software Development". 2003

[Tan02] Eric Tanter. "Más alla de la orientación al objeto. Reflexión, metaprogramación y programación por aspectos". Reflexión, metaprogramación y AOP – JCC 2002; Copiapó, Chile. 2002.

[Tes01] Olivier Teste. "Towards Conceptual Multidimensional Design in Decision Support Systems". Université Paul Sabatier – IRIT/SIG. 2001.

[Tho97] Erik Thompson. "OLAP Solutions - Building Multidimensional Information Systems", John Wiley and Sons. 1997.

[Tru98] Trujillo, J., Palomar, M. "An Object Oriented Approach to Multidimensional Database Conceptual Modeling (OOMD)". Proc. of ACM 1st International Workshop On Data Warehousing And OLAP (DOLAP'98); Bethesda, Washington D.C., USA. 1998.

[Tru99] Juan Trujillo. "The GOLD Model: An Object Oriented Multidimensional Data Model for Multidimensional Databases". 1999.

[Tru00] Trujillo, J., Palomar, M., Gómez, J. "Detecting Patterns and OLAP operations in the GOLD Model". 2000.

[Tru01] J. Trujillo, S.Luján Mora, E. Medina. "Utilización de UML para el modelado multidimensional". 2001.

[Tru01b] Trujillo, J., Palomar, M., Gómez, J., Il-Yeol Song. "Designing Data Warehouses with OO Conceptual Models". IEEE Magazine, Pages 66-75, December 2001.

[UML03] Object Management Group (OMG). Unified Modeling Language (UML). Specification 1.5. http://www.omg.org/cgi-bin/doc?formal/03-03-01. 2003.

[Vas99] Panos Vassiliadis, Timos Sellis. "A Survey on Logical Models for OLAP Databases". 1999

[Vil05] Rodolfo Villarroel, Eduardo Fernández-Medina, Mario Piattini, Juan Trujillo. "A UML 2.0/OCL Extension for Designing Secure Data Warehouses". 2005

[Wal99] Robert J. Walker, Elisa L.A. Baniassad and Gail C. Murphy. "An Initial Assessment of Aspect-oriented Programming". 1999.

[Wan03] Lingyu Wang, Yingjiu Li, Duminda, Wijesekera, Sushil Jajodia. "Precisely Answering Multi-dimensional Range Queries Without Privacy Breaches". 2003.

[Wir90] Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener. "Designing Object-Oriented Software". Prentice Hall PTR. 1990

[Wix01] Wixom B., Watson H.J. "An Empirical Investigation of the Factors Affecting Data Warehousing Success"; MIS Quarterly, 2001.

[You95] Yourdon et. Al. "Objects: An analysis and Design Approach for Business", Yourdon Press. 1995

ANEXO A – Glosario

Acme Lenguaje de descripción de arquitectura simple y genérico (ADL) que puede ser usado como un formato de intercambio común entre herramientas de diseño de arquitectura o como fundación para desarrollo de nuevas herramientas de diseño de arquitectura y análisis.
ACP (Attribute Classification Paths) - definición de jerarquías de atributos dimensionales en [Tru99].
Ad-Hoc El sistema permite al usuario personalizar una consulta en tiempo real, en vez de estar atado a los queries prediseñados para reportes.
ADAPT notación propietaria de ORACLE Express para describir modelos multidimensionales.
ADAPTed UML Extensión de UML que utiliza símbolos ADAPT como estereotipos UML.
ADL (Architecture Description Language) Lenguaje de descripción de arquitecturas.
AML Extensión de UML para modelado de aspectos.
AOP (Aspect Oriented Programming) Programación orientada a objetos.
Architectural Style Estilo de architectura.
Archiving Almacenamiento de la historia.
ARR (Attribute Roll-up) Formalización de la dimensión con su jerarquía.
ARRP (Attribute Roll-up Relation Paths) Definición de Jerarquías de atributos Dimensionales en [Tru99].
AS (Architectural Style) Estilo de arquitectura.
Assembler (programa ensamblador) Es un programa para traducir lenguaje ensamblado (ver assembly language) a código objeto.
Assembly Language Usa representación mnemotécnica del lenguaje máquina.
Asynchronic Message Brokering Adaptación o intermediación asincrónica de mensajes. En el contexto de informática o telecomunicaciones se refiere a una entidad como un middleware o un bus que intermedia el intercambio de mensajes de manera asíncrona (por ej. CORBA).
AVG Average, promedio.
Bases de Datos Multidimensionales Son bases de datos orientadas a la implementación de estructuras de datos para gestión y análisis.
Batch (AS) Procesamiento por lotes sin intervención del usuario.
BDM Ver Bases de Datos Multidimensionales.
Bind/Binding Vincular.
BI Business Intelligence Es una amplia categoría de aplicaciones y tecnologías para reunir, proveyendo acceso a, y analizar datos con el propósito de ayudar a los empresarios a la toma de decisiones. El término implica un amplio conocimiento de factores tales como clientes, competidores, socios, ambiente económico y operaciones internas para tomar decisiones de negocio acertadas.
Bounded-Buffer Monitor de sincronización del estilo Productor/Consumidor.
Buffering Memoria intermedia de almacenamiento temporal.
C++ Lenguaje de Programación OO.
CASE Computer Aided Software Engineering. En la jerga computacional CASE es el uso de soporte electrónico en el proceso de desarrollo de software.
Caching Un mecanismo de almacenamiento rápido especial. Puede ser una sección reservada de la memoria principal o un dispositivo de almacenamiento independiente de alta velocidad.
Call and return (AS) Arquitectura de llamada y retorno. Engloba los estilos: Programa principal y subrutina, sistemas orientados a objetos, niveles jerárquicos.
Celda Intersección de las caras de un Cubo que contiene los objetos de análisis, los hechos.
CIM Computation Independent Model – [Maz05].
Class Key Clave que combina la hoja de una jerarquía con los hechos.
Collapse Ver Combine.

Combine Operación que permite cerrar la visualización de atributos de calificación en un paso de navegación.
Consultas Ad-Hoc Ver Ad-Hoc.
Conforming Dimension Es una vista única y coherente del mismo conjunto de datos a través de toda la organización. La misma dimensión es utilizada en todos los esquemas estrella definidos subsecuentemente [Kim96].
Constellation / Constelación Variante del Esquema Estrella.
Constraint Languages Se corresponde con el paradigma de **Constraint Programming** donde las relaciones entre las variables pueden ser establecidas en forma de restricciones. Difieren de las primitivas comunes de otros lenguajes de programación en que no especifican un paso o secuencia de pasos para ejecutar sino más bien las propiedades de la solución a encontrar.
Copo de Nieve o Snowflake Es un modelo que nace a partir del Esquema Estrella pero un un grado mayor de complejidad en su diseño.
Cubo de Datos Es una metáfora que representa cómo los analistas visualizan los datos y la navegación a través de los mismos.
CWM Common Warehouse Model.
Darwin es un lenguaje de descripción de arquitectura (ADL) para describir arquitecturas basadas en componentes. Es escalable porque soporta un modelo jerárquico y soporta notación gráfica.
Data Mart Datawarehouse orientado a una unidad de negocio.
Data Mining Es un conjunto de técnicas para la inducción de conocimiento útil a partir de masas ingentes de datos. Se suele aceptar que la minería de datos estudia información acumulada en empresas y otras organizaciones acerca de sus clientes, en búsqueda de patrones de comportamiento que permitan realizar acciones dirigidas a estos clientes.
Datawarehouse Almacén de Datos.
Datawarehousing Diseño y construcción de un Datawarehouse.
Degenerate Dimension Dimensión con una cardinalidad similar al hecho.
Denial Denegación de permiso para acceder a un recurso de un sistema.
DBMS DataBase Management System.
DFM Dimensional Fact Model propuesto por [Gol98].
Dimensión / Dimension Cara de un Cubo, a través del cual se accede al objeto de análisis.
Divide Operación que permite visualizar atributos de calificación en un paso de navegación.
Drill Across Operación que permite navegar entre dos esquemas estrellas o cubos de datos.
Drill Down Operación que permite desglosar a un nivel mayor de detalle.
DW Datawarehouse.
DWH Datawarehouse.
E/R Modelo entidad relación.
Esquema Estrella (Star Schema) Es el modelo lógico de facto que se utiliza actualmente para modelar cubos multidimensionales, bases de datos multidimensionales.
ETL (Extraction, Transformation and Loading). Se refiere a los procesos de extracción, transformación y carga de un datawarehouse.
Expand Ver Divide.
Extent Objeto raíz.
Fact Objetivo de análisis.
Fact Constellation Ver Constellation.
Fact Instance Expressions Lenguaje de expresiones del modelo DFM.

Feature-Oriented Development Es el estudio de la modularidad de características, donde estas características son entidades de primera clase en diseño. Las carácterísticas (features) son implementadas por funcionalidad transversal que consisten en modificaciones o extensiones a múltiples clases. A diferencia de AOP que focaliza en la cuantificación – i.e. especificación de join points en donde insertar el código – las implementaciones de características son mucho más cercanas a los diseños de frameworks orientados a objetos. Para agregar una característica (feature) a un framework OO, hay métodos predefinidos y clases que lo van a extender. En tales diseños hay poco o nada de cuantificación, pero hay de toda forma funcionalidad transversal.
Feature-Oriented Programming (FOP) Ver Feature-Oriented Development.
Front-End Fachada de una aplicación.
Generative Programming Programación Generativa. Un estilo de programación que usa creación automática de código fuente a través de clases genéricas, templates, aspectos y generadores de código para mejorar la productividad de los programadores.
Generic Programming Programación genérica. Es una forma de programar independiente de los tipos de datos. Esto significa que el mismo código fuente va a ser instanciado con diferentes parámetros sin importar su tipo. La programación genérica está implementada y soportada de manera diferente según el lenguaje. Ejemplos de lenguajes que siguen en parte este paradigma son: Ada, C++, Eiffel, Java.
Graphical User Interface Interfase de un programa que utiliza las capacidades gráficas del lenguaje para hacer el programa más fácil de usar.
Grant Permiso de usuario (pueden variar según la política de seguridad aplicada)
Granularidad Nivel de detalle de un modelo multidimensional.
GOLD Modelo propuesto por [Tru99].
GUI Acrónimo para Graphical User Interface.
Hecho Ver Fact.
Hide Esconder
Historical Truth Verdad Histórica, forma de análisis con dependencia de tiempo.
HiveMind ES un framework para crear aplicaciones basado en un micro núcleo de servicios y configuración.
HOLAP Hybrid OLAP.
IOC Ver Inversion of Control
Inversion of Control Es un principio importante de la programación orientada a objetos. En un framework de inversión de control los objetos no se crean sino que se describe cómo deben ser creados. Los componentes y los servicios no se conectan por medio de código, sino que se estable en descriptores qué componentes necesitan qué servicios.
JAVA Lenguaje de programación OO.
J2EE son las siglas de *"Java 2 Enterprise Edition"* que es la edición empresarial del paquete Java creada y distribuida por Sun Microsystems. Comprenden un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales. Debido a que J2EE no deja de ser un estándar, existen otros productos desarrollados a partir de ella aunque no exclusivamente.
KPI Key Performance Indicators.
Logging Registro o bitácora de accesos/actividad en un sistema o aplicación.
Marshalling traducción de los argumentos de una función a un formato determinado.
Metadata Datos sobre los datos.
MDA Model Driven Architecture.
MD^{2}A MultiDimensional Model Driven Architecture framework MDA propuesto por [Maz05].
MDDM Modelo para bases de datos multidimensionales.

MDX (Multidimensional Expressions). Es un lenguaje de consulta creado por Microsoft para consultar objetos multidimensionales tales como cubos, y retorna conjuntos de celdas multidimensionales (cellsets) que contienen los datos del cubo.
Modelo Conceptual Es el modelo más cercano a la percepción que tiene el usuario del negocio, independiente de su implementación.
Modelo Conceptual Multidimensional Es un modelo conceptual orientado a bases de datos multidimensionales.
Modelo Estrella Ver Esquema Estrella.
Modelo Físico Un Modelo de datos estrechamente asociado a la Base de Datos donde se implementará la solución.
Modelo Lógico Modelo de datos que provee conceptos que si bien pueden ser entendidos por el usuario, no se encuentran tan lejos de cómo los datos serán almacenados.
MOF Meta Object Facility.
MOLAP (Multidimensional Online Analytical Processing). Utilizada para denominar soluciones OLAP sobre entornos de BD Multidimensionales Propietarias.
MultiCubo Consultas que combinan o toman como fuente dos cubos.
Multidimensional OLAP Ver MOLAP
Multiestrella Ver Constellation.
O3LAP Modelo propuesto por [Buz98].
OCL (Object Constraint Language). Un lenguaje formal para expresar restricciones definido poe el OMG – Object Management Group.
OLTP OnLine Transactional Processing.
OLAP OnLine Analytical Processing.
OQL Object Query Language.
OOMD Object Oriented Multidimensional Database Modeling. Modelo propuesto por [Tru98].
OO Object Oriented.
Open-source Una aplicación open-source es aquella que tiene una licencia de libre distribución, que incluye el código fuente, que permite modificaciones y trabajo derivado y no discriminar a ninguna persona o grupo que requiera utilizarlo entre otras propiedades. Para una definición más amplia consultar http://www.opensource.org/docs/definition.php.
Open World Policy O Política de Mundo Abierto, se refiere a la política de seguridad en donde todo está permitido a menos que sea explícitamente prohibido.
Overlapping Schema Ver Constellation.
Part-Whole Relationship Relación de composición.
PicoContainer es un contenedor de aplicaciones liviano y altamente imbebible para componentes que respetan IOC.
PIM Platform Independent Model - [Maz05].
Pipes&filtres (AS) Tuberías y Filtros. Es un popular estilo de arquitectura que consiste en una tubería (pipeline) que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas.
Pivoting Operación Multidimensional que implica la rotación de dimensiones.
Poder Semántico Expresividad alcanzada.
Política Query-set size control Es una política de seguridad que consiste en rechazar todas las consultas que no caen dentro de un determinado umbral.
PSM Platform Specific Model - [Maz05].
QVT Query, View, Transformation.
Rapide Es un ADL especializado en modelar arquitecturas para sistemas de tiempo real, distribuidos.

Relational OLAP Ver ROLAP.
ROLAP (Relational OnLine Analytical Processing). Utilizada para denominar soluciones OLAP sobre entornos de BD Relacionales.
Roll-up Operación Multidimensional que permite subir un nivel de desagregación.
Show mostrar.
Slice and Dice Operación multidimensional que permite filtrar dimensiones y desglosar.
Slowly Changing Dimension Dimensión que va modificándose a lo largo del tiempo [Kim96], [GIO00].
Smalltalk Lenguaje de programación OO.
Snowflake Variante del esquema estrella.
Spring Es un framework de aplicaciones Java/J2EE full-stack Full-stack significa que puede utilizarse en varios niveles de la aplicación como por ejemplo persistencia, negocio o capa de control de una aplicación Web.
SQL Structured Query Language.
Statechart Son diagramas que permiten documentar varios modos o estados por los que puede pasar una entidad, y los eventos que causan esas transiciones de estados.
Template Plantilla
Time-stamping Método para asociar fecha y hora a la modificación de un dato determinado.
Tracing Trazado, en el sentido de rastreo hacia la fuente.
UFA Un enfoque abstracto para generar modelos de diseño AO.
UML (Unified Modeling Language). Lenguaje de especificación, visualización y construcción de los objetos de un sistema de software.
Viewpoints Es una vista del sistema desde una perspectiva posible en donde se focaliza en un conjunto de concerns específico. Por ejemplo se puede ver a un framework para sistemas distribuidos desde el punto de vista del usuario o desde el punto de vista de las reglas de negocio. En contraste, también se puede ver al framework desde una perspectiva estructural, funcional o secuencial.
Virtual Machine (AS) Es un estilo de arquitectura que se utiliza cuando un componente debe ser construido en forma independiente del software o hardware subyacente.
WarGen (Warehouse Generador). Prototipo para la generación automática de warehouses propuesto por [Rav99].
Wrigh (ADL) Un ADL de propósito general desarrollado por Robert Allen y David Garlan.
XMI XML Metadata Interchange.
XML Extended Markup Language.
YAM² Yet Another Multidimensional Model – Propuesta de [Alb02].

ANEXO B – Introducción a la POO

Presenta un resumen de los conceptos de programación orientada a objetos a manera de introducción para el lector que no posea conocimientos previos del tema.

Conceptos de la POO

Programa
(En el paradigma de objetos) es un conjunto de objetos que colaboran entre sí enviándose mensajes.

Objeto
Modelo(abstracción) de algo que existe en la realidad.

Abstracción
La acción de abstraer consiste en observar un objeto de la realidad y tomar sus características principales. Describir en función de la esencia.

Clase
- Template y Factory: Las clases nos permiten describir el comportamiento genérico de un conjunto de objetos que representan el mismo tipo de componente del sistema (template) y también crear esos objetos (factory).

- Clasificación: Permiten construir una taxonomía de objetos a un nivel abstracto y conceptual.

Instancia
Los objetos que son creados por una clase y se comportan de la manera especificada en una clase se llaman instancias de esa clase.

- Todas las instancias de una clase comparten comportamiento y utilizan los mismos métodos para responder a los mensajes especificados en la clase.

- Cada instancia de una clase tiene su propia parte privada constituida por un conjunto de variables de instancia cuyos nombres están definidos en la clase.

Subclase
Es una clase que hereda comportamiento de otra clase. Su propósito es agregar comportamiento propio para definir su único tipo de objeto. Ejemplo: subclasificación de *String* a partir de *Array*. La subclasificación de *String* a partir de *Array* y *Magnitude* para poder ordenar el *String* lexicográficamente sería un ejemplo de herencia múltiple.

Herencia o subclasificación (mecanismo de abstracción)
Es la habilidad de una clase de definir el comportamiento y las estructuras de datos de sus instancias como un superconjunto de la definición de otra clase o clases (herencia múltiple).

- La herencia nos provee un mecanismo de clasificación, con el cual podemos crear taxonomías de clases.

- Se puede ver una nueva clase como un refinamiento de otra, para abstraer las similitudes entre clases y para diseñar y especificar sólo las diferencias para la nueva clase (programar por diferencia).

- La herencia nos permite reusar código, una clase que hereda de otra hereda su comportamiento.

- Hay herencia estricta y herencia no estricta, el primer caso es cuando las subclases heredan todo el comportamiento de la superclase, en el segundo no.

- Otra clasificación de la herencia puede ser en simple y múltiple: todo problema solucionable con herencia múltiple lo es también con herencia simple.

Clase abstracta
Clases que tienen comportamiento abstracto común a una jerarquía, no son instanciables. Pueden tener mezcla de comportamiento abstracto y comportamiento concreto (template methods) o pueden tener sólo comportamiento abstracto como las interfaces de Java. Sólo existen para que el comportamiento común a varias clases sea factorizado y ubicado en un lugar común en donde sólo tenga que ser definido una vez y reusado una y otra vez.

Creación de objetos
Puede ser por clasificación: instanciando una clase, o por prototipación: clonando un prototipo.

Encapsulamiento
El acto de agrupar en un único objeto tanto datos como comportamiento que afecta esos datos.

Forma una barrera conceptual alrededor de esas operaciones y esos datos.

Funciona como una ayuda para conceptuar y abstraer partes del dominio del problema y así tratar con su complejidad.

Transforma muchos en uno mediante el agregado de un nuevo nivel de conocimiento. Ejemplo de los 7 números que se transforman en un número de teléfono.

Information Hiding
El objeto tiene una interfaz pública y una representación privada, a esto se lo llama *"Information Hiding"*. Como cada objeto puede modificar su representación sin afectar al resto del sistema este concepto es útil para crear código fácilmente modificable, mantenible y extendible. Diferencia la habilidad para realizar ciertas acciones de los pasos específicos para llevarla a cabo.

Los objetos saben solamente que operaciones pueden pedir a otros objetos, no saben cómo las llevan a cabo. Diríamos que actúan como buenos jefes.

Encapsulamiento e "*information hiding"* trabajan juntos para aislar una parte del sistema de las otras.

Acceso a los objetos

Mensaje
Un mensaje consiste en el nombre de una operación y los argumentos requeridos.

El conjunto de mensajes a los que un objeto puede responder son conocidos como el comportamiento del objeto.

Cuando un objeto recibe un mensaje envía mensajes a otros objetos.

En general cuando un objeto recibe un mensaje no sabe qué objeto se lo manda. El acoplamiento se da en el sentido objeto emisor - > objeto receptor pero no al revés.

Cuando un objeto le manda un mensaje a otro abre un canal de comunicación entre los dos.

El resultado de recibir un mensaje casi siempre es otro objeto.

Método
Un método es un conjunto de colaboraciones o un conjunto de mensajes que un objeto le manda a otros y que se ejecutan en respuesta a la recepción de un mensaje cuyo nombre coincide con el nombre del método. Siempre es parte de la representación privada del objeto.

Signatura de un mensaje
Es el nombre del método, los tipos de los parámetros, y el tipo de objeto que el método devuelve. Es una especificación formal de las entradas y salidas de un método.

Polimorfismo (mecanismo de abstracción)
Es la habilidad de dos o más clases de objetos de responder a los mismos mensajes cada uno con sus propios métodos. Un objeto manda un mensaje: si el receptor implementa un método con la misma signatura va a responderlo.

Protocolo
Un objeto está determinado por lo que sabe hacer y eso que sabe hacer está determinado por los *mensajes que puede recibir/responder*, ese conjunto de mensajes es el protocolo del objeto.

Responsabilidades
Determinan el propósito del objeto en el sistema y constan de:

- El conocimiento que un objeto mantiene.
- Las acciones que puede realizar.

Colaboración

- Representa un requerimiento de un cliente a un servidor para cumplir una responsabilidad del cliente.
- Decimos que un objeto colabora con otro si, para cumplir una responsabilidad, necesita mandarle algún mensaje al otro objeto.
- Es la materialización del contrato entre un cliente y un servidor.
- Una única colaboración fluye en un solo sentido, representado un requerimiento de un cliente a un servidor.
- Un objeto puede cumplir con una responsabilidad valiéndose por si mismo o puede requerir de la asistencia de otros objetos, en ese caso necesita colaborar con ellos.

- Encontrar las colaboraciones puede ayudar a identificar las responsabilidades de los objetos y viceversa encontrar las responsabilidades de los objetos puede ayudar a encontrar colaboraciones que faltan.

Contrato
Define un conjunto cohesivo de requerimientos que un cliente puede hacer a un servidor. El contrato debe garantizar que el servidor puede responder a dichos requerimientos. Una clase puede soportar uno o mas contratos distintos.

Method-Lookup (búsqueda de método)
La búsqueda de un método que responda a un mensaje sigue la cadena de la superclase, terminando en la clase Object. Si no se encontró un método que coincida con el mensaje se le manda al receptor el mensaje doesNotUnderstand:, el argumento es el mensaje en cuestión. Existe un método para el selector doesNotUnderstand: en la clase Object que reporta el error al programador.

Self vs Super
Self y super son pseudovariables. Ambos representan al receptor del mensaje pero si se utiliza self el method-lookup comienza en la clase del receptor mientras que si se utiliza super el method-lookup comienza en la superclase de la clase del receptor.

Comparación de lenguajes de objetos puros contra lenguajes híbridos
Lenguajes OO puros versus híbridos: en los primeros todo es un objeto en los segundos también hay tipos de datos que no son objetos, los cuales deben ser encapsulados en clases para tratarlos según el paradigma de objetos.

Tipado estático vs. Tipado dinámico: En el primero el compilador chequea los tipos de todos los argumentos, receptores y tipos de retorno cuando se compila un envío de mensaje, si los tipos no son correctos el compilador tira un error. El tipado dinámico sólo ocurre durante la ejecución

Manejo automático de memoria: En un programa orientado a objetos generalmente se crean gran cantidad de objetos durante la ejecución. Si la memoria que usan estos objetos no es reclamada el software se va a quedar eventualmente sin memoria. En lenguajes como Smalltalk o CLOS un objeto se considera sin uso cuando ya no está más referenciado por otros objetos.

El manejo automático de memoria evita tener que conocer la estructura interna de los objetos y sus dependencias y es menos propenso a errores.

Ambiente de programación que permita:

- Browsear código
- Compilación incremental
- Ejecución de expresiones interactivamente
- Debugging interactivo del código
- Inspeccionar los estados de los objetos.

Componentes o Toolkits
Son entidades lo suficientemente generales como para ser utilizadas en diferentes programas. Por ejemplo List, Array y String son componentes, RadioButton y CheckBox también. Obviamente para los programas que los usan funcionan como cajas negras.

Framework
La meta más importante al diseñar un framework es hacerlo refinable. Las interfaces hacia las aplicaciones deben ser lo más claras y precisas posibles. Los frameworks son cajas blancas para los que hacen uso de ellos. Consisten en un diseño y código reusable.

Aplicación
Las aplicaciones son programas completos. Por ejemplo un procesador de texto o una calculadora son aplicaciones.

La meta principal al desarrollar aplicaciones es hacerlas fácilmente mantenibles. Lo ideal es desarrollar una aplicación a partir de la instanciación de un framework.

Subsistema
Un subsistema es un grupo de clases y/o otros subsistemas que colaboran entre ellos para soportar un conjunto de contratos.

Desde afuera del subsistema, el grupo de clases puede ser visto trabajando muy cohesivamente para proveer una unidad funcional claramente delimitada, para cumplir determinados contratos con sus clientes.

Desde adentro, los subsistemas tienen una compleja estructura. Están formados por clases y otros subsistemas que colaboran entre ellos para soportar distintos contratos que contribuyen a la funcionalidad total del sistema.

Ejemplo de subsistema: subsistema de impresión, subsistema de cobro de una máquina expendedora.

No existe diferencia conceptual entre un subsistema y una clase, ambos soportan contratos. Para determinar los contratos soportados por un subsistema, hay que encontrar todas las clases que proveen servicios a los clientes fuera del subsistema. Los contratos de estas clases son los contratos soportados por el subsistema. Hay que establecer los contratos de un subsistema todos al mismo nivel de abstracción.

Los subsistemas sólo soportan los contratos conceptualmente, en tiempo de ejecución cada subsistema delegará el cumplimiento de los contratos a las clases correspondientes que realmente los soportan.

Los subsistemas sirven para simplificar el diseño de una gran aplicación. Si primero se identifican los subsistemas ya se los puede tratar igual que cajas negras como si fueran otras clases.

Como los clientes utilizan la funcionalidad de un subsistema a través de un conjunto claramente definido de contrato, su funcionalidad puede ser extendida sin alterar el resto de la aplicación. Un nuevo contrato puede ser definido o uno existente extendido para soportar funcionalidad adicional.

Metaprogramación
Un metaprograma es un programa que manipula otros programas (o él mismo) como sus datos. El ejemplo canónico es un compilador.

Reflección
Un programa que se manipula o examina a sí mismo es llamado reflectivo. La reflección puede ser estructural o de comportamiento.

Estructural: se tiene acceso a la estructura del programa. Ej: clases, métodos y campos.

De comportamiento: se tiene accesos a los aspectos de tiempo de ejecución por ejemplo cómo los objetos son creados o cómo los métodos son llamados.
La reflección se puede dar en tiempo de compilación, tiempo de carga o tiempo de ejecución.

Introspección
Significa la capacidad de examinar propiedades del programa.

Jerarquía de clases y metaclases
Una metaclase es un objeto que describe una clase.

Patrones de diseño
Un patrón de diseño describe un problema recurrente en un contexto y luego describe la clave de la solución, de tal manera que se pueda usar esta solución una y otra vez en el futuro.