16.

DIMBO

Direct Marketing With Business Object

Trabajo de Grado

Lucio Dinoto

1997

Licenciatura en Informática

Universidad Nacional de La Plata

Directores :

Dr. Gustavo Rossi

Dr. Fausto Simonelli

INDICE - Trabajo de Grado

Lucio Dinoto
Directores : Dr. Gustavo Rossi y Dr. Fausto Simonelli

B. Modelos de Análisis del Negocio (Business Analysis Models) y Modelos de Análisis de Objetos (Object Analysis Models)

C. Modelos de Diseño e Implementación

D. An approach and an experience working with use cases and business objects (presentado workshop #32 OOPSLA 97)

5. Referencias

Introducción

Los Objetos de Negocio en la Pequeña y Mediana Empresa: La Gestión de Clientes y el Marketing Directo

Lucio Dinoto

Motivación e Idea Básica

Deseamos proveer a los pequeños comerciantes y a los gerentes de las pymes, de una herramienta capaz de realizar una gestión inteligente del universo de los clientes actuales y también de los potenciales, de cada una de las empresas.

Dicha herramienta es el vehículo conductor de una **Estrategia de Gestión de Clientes** que permita concretar con alto grado de certeza y anticipación, los objetivos comerciales y económicos de la Pyme, utilizando la funcionalidad de las técnicas del **Marketing Directo**, que le permiten al pequeño empresario conocer en cada momento **el Estado de Situación comercial, financiera, social, económica y de marketing** de cada cliente en particular, así como de cada uno de los grupos de ellos, según corresponda.

Descripción

Dicha herramienta **está desarrollada bajo un modelo de objetos**. Debido a los fuertes cambios que impone la turbulencia del contexto económico, social y comercial a las pymes, así como la constante evolución existente en las técnicas
de gestión del mercado, el diseño debe ser cuidadosamente desarrollado para alcanzar altos grados de flexibilidad, modificabilidad, reutilizabilidad y extensibilidad.
El Modelo deberá ser útil de igual modo, para desarrollar una Estrategia de Gestión de Clientes en una empresa de mayor envergadura transitando caminos similares. Pero es necesario hacer notar que el proceso final de esta tesis será la implementación de la Herramienta en los comerciantes y en las pymes.

Interrogantes a resolver

Uno de los aspectos principales a resolver es el caracter genérico del modelo. El usuario trabajará con clientes al aplicar sus políticas de gestión, pero **¿qué es un cliente? ¿Se trata de la misma cosa en una estrategia que en otra**? La herramienta deberá ser suficiente

para ajustarse a las necesidades del **Usuario** (el empresario Pyme), a la hora de manipular la base de datos de sus clientes. Por lo tanto, dicho Usuario no podrá verse limitado bajo circunstancia alguna, en el acceso a la información o en la interacción que desee, con sus clientes.
La flexibilidad antes mencionada, debe permitirle, a través del Modelo, un creciente grado de aplicación de creatividad a la ejecución de sus Estrategias Comerciales.

Ambiente de Desarrollo

El Paradigma de Objetos es el elegido para el Desarrollo de este Trabajo, poniéndolo bajo un **Ambiente Puro de Objetos** como es el **Smalltalk-VisualWorks**. El proceso de ingeniería de software será conducido por la metodología de Jacobson, ya sea la propuesta en "The Object Advantages" como la definida en "OOSE", intentando dejar una fuerte documentación del proceso de desarrollo de la Herramienta, lo que será seguramente muy útil en las futuras extensiones que se realizarán.

Futuro y consideraciones de la herramienta

Sin pensar en Versiones Extendidas de la Herramienta, es decir, considerando solamente la Implementación del Proceso Final de la Tesis, puede darse por razonable la idea de la existencia de un importante mercado potencial que podría estar interesado en el uso de la misma y con quien evaluar su funcionalidad y evolución. Ese mismo mercado que muy raramente utiliza plataformas de objetos para sus procesos de negocios. Por otro lado, se incursionará en un mundo poco explorado: el de los business objects.

Funcionalidad desde la perspectiva del marketing

En las empresas argentinas, incluso en las grandes, no se realiza la gestión comercial **a partir de la administración concreta y monitoreada de la relación de marketing con cada uno de los clientes**, a lo sumo se **lo controla** casi policialmente desde el lado de los pagos, los cobros, las deudas, etc.
Lo que no se hace, es diseñar una Estrategia General de los Negocios a partir de dicha Administración Monitoreada de la relación de marketing con cada uno de los clientes, estableciendo para cada caso los respectivos objetivos comerciales y económicos asignando a la vez los recursos correspondientes alcanzando así los resultados totales de la empresa.
Por ejemplo: el cliente XZY debe alcanzar en el ejercicio actual un nivel dado de facturación, que implicará un nivel dado de rentabilidad, un nivel dado de cumplimiento en cuanto a fluir de fondos (cash-flow), un nivel adecuado de trabajo de ventas (capacitación y desarrollo del trabajo de los vendedores que se le asignaren para atenderlo, etc.).
A partir de la Proyección de Demanda anual que implica la sumatoria de los objetivos asignados a cada uno de los clientes, tendremos la posibilidad de construir los presupuestos

mensuales, trimestrales, anuales y bianuales. Ello estará inscripto en el Plan de Gestión a tres y a cinco años que es el punto inicial del Planning de la empresa a Largo Plazo.

Para cumplir con lo mencionado en cuanto a la gestión de clientes: se hacen imprescindibles las acciones del marketing directo en cada uno de los clientes que componen la base de datos.
Surgen así una gran cantidad de preguntas, como por ejemplo:
¿Qué informaciones necesito?
Una ficha de cliente en la cual se sostengan datos diversos como la fecha de la primera compra (antigüedad como cliente), las compras posteriores realizadas con sus fechas, otras formas de interacción con el cliente (reclamos, nuevos pedidos, etc.) etc. Donde el "etc" puede comprender cualquier tipo de información que el comerciante considere importante respecto de su cliente.
Mediante los análisis de las propias estrategias de gestión, es vital que la herramienta provea, por ejemplo: comparaciones entre performances de cada cliente contra los parámetros predeterminados (compró **X** mientras que el objetivo que se le ha asignado es el de comprar **Y**), los Estados de clientes actuales, activos, inactivos, potenciales y todos ellos comparados contra los parámetros definidos, para medir performances.
Es importante que la herramienta provea de elementos inteligentes que avisen sobre determinados hechos de interés. Estos comportamientos son parte de las estrategias de gestión, pero son controlados por la propia herramienta.
La misma puede considerar los diferentes niveles de operación en una pyme. El nivel mínimo implica apertura total en el acceso a la ficha del cliente en términos personales y la historia del último ejercicio. El nivel más alto corresponde a la gerencia, quien determina la ejecución y el control de cada estrategia puesta en marcha.

Los clientes deben ser calificables, esto es, debe agregarseles un código de calificación en función a parámetros como cumplimiento en los pagos, montos de facturación, potencialidad en su desarrollo como cliente teniendo en cuenta por ejemplo, si puede generar ventas importantes como contacto (un director de escuela privada para un textil que vende sacos azules y pantalones grises). Se deben buscar diversos parámetros para construir el modelo "de calificación" , que podría salir automáticamente para restringir toda posibilidad de "dibujo" del código buscado.
Por cada cliente se deberían tener los datos de su familia lo mas completos posible respecto de gustos personales, inclinaciones, hobbies, estudios cursados, clubes a los que asiste, actividades sociales a las que se dedica, cartas de crédito que posee, algun código donde se puedan escribir libremente apostillas, chismes y datos que redondeen un perfil del sujeto en cuestión y de su flia. La herramienta debe considerar que toda información que se quiera sostener de un cliente pueda ser agregada, así como funcionalidad y medidas sobre dicha información, todo esto, obviamente en forma dinámica. P. Ej.:
Parámetros de gestión: Productividad, nivel de producción, costos unitarios., costos fijos y costos variables. Punto de equilibrio, ventas en pesos, utilidades, rentabilidades.

Pasos seguidos

- Estudio de bibliografía de gestión de marketing directo para capturar procesos y métodos que se utilizan para hacer este tipo de marketing. Definición y elección de estrategias o políticas para la gestión.
- Definición de procesos comerciales (estrategias, políticas de negocios, etc.) con modelos de objetos utilizando Object Advantages approach compactado a un handler: el manager de marketing.
- Captura de procesos automatizables y desarrollo de modelos de objetos para los mismos.
- Construcción de la herramienta en VisualWorks.

Tutores elegidos por el autor

Para la captura, testeo, evaluación y evolución de la funcionalidad de la herramienta, el Doctor Fausto Simonelli.
Para el area de desarrollo de los modelos de objetos, el Doctor Gustavo Rossi.

Aspectos Teóricos

Marketing Directo - Conceptos Generales

El principal objetivo del marketing directo es **optimizar el diálogo entre la empresa y el cliente**, creando un ida y vuelta creciente en beneficio de ambos.
Mediante el uso de uno o más canales de comunicación (correo, tv, diarios, venta directa, etc.), intenta aumentar la valorización de sus clientes actuales e incorporar nuevos clientes a su rutina.
Suele confundirse el marketing directo simplemente como la venta por correo, pero ésta es solo una modalidad de distribución del marketing (el medio) a veces utilizada por el marketing directo para provocar una acción en el receptor.

La característica humana más creciente en los últimos tiempos es la desmasificación. Y el marketing directo saca ventajas de esta realidad concentrándose en segmentos de mercados conseguidos muchas veces gracias a la tecnología y la informatización del conocimiento sobre el mercado (clientes). Y aún más, entre los centros de compras más significativos de los último años ha apareceido la tarjeta de crédito y el teléfono. Y ambos son medios muy utilizados por las estrategias de marketing directo.

El marketing directo es un sistema de comunicación interactiva que usando uno o más medios publicitarios se dirige a un público perfectamente individualizado con el fin de motivarlo hacia una acción, casi siempre de compra, para que otorgue finalmente al empresario un resultado medible, que sea coherente con los planes trazados previamente. Así entonces, el marketing directo aparece como un instrumento y rentable y relativamente **poco costoso** que además, ofrece la ventaja de obtener resultados concretos y apreciables en un lapso muy corto, es decir, el que dura cada campaña específica. El marketing directo obliga a la empresa a desarrollar su filosofía empresarial partiendo desde el cliente, logrando una serie de informaciones que permiten conocerlo mejor, anticipar sus necesidades y sus exigencias.

Ventajas del Marketing Directo

- Confidencialidad entre la empresa y el cliente.
- Mide resultados entre clientes-productos
- El marketing tradicional propone y seduce. El marketing directo, **vende** directamente.
- Se dirige hacia un público bien preciso y segmentado.
- Crea clientes mientras vende.
- Es invisible para la competencia.
- Crea, actualiza, mantiene y usa una base de datos de información de clientes.

¿En qué se utiliza el marketing directo ?

- Venta directa empresa - clientes.

- Canal de distribución para servicios post-venta y productos afines.
- Como canal de ventas de productos poco motivantes para la fuerza de ventas.
- Como canal para llegar a zonas no fácilmente accesibles para la empresa.
- Venta de productos o servicios a segmentos muy reducidos.
- Para generar bases de datos de clientes posibles y potenciales.
- Para conocer las necesidades y preferencias de los clientes mediante "premios a las respuestas"

¿Qué consigue el marketing directo ?

- Animar ventas de temporada.
- Personalizar mensajes para el consumidor, alentando acciones rápidas.
- Calificar la relación cliente-producto.
- Segmentar masas.
- Mejorar la fidelidad del cliente con la empresa pudiendo afinar las acciones acorde a los intereses de cada cliente.
- Evita pérdidas imprevistas al dirigirse a un destino bien conocido.
- Singularizar la relación empresa-cliente, haciendo sentir a este último un ser reconocido por la empresa.

El proceso del marketing directo cuenta con dos subprocesos claves : el marketing front-end y el marketing back-end.

Marketing Front-End

Objetivo :

Obtener un determinado número de clientes al menor costo posible.

Formas :

- Venta Directa utilizando un medio.
- Envío de ofertas a clientes potenciales muy probables.
- Cadena de amistad mediante premios y descuentos a los clientes promotores.
- Solicitudes gratuitas.

Marketing Back-End

Objetivo :

Rentabilizar al máximo el valor del cliente.

Formas :

- Conservar fidelidad del cliente manteniéndolo activo como comprador.
- Fomentar el crecimiento del volúmen de compra.
- Fomentar el crecimiento de la frecuencia de compra.
- Generar ventas cruzadas o enganchadas.

¿Qué significa segmentar clientes ?

Segmentar es seleccionar al público objetivo y separarlo y distinguirlo del resto del universo considerado que constituye el mercado total. Esto genera las listas de personas sobre las cuales se realizarán las distintas acciones de marketing. Un segmento representa un perfil de cliente determinado.

Para segmentar, el manager debe :

- Definir las variables claves que muestran el comportamiento frente a un producto o servicio.
- Identificar y separar los clientes con mayor potencial o afinidad al respecto.
- Seleccionar los grupos más valiosos.
- Definir una estrategia comercial a cada perfil.

Patterns - Conceptos

Un pattern describe un problema que ocurre una y otra vez en un ambiente determinado y su solución para ese problema de manera tal que la misma pueda ser utilizada cientos de veces y de diferentes maneras sin tener que pensar nuevamente en la forma de alcanzar dicha solución.
Los patterns constituyen uno de los últimos "hot-topics" de la comunidad de objetos, aunque el concepto de patterns haya surgido de la Arquitectura y los mismo puedan ser aplicados sobre cualquier paradigma más allá de la tecnología de objetos.
Un pattern es una regla de tres partes que expresa la relación entre un contexto, un problema y su solución. Sin embargo no toda solución o buena práctica es un pattern. Antes, necesita ser comprobado como un fenómeno recurrente, preferentemente en tres diferentes sistemas. Esta es la llamada *rule of three*.

Propiedades de los patterns

- Documentan experiencias de diseño probadas exitosamente.
- Identifican y especifican soluciones abstractas.
- Proveen vocabulario común y entendimiento de los principios de un buen diseño.
- Documentan arquitecturas de software.
- Combaten la complejidad del software.

Clases de Patterns

Analysis Patterns o Conceptual Patterns : Son descriptos en términos conceptuales desde el dominio en que se trabaja (Martin Fowler, Analysis Patterns)

Design Patterns : Son descriptos mediante constructores y propiedades del paradigma de objetos (herencia, polimorfismo, abstracción, etc.)

Arquitectural Patterns : Expresan la estructura de una organización y sus relaciones.

Pattern Language : Representa una colección de patterns que conforman un vocabulario para comunicar y entender ideas de un contexto común.

Componentes de los patterns en general

- **Nombre**. Lo suficientemente significativo. El conjunto de nombres representan nuevos lenguaje para quienes trabajan con patterns.
- **Problema**. Una declaración de la situación describiendo metas y objetivos a los que se quiere llegar dentro de las fuerzas y el contexto definido.
- **Contexto**. Especifica las precondiciones de aplicabilidad.

- **Fuerzas**. Las motivaciones y las restriciones para la solución en el contexto dado.
- **Solución**. Puede ser textual o bajo algún sistema gráfico de relaciones. Identifica la estructura, los participantes y las colaboraciones entre ellos.
- **Ejemplos**.
- **Patterns relacionados**.
- **Usos conocidos**.

Patterns de diseño

Los patterns de diseño son descripciones de clases y objetos comunicantes, customizados para resolver un problema general de diseño en un contexto particular. Estos patterns nombran, abstraen e identifican aspectos claves de una estructura de diseño que la hace útil para crear un diseño reusable. Un pattern identifica que clases participan, cuales son sus roles y de que manera colaboran y distribuyen sus responsabilidades.
Cuatro elementos se destacan de los patterns de diseño :

Un **nombre**, el cual describe el problema de diseño que el pattern resuelve y generalmente se trata de una o dos palabras y debe ser, por si solo, muy significativo.

El **problema** donde el pattern es aplicado. Esto incluye la situación que se presenta y su contexto, así como la lista de posibles condiciones para justificar la aplicación del pattern.

La **solución** al problema antes planteado, estableciendo objetos, relaciones, colaboraciones y responsabilidades. Esta solución no debe incluir ningún detalle implementatitvo ya que debe abstraerse de la plataforma donde se vaya a aplicar el pattern,

Y las **consecuencias** que tiene la aplicación del pattern. Aquí, ventajas y desventajas pueden aparecer, así como una evaluación de las diferentes alternativas de diseño.

Más específicamente, un pattern contiene los siguientes elementos (Design Pattern - Gamma, Helm, Johnson y Vlissides) : Nombre y Clasificación, Intención, Conocido también como..., Motivación, Aplicabilidad, Estructura, Participantes, Colaboraciones, Implementación, Código de Ejemplo, Usos conocidos y Patterns relacionados.

La gran ventaja de los patterns radica en el hecho de que brindan a los desarroladores de software de una literatura común que le permite encontrar las mejores soluciones a problemas comunes y recurrentes.
Inicialmente los patterns de diseño han sido utilizados para la arquitectura y el desarrollo de sistemas, extendiéndose en los últimos tiempos a los procesos de negocios y las organizaciones.

Patrones de Negocio y Objetos de Negocio. Conceptos

Objetos de Negocio

Un objeto de negocios (business object) es un objeto que representa una persona, lugar, evento o concepto en el dominio del negocio. Los business objects sirven como el medio de representación de políticas de negocios (financieras, estructurales, organizativas, etc.), reglas y datos.
El propósito de desarrollar un conjunto de business objects es proveer al mercado de ojetos que sirvan como estándares para recurrentes problemas del propio dominio de negocios. De esta manera un conjunto de business objects filtrado, depurado y mejorado en el proceso de instanciación de diferentes aplicaciones, se convierte en un business framework para el dominio en cuestión.
Otra importante característica cuando se trabaja sobre business patterns y business objects es que tanto managers (usuarios) como desarrolladores acortan la brecha que los separan conceptualmente, logrando de esta manera hablar en el "mismo lenguaje". Esto significa que los modelos obtenidos sirven tanto para el perfil de los managers como para el de los desarrolladores.
Existen tres tipos diferentes de business objects : entidades, eventos y procesos. Los primeros representan, por ejemplo, gente, lugares, empleados, cuentas, etc. Los eventos representan los límites de tiempo que surgen en los negocios, tales como ciclo de vida de un producto, fecha de settlement en una transacción financiera, etc. Estos objetos aparecen en el principio y en el final de la interacciones entre entidades. Por último aparecen los procesos, quienes definen la interacción entre varios objetos en una manera predecible, estable y recurrente con el objeto de producir una resultado significativo para el negocio.

Objetos de Negocio y Patrones de Negocio

Los patrones de negocio son patrones de análisis (ver referecia Business Analysis Patterns - M.Fowler). Los patrones de negocio y los objetos de negocio son complementarios entre si. La principal diferencia es que si alguien compra un paquete de business objects (por ejemplo, para manejo de transacciones financieras), los objetos tienen que ser usados tal cual han sido representados en la implementación adquirida. Si bien todo el framework puede ser modificado, el costo es lo suficientemente alto (en general) como para no intentarlo. Por el contrario, los patrones de negocio no brindan implementación alguna, sino que tan solo una especificación de los problemas que rodean al dominio comercial y la "mejor" forma de resolver los mismos.

Patrones de Negocio

Otra vez, los patrones de negocio son patrones de análisis. Un patrón de análisis (analysis pattern) es un modelo y una discusión de porque el modelo y de la forma que se lo plantea. El desarrollador es libre de implementar el pattern de la forma que el desee. De esta manera, un analysis pattern documenta un conjunto de business objects. Pero la principal ventaja de tener analysis patterns para describir generalidades del negocio de la forma más detallada, se fundamenta en el hecho de que funcionan como una poderosa herramienta para entender y mejorar el propio proceso comercial que representan.
Los patrones de análisis proveen a los managers un medio estándar de entender el negocio y su futura (o actual) implementación sin tener que capacitarse para entender las notaciones de objetos que los diferentes procesos de análisis y diseño especifican. Así, los managers pueden dedicarse a hacer reingeniería de sus procesos de negocios a través de los analysis patterns.

Un approach y una experiencia trabajando con business objects

El desarrollo de esta tesis se ha basado en el approach que se describe a continuación. Aquí, el autor de esta tesis ha trabado como analista y diseñador de toda la aplicación, por lo cual, dicho approach podría encajar adecuadamente en proyectos de mediana envergadura que posea un escaso número de recursos humanos involucrados.
El tutor de esta tesis especializado en el área de marketing fue quien condujo los requerimientos funcionales de la aplicación. Como desde el punto de vista del desarrollador, el dominio era no solamente amplio, sino desconocido, se intento desde un principio hablar en los mismo términos, es decir, con las mismas herramientas de especificación y sobre los mismos conceptos.
El proceso fue fundado sobre la metodología de Jacobson (OOSE - 1992). El modelo de actores y el modelo de use cases fue utilizado para fijar los límites funcionales de la aplicación. Como una empresa en una secuencia de actividades, cada una tomando una fuente, mejorándola y produciendo una salida para el próximo paso del canal (Porter, 1980), toda la funcionalidad de la supuesta compañía fue expresada en términos de use cases, cuya propia definición se ajusta a la expresada en estas líneas.
Luego de este proceso, conseguimos estar de acuerdo en la funcionalidad que la aplicación debía proveer y alcanzamos una clara vista del dominio. Los use cases y sus descripciones nos condujeron a eso. Pero teníamos un problema mayor : no éramos capaces de poder discutir las políticas de mejoras o -aún más- las óptimas soluciones a los procesos de negocios que surgían de los use cases. Peor aún : ni siquiera teníamos documentados, más allá de nuestra mente, cuales eran esos procesos de negocios. De esta manera, empece a escribir los problemas y los primeros intentos de solución a cada uno de ellos bajo el estilo de escritura de un pattern. Primeramente, describiendo textualmente contexto-problema-solución, para agregar en su segundo intento, las fuerzas o influencias del problema en las posibles soluciones (context, problem, force y solution constituyen las cuatro partes con las cuales se describe un pattern). Finalmente, en la tercer iteración sobre estos patrones se agregó a la solución la descripción de la misma en forma gráfica utilizando diagramas que describan las entidades, reglas y relaciones entre los objetos. Dichos diagramas son los

mismos utilizados en Analysis Pattern - Fowler 96, aunque fueron especificados inicialmente por Martin95.
Con todo esto en mente, fuimos capaces de mejorar cada proceso de negocios (ahora un pattern) significativo para nuestra aplicación. Con los patrones de negocios cuidadosamente filtrados luego de varias iteraciones, se constituyeron en la principal base de conocimiento para construir los modelos de diseño de la aplicación. Un hecho a destacar fue que durante la construcción de los propios modelos de los patterns, mi proceso mental se veía influenciado por la experiencia modelando objetos. Esto no significa que este mal. De hecho, quizá sea una buena práctica. Pero el punto a destacar es que este no es el esquema mental de un manager, con lo cual concluyo que el proceso de definición de los patrones de negocios debería ser desarrollado en conjunto por managers y desarrolladores, considerando aquí que los managers no conocen absolutamente nada del paradigma de objetos. De cualquier manera, este esquema debería ser probado en un contexto más grande para lograr conclusiones más interesantes y concisas.

Patrones de Negocio para el Marketing y el Marketing Directo

El principal resultado de este punto es conseguir un profundo conocimiento del negocio alrededor del marketing directo. Los patrones de analisis cubren dicha característica. Posteriormente, y fundado en este resultado, los patrones de negocios descriptos aquí sirven como el sustento de una implementación acorde a las especificaciones del dominio comercial y con el grado de reusabilidad y extensionabilidad que proveen los propios patterns.
Los analysis patterns han sido descriptos usando los mismos diagramas de Fowler96. Los mismos resultan simples y más familiares para los managers debido a que básicamente representan relaciones y reglas sobre entidades del negocio.

Ventajas del approach

La principal ventaja es tener a todo el mundo hablando en el mismo lenguaje a la hora de especificar los procesos de negocio (sus reglas, entidades y relaciones) y fundamentalmente, sin sacarlos de sus contextos. Esta es la forma en que gente de sistemas y gente de gerencia colaboran para un resultado común sin pelearse mutuamente por el desconocimiento de los formalismo y técnicas de ambas ciencias.
Algunos approaches existentes para estos desarrollos colaborativos tales como Business Objects Advantages - Jacobson95, mantienen a la gente involucrada hablando en términos exclusivamente de objetos. Así, estas técnicas involucran al manager en conceptos del paradigma tales como el polimorfismo, que puede derivar en buenos modelos, pero que agrega niveles de complejidad y rechazo que los managers generalmente no están dispuestos a interpretar. Además, la gente de gerencia debe capacitarse en notaciones de objetos para poder participar activamente en los desarrollos. Esto puede generar un caos inútil, que podría evitarse con el simple hecho de no sacar a nadie de su contexto. Y este es el intento de este approach.

Intento

Principalmente la intención de construir patrones de análisis para el dominio mencionado es la de poder fundar un framework de objetos de negocios para soportar los procesos que el negocio involucra. Pero como consecuencia, se obtiene una vista del negocio con características de extensionabilidad, reusabilidad y fácil comprensión por gente nueva al proyecto. Aún más, la gente sin conocimientos del dominio, en este caso del marketing directo, puede conseguir una rápida, concisa, concreta y efectiva vista y entendimiento del negocio sin necesidad de revisar extensas bibliografías que seguramente hablen del negocio en general pero no de las particularidades de la empresa en cuestión. Por otro lado, los managers se aseguran un medio de comunicación de las reglas del negocio con menor probabilidad de que las implementación conseguidas fallen a los reales

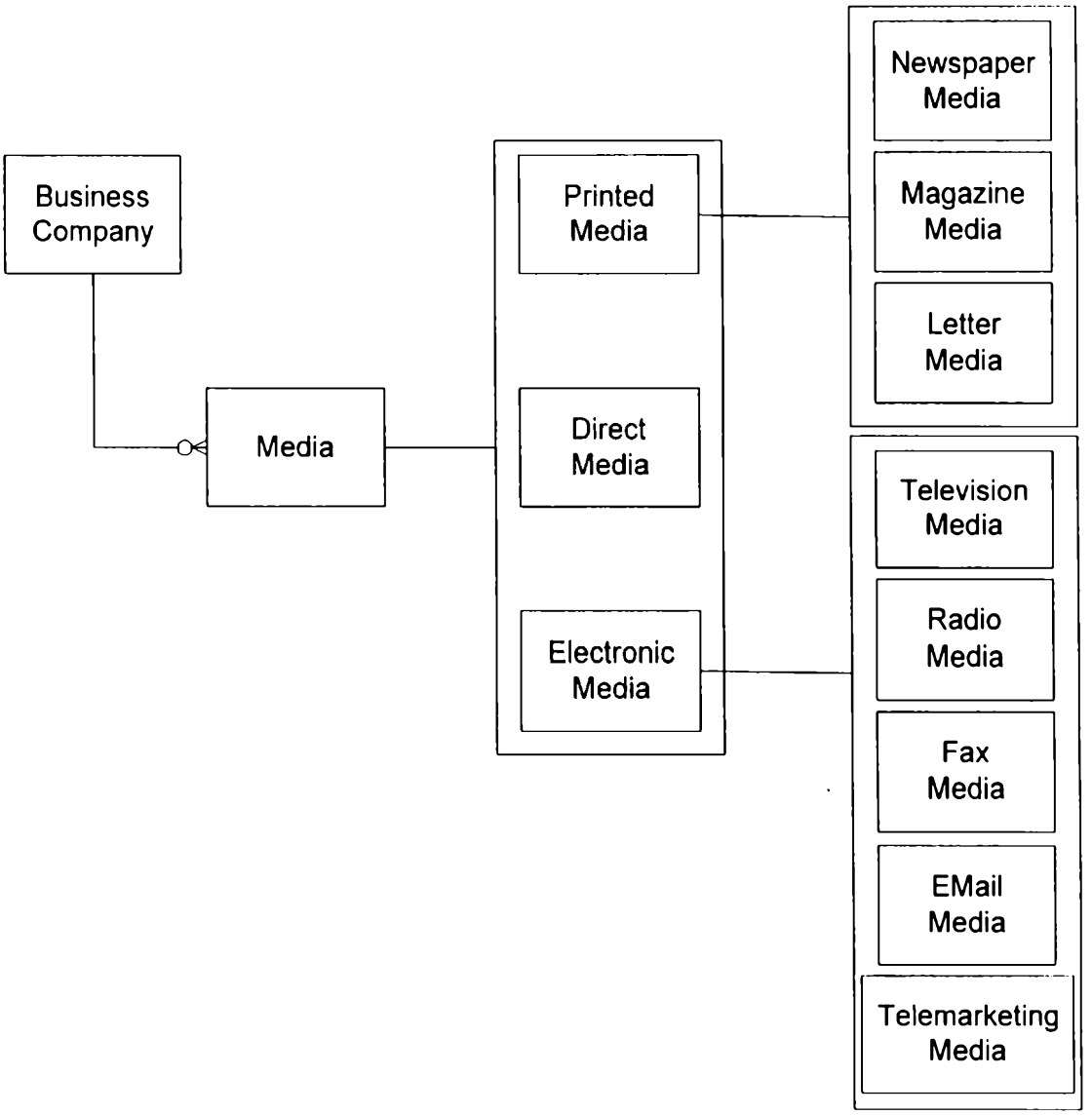
requerimientos del negocio, evitando así el radio-pasillo generado en las grandes organizaciones a la hora de transmitir conceptos del negocio desde la alta gerencia hacia los desarrolladores de software (el manager dice A, el analista de requerimientos entiende B, el analista de sistemas cree C, el diseñador modela D y el implementador desarrolla Z).

Modelo de Negocios - Patrones definidos.

Los patrones han sido descriptos en una forma evolutiva. Es decir, los patrones más sencillos del negocio han sido ubicados al principio para evitar introducir conceptos más complejos del negocio, y por el contrario, ir introduciéndolos pausadamente.
Para conseguir una simple y rápida visión de las actividades del negocio, el lector puede referirse a los apéndices para observar el modelo de use cases.

Distribution Media pattern

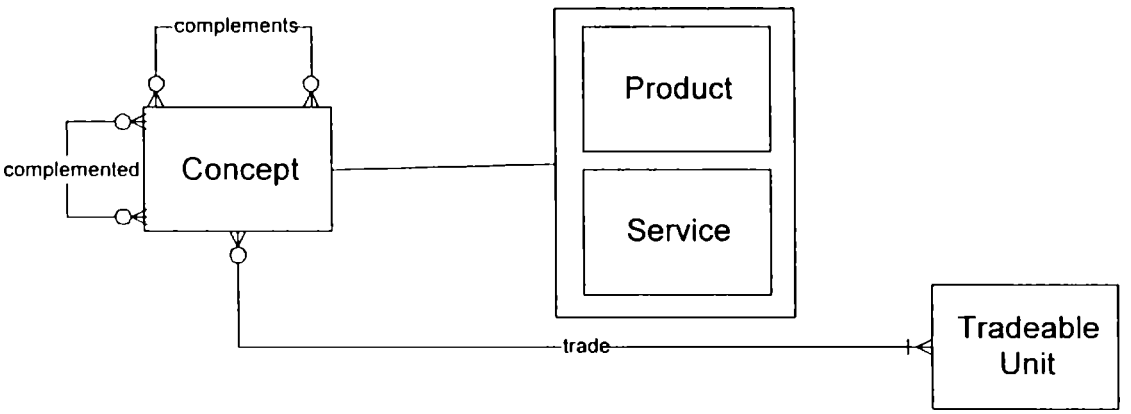
Context : El marketing directo inolucra un lugar donde las actividades son soportadas
Problem : El profundo conocimiento del lugar o el medio donde los negocios son ejecutados es muy importante para la organización. Esto significa que la cantidad de información que la compañía posee acerca del medio puede tener una significativa influencia en las decisiones financieras y comerciales que se apliquen.
Forces : Diferentes medios aparecen instanciables para el marketing directo. Normalmente, la gente se confunde acerca del significado de medio en el marketing directo. El medio no es necesariamente mailing o correo electrónico. Por ejemplo, un supermercado puede intentar hacer marketing directo en su propio local, intentando establecer así una robusta relación con sus clientes. Por esa razón es importante que la compañía sepa los diferentes medios que puede utilizar para el marketing directo y la diferente información que debe manipularse sobre cada uno.
Solution : Identificar todos los tipos de medios posibles para el marketing directo y la información sujeta en cada uno. Estos medios serán útiles para instanciarlos en nuevas estrategias y polícias de la compañía, utilizándolos como los canales de comunicación entre lo que la compañía ofrece y el propio cliente. Dicha solución debería ser como lo indica el siguiente diagrama:



Concept pattern

Context : Un complejo, duro y competente mercado.
Problem : Una compañía puede desarrollar una gran cantidad de cosas para ofrecer al mercado, pero solo algunos de ellos lograrán ser exitosos. Solo aquellos que son puestos en ele marcado en la forma correcta, en el momento exacto y con el precio justo. Entonces, ¿cuáles son las políticas a utilizar para tener una mejor idea de lo que la compañía esta tratando de poner en el mercado? ¿Qué constituye eso que la compañía insertará en el mercado ? ¿Todo es igual ? ¿un servicio o un producto ?
Forces : Primeramente, las empresas no deben vender simples cosas: ellos deben vender conceptos. Como un todo. Un concepto envuelve una de dos cosas que una empresa puede ofrecer a un mercado : un producto o un servicio. Y esto es lo que la compañía ofrece

al mercado, fundada en el hecho de que para el mercado ambas cosas son lo mismo. Por el contrario, si la compañía discrimina ambas cosas, puede dejar de prestar atención, por ejemplo, a los productos, consiguiendo así desinterés por parte de sus consumidores.
Con esto en mente, cabe destacar que un concepto tiene una línea de evolución a través del tiempo. El concepto nace envolviendo un concepto o un servicio y evoluciona mejorándose a si mismo con nuevas versiones. El mejoramiento puede ser a causa de un nueva característica del concepto o bien debido a necesidades del mercado que imponen dicho cambio.
Un producto o un servicio no tiene porque ser una unidad. Por ejemplo, en el contexto de un supermercado, un concepto puede ser 100 latas de tomate y no necesariamente solo una de ellas.
Solution : Describir la información que un concepto debe sostener. Un concepto puede ser el primero de su generación, o por el contrario, tener un ancestro. Consecuentemente un concepto podría llevar consigo ventas enganchadas. Esto implica que cuando una empresa vende un concepto, el mismo podría acarrear consigo ventas adicionales en el mismo momento o posteriormente. De esta manera, los conceptos tienen complementos, que no son otra cosa que otros conceptos. La solución puede lucir de la siguiente manera:



Evolution pattern

Context : Los mercados son extremadamente dinámicos.
Problem : ¿Cómo es posible mejorar los conceptos de una compañía en forma eficiente y segura? ¿Cómo es posible aprender y mejorar desde los errores cometidos en el pasado?
Forces : El mercado cambia en forma dinámica y variada. Y las empresas deben moverse en dirección del propio mercado. Conocer que, cuando, como y a quién ha sido vendido un concepto en el pasado es extremadamente útil para las futuras decisiones comerciales. Por esa razón, las empresas no deberían descuidar la información de como los conceptos que ellos ofrecen y han ofrecido al mercado han ido evolucionando a través del tiempo.
Sobre estas bases, la compañía puede también identificar que conceptos pueden llevar consigo ventas enganchadas. Por ejemplo, la muñeca "Barby" es un concepto que vende la empresa XYZ. Otro concepto diferente puede ser la ropa que usa la muñeca, concepto el

cual es comercializado también por la misma empresa. Sin embargo, vender una muñeca puede tentar al cliente a comprar un set de ropa para ir vistiendo a la muñeca. De esta manera, la ropa pasa a ser un complemento de la muñeca. Las empresas deben conocer e identificar este tipo de situaciones para instar a sus empleados o a su fuerza de venta a aplicar la psicología para hacer efecto de estas ventas enganchadas.

Solution : Varios aspectos deben ser tenidos en cuenta:

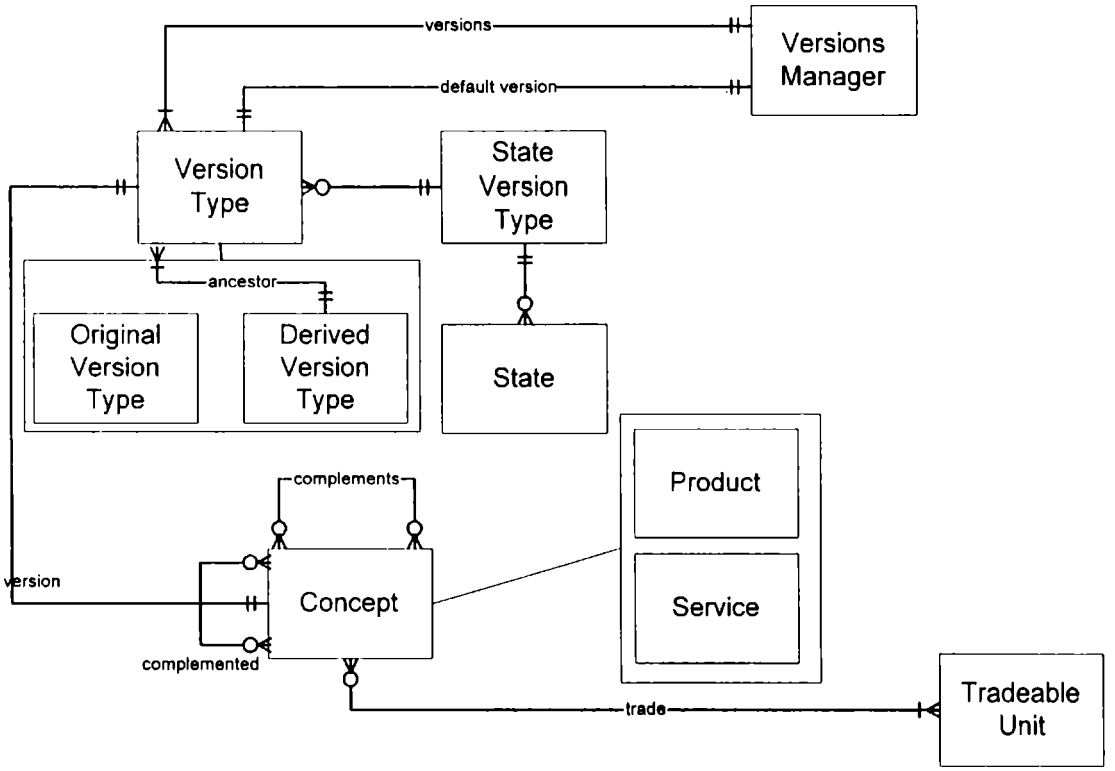
1. Para conocer como un concepto ha evolucionado, su propia línea de evolución debe ser mantenida a través del tiempo. Cada nodo es una versión del concepto en se instante de tiempo.

   Un concepto nace sin ancestros. Luego, dicho concepto es cambiado o mejorado, entonces aparece un nuevo concepto que refleja dichos cambios y que tiene como ancestro al primeramente mencionado. Sin embargo, esto no implica que el primer concepto creado y posteriormente evolucionado, deba desaparecer del mercado. Por el contraio, ambos pueden convivir juntos y tener márgenes de rentabilidad similares. Bajo ningún punto de vista, un concepto (en realidad una versión de un concepto) debe desaparecer del sistema de información de la empresa debido a que el mismo puede resultar útil para futuras decisiones comerciales o bien para ser reinsertado al mercado en el futuro.
2. Los complementos de los conceptos también evolucionan, porque ellos también son conceptos. Esto implica que si un complemento evoluciona, todo concepto que lo tiene como un complemento debe evolucionar también a una nueva versión. Con esto en mente, algunas reglas deberían ser aplicadas:

   .- Si un concepto a versionar tiene complementos entonces la nueva version debe tener los mismos complementos.

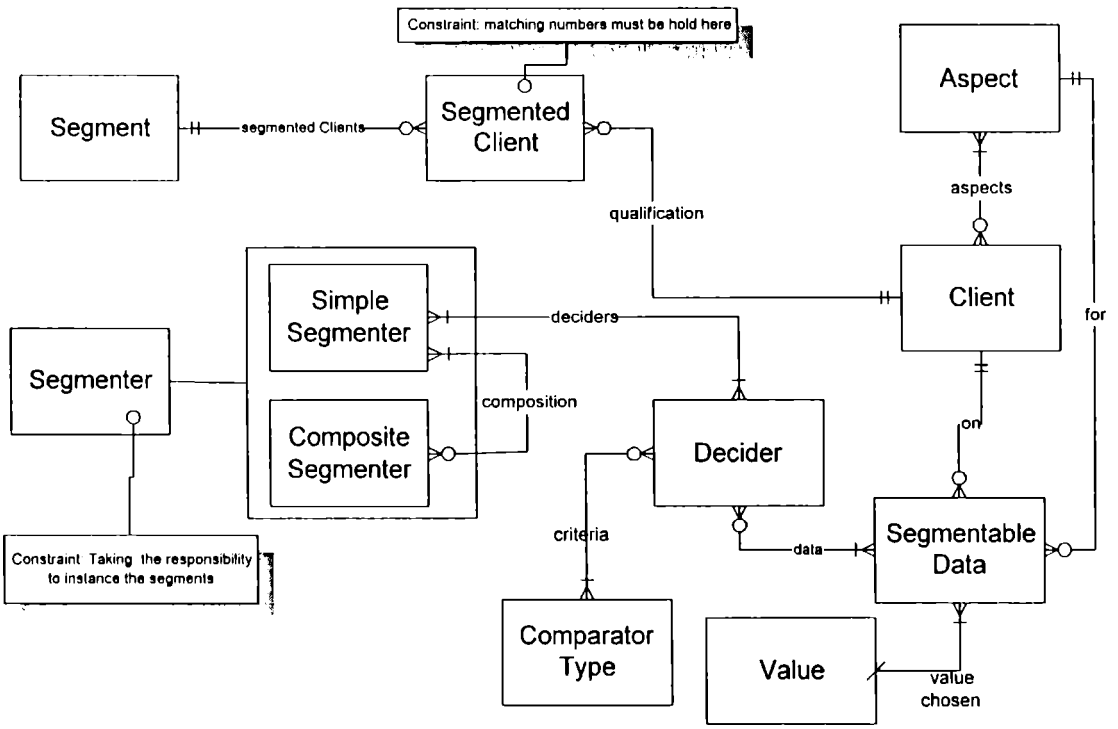
   .- Si un concepto a versionar complementa otros conceptos, nuevas versiones de estos ultimos deben ser creadas también. Este último punto es recursivamente aplicado. .

El diagrama que refleja lo anterior es el siguiente:



Segment pattern

Context : Las acciones de management necesitan ser aplicadas sobre conjuntos de clientes agrupados bajo ciertas características.
Problem : ¿Cómo pueden agruparse los clientes bajo características conocidas y comunes de los mismos? ¿Cómo no perder eficiencia y efectividad en tal proceso?
Forces : Todo cliente conocido dentro de la compañía debe poder ser filtrado y agrupado por aspectos que lo caractericen de manera tal de poder tomar decisiones comerciales con dichos grupos. Como tales grupos tienen características comunes, los managers pueden, por ejemplo, disparar una serie de ofertas que apunten a ese tipo de consumidor. Los segmentos pueden ser combinados entre si para lograr nuevos segmentos, y esto no es otra cosa que una intersección de clientes de ambos segmentos. La principal ventaja de segmentar es la posibilidad de dirigir una decisión comercial (como una oferta) a un conjunto selecto y estudiado de clientes potenciales, incrementando la posibilidad de éxito del plan de marketing.
Solution : Crear y mantener segmentos de clientes de la siguiente manera:



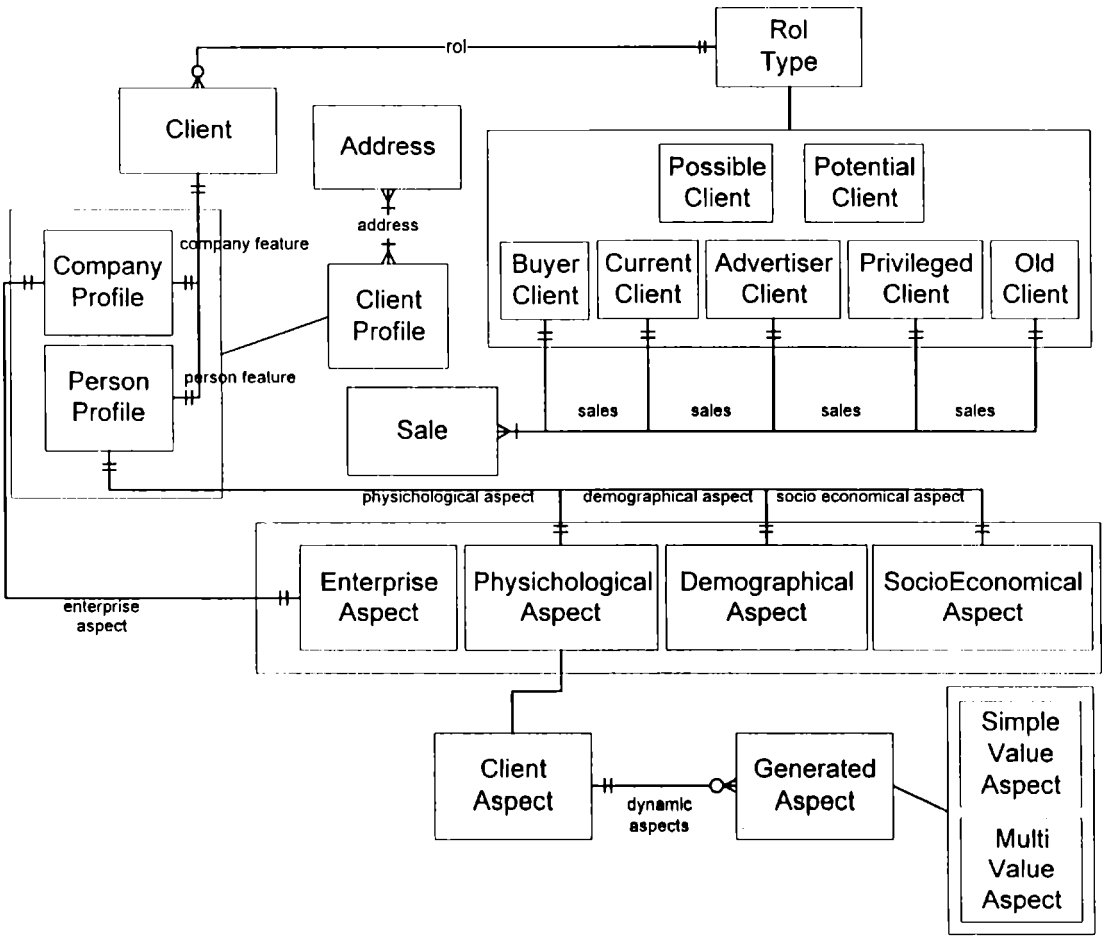
Client Knowledge pattern

Context : El más importante de los conceptos del marketing directo es el conocimiento de los clientes.

Problem : ¿Cómo poder conocer todo lo que queramos sobre los clientes y poder utilizar esa información para cualquier actividad del marketing directo?

Forces : Los clientes de una empresa deben ser conocidos tanto como sea posible... y aún más. Otra vez, este variado y complejo conocimiento de cada cliente brinda a la empresa la posibilidad de ajustar sus decisiones con más precisión, de manera tal de poder ofrecer nuevos conceptos a los clientes indicados, recuperar viajas relaciones comerciales, ganar la onfianza de los clientes, lograr que se conviertan en publicistas gratuitos de los conceptos de la empresa fruto de la identificación de los clientes con la compañía, etc. Es prácticamente imposible poder lograr genericidad en este aspecto. Esto es, resulta muy difícil lograr identificar toda la información que debe conocerse acerca de los clientes en forma general para cualquier empresa. Además, aunque lo logremos, en el futuro es altamente probable que al menos una de las compañías requiera aún más conocimiento. Esto indica que el conocimiento acerca de los clientes no solo es variado sino dinámico. Los aspectos considerados suficientes hoy, serán seguramente escasos para mañana. Sin embargo, con un profundo análisis puede lograrse conocer globalmente cuales son las características que deben asentarse acerca de los clientes. Igual, no será suficiente.

Solution : Definir un profundo conocimiento acerca de los clientes y dejarlo abierto al crecimiento dinámico de la información asociada a cada cliente sin que esto afecte la solución encontrada. El siguiente diagrama resume el pattern:



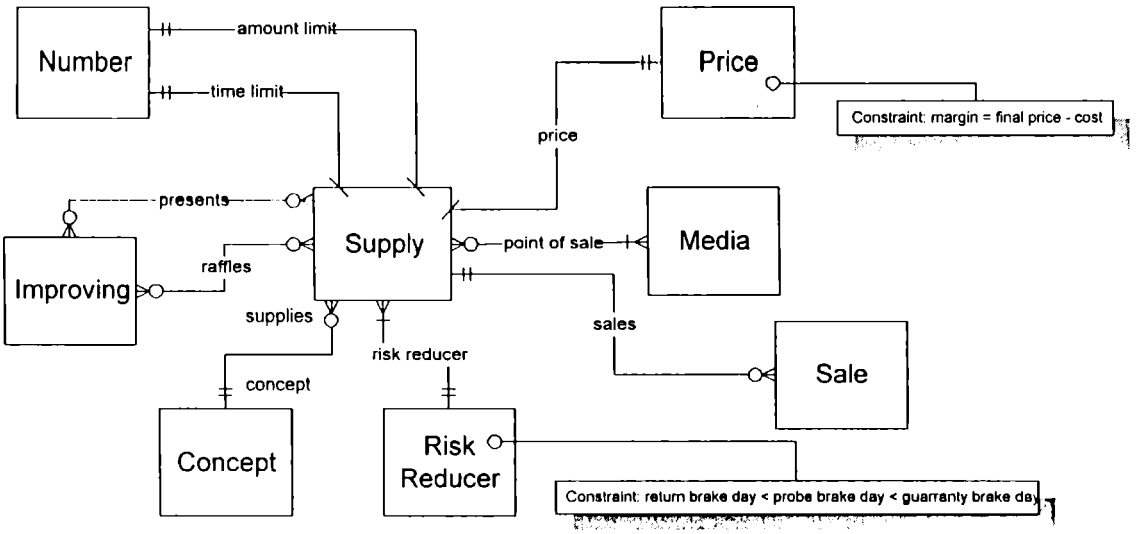
Supply pattern

Context : Las empresas sobreviven gracias a las ventas.

Problem : Para crecer, las empresas necesitan incrementar sus ventas constantemente. De esta manera es necesario estudiar que conceptos realmente necesita o consume el mercado, para luego lanzarlos a la venta.

Forces : Un error simple que las empresas cometen es que ellas lanzan al mercado conceptos en vez de ofertas. Estudiando las preferencias y las necesidades actuales del mercado, los managers deben ser capaz de identificar la mejor forma de ofrecer un concepto al mercado. En cualquier caso, un concepto debe ser envuelto en una oferta. Una oferta es una forma dinámica de observar un concepto. Pueden crearse, en diferentes mercados o tiempos, diferentes ofertas para exactamente el mismo concepto, diferenciandose por el precio, el punto de venta, el posicionamiento, etc.

Solution : El siguiente diagrama resume los que una oferta debería abarcar:



Parameter pattern

Context : Los clientes deben ser matemáticamente evaluados

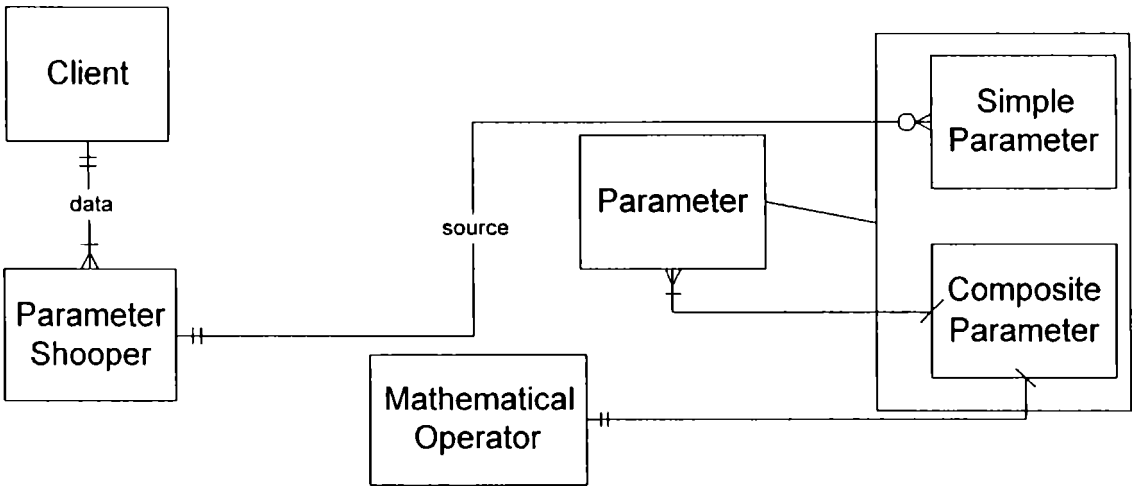
Problem : ¿Cómo se puede calificar a los clientes con un valor numérico significativo para poder compararlos entre ellos ? ¿Cuántas formas numéricas de evaluar clientes pueden existir ?

Forces : El problema es un poco más complejo aún. Las empresas deben poder calificar a los clientes dinámicamente según qué y cómo se los conozca y en formas combinadas.

Solution : Para proveer esto los managers deberían definir parámetros. Un parámetro es una función matemática aplicable sobre el cliente que retorna un valor numérico. También, ciertos parámetros pueden representar una constante que siempre retorne el mismo valor para todos los clientes.

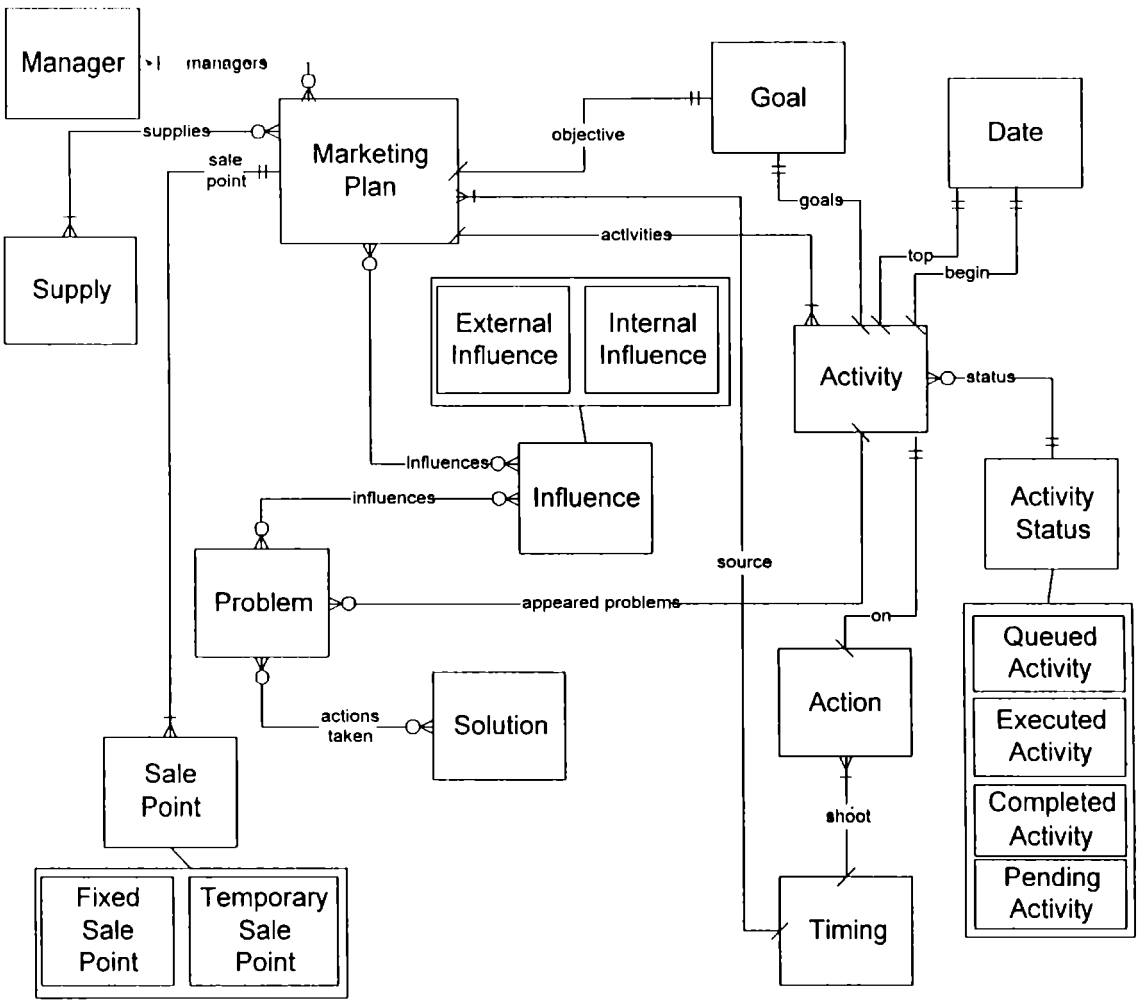
Con lo anterior, los managers pueden combinar parámetros existentes para obtener nuevos parametros. Dicha combinación se hace a partir de un operador binario que actúa de nexo entre amos (*, +, -, etc.).

Notar que el centro de información aquí es el cliente. De esta manera, toda información útil para construir parámetros debería ser alcanzada desde el propio cliente (compras, medios usuales, etc.). En consecuencia, los aspectos parametrizables de los clientes deberían ser conseguidos por un proveedor especifico y dinámico (shooper), tal cual lo indica el diagrama :



Plan pattern

Context : Todo proceso de marketing debe ser conducido desde un esquema claro, estructurado y preciso llamado plan de marketing.
Problem : ¿Que involucra un plan de marketing? ¿Cómo debe ser conducido y estructurado? ¿Cuál es la información que debe manejarse?
Forces : Primero, todo nuevo negocio debe ser conducido y estructurado bajo un plan de marketing que asegure el orden en las actividades involucradas, el momento de ejecutar las acciones y el seguimiento de los problemas y soluciones aparecidas en su ciclo de vida. Teniendo un claro mapa de actividades a realizar para cada plan, el éxito del mismo puede ser probabilísticamente más alto.
Además, un plan es creado para asentar ofertas en el mercado que lucren en la busqueda del objetivo que los managers asignaron al plan en cuestión.
Solution : Un plan debe definir actividades. Cada una debe ser completada en un periodo de tiempo, con fechas de inicio y fin claramente identificadas. Un proceso de timming puede dispararse bajo esas premisas. El modelo luce de la siguiente manera:



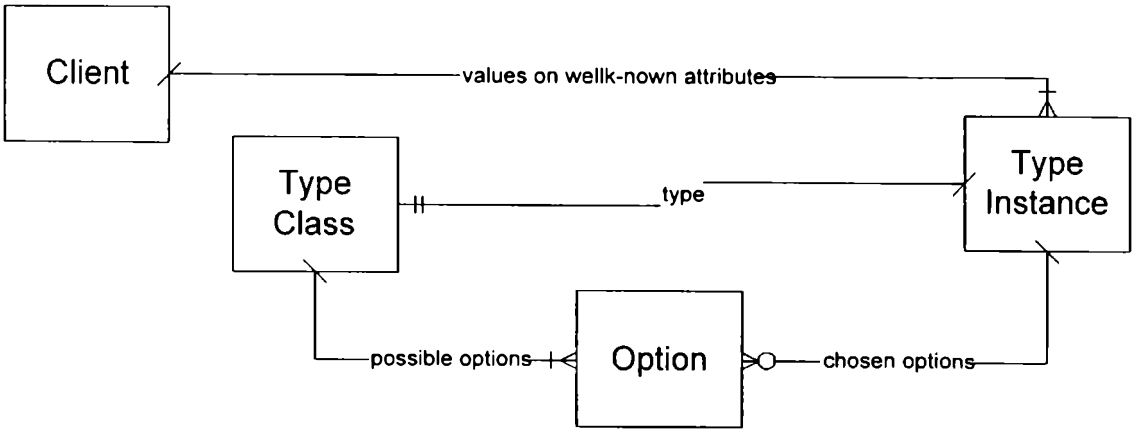
Typed Information pattern

Context : La información que se conoce de cualquier aspecto del mercado debe ser precisa. Redundancias y ambiguedades del lenguaje afectan las actividades del marketing directo.

Problem : Suponga que el manager debe filtrar los clientes por un aspecto tal como sexo. El resultado debería ser preciso porque el manager tomará decisiones significativas en base a la cantidad de clientes que consigue. Imagine que unos meses antes un viejo empleado conservaba información de los clientes asentando el sexo como "Hombre" o "Mujer". Pero desde hace un tiempo, un nuevo empleado a cargo de dicho proceso salva la información de los clientes asentando en el sexo de los mismos "Masculino" o "Femenino". El manager no sabe nada al respecto y cuando filtra sus clientes lo hace por "Hombre". Obviamente, una importante cantidad de información quedará al margen de su segmentación debido a la falta de completitud de sus datos.

Forces : Carpetas de tipos predefinidos deberían ser utilizados para evitar el problema descripto. De esta manera, no existirán valores ambiguos para cada información tipada.

Solution : Identificar cada atributo que tiene valores fijos y aplicar la solución abajo descripta. Procesos como el de segmentación se verá ampliamente favorecido por este pattern.



ANALISIS DE APLICABILIDAD DE OOSE - Variantes

FIVE STEP JACOBSON APPROACH

En el estudio y práctica usando OOSE, es muy probable que los analistas y diseñadores - aún contando con una herramienta CASE- no apliquen formalmente toda la metodología paso a paso.
Dependiendo de la magnitud del desarrollo, los cinco modelos (Requerimientos, Análisis, Diseño, Implementación y Testeo) serán más o menos formales.

El primero resulta fundamental para el entendimiento del dominio, con lo cual se respeta la construcción del Modelo de Use Cases tal como lo especifica Jacobson. Algo similar ocurre con las Interfaces de Usuarios que generan el refinamiento y claridad de los requisitos del cliente. Por otro lado, el Modelo de Objetos del Dominio no aporta demasiada utilidad, ya que en general los objetos encontrados terminan en mutaciones, desapariciones, resultan insulsos y fundamentalmente nos pueden inducir soluciones que no son las mejores, esto es, mentalmente nos regimos por un objeto que inicialmente hemos visto en el dominio y tendemos a modelarlo a la fuerza durante el análisis. En otras palabras, tendemos a no descartar los objetos que vimos por primera vez.

Por otro lado, parte de los modelos de Análisis y Diseño se terminan fusionando. Esto se debe a que generalmente se tiene totalmente identificada la plataforma de desarrollo. La única diferencia que podría considerarse entre ambos procesos en esta fusión radica en el hecho de que en Análisis se establecen los objetos y sus conocimientos y colaboraciones en términos de cada use case y en Diseño el cómo colaboran y qué responsabilidades tienen a través de los Diagramas de Interacción en el flujo indicado por cada use case. Sin embargo, una fusión alternativa usada frecuentemente por el autor de esta tesis es detallada más abajo.

En el Modelo de Implementación su última fase consiste tan solo en traducir el diseño a pseudocódigo o código. Este parte del proceso puede evitarse sin perder formalidad, indicando en los propios diagramas de interacción la secuencia de código para la resolución del use case. Aunque el objetivo de este proceso es, usando la notación Wirfs-Brooks como sugiere Jacobson, dejar establecido cuales son las clases, su representación, sus responsabilidades y sus colaboraciones. Lo que habitualmente se hace es el mapa de objetos que conforman el modelo todo, usando la notación del Design Patterns.

El Modelo de Testing se aplica directamente sobre la implementación, aprovechando el aporte del paradigma de juntar funciones y datos, y sujetos a lo que se estableció en cada use cases original comparado con lo que retorna la aplicación final ante el disparo de cada uno de ellos.

En definitiva, una aplicación reducida - pero a su vez productiva - de la metodología, podría resumirse en los siguientes cinco pasos :

1. ***Construcción del Modelo de Use Cases a partir de actores***
   Esto incluye la redacción de los use cases y el chequeo e interacción con el/los cliente/s y/o los conocedores del dominio.
2. ***Construcción del Modelo de Análisis para cada use cases desde su propia redacción***
3. ***Revisión del Modelo de Análisis***
   Esto implica un refinamiento general, unificación de objetos diferentes que resultan ser los mismos, aplicación de patterns de diseño, etc.
4. ***Construcción de un Diagrama de Interacción para cada modelo de análisis de cada use cases***
   Esto puede evitarse para use cases muy simples. También, la línea de secuencias de los DI puede incluir tanto código, como pseudocódigo como el mismo texto de los use cases.
5. ***Construcción de un mapa final de los objetos del sistema usando la notación del Design Patterns***

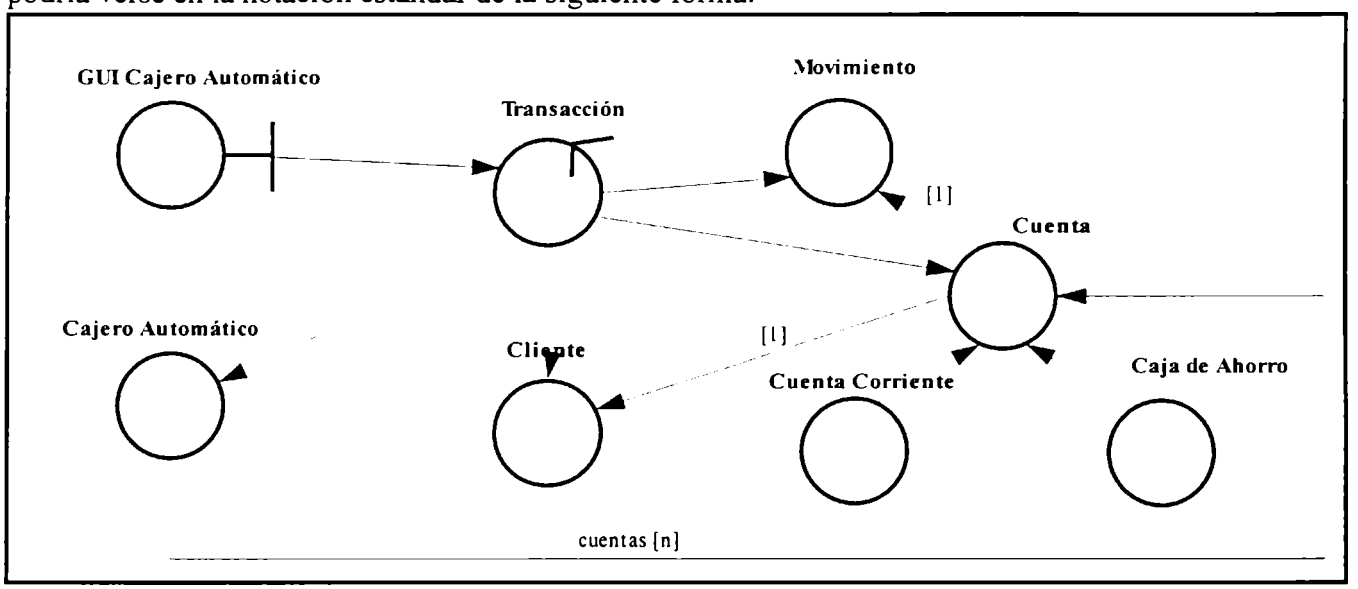
VARIANTE REDUCIDA

Una variante aplicada por el autor de esta tesis consiste en la idea de poder tener todo lo modelado sobre un use case en una sola view. Y esto se logra eliminando los diagramas de interacción. Pero para tal causa los modelos de análisis de los use cases deben ser lo suficientemente precisos como para soportar lo que nos brindan los DI.
Esta necesidad surgió al hacer reingeniería de modelos de análisis y fundamentalmente al tratar de entender la lógica modelada en el proceso unificado de Analisis-Diseño a la Jacobson. Este último representa en sus modelos de análisis colaboraciones o conocimientos sin diferenciar para qué ni cómo, tan solo usando flechas. Esto provoca que quien este estudiando el modelo de objetos del use cases deba recurrir permanentemente a los diagramas de interacción (si es que existen).
Ahora bien, como los DI indican mensajes que se disparan entre objetos a través de la secuencia de tiempo, esta secuencia podría expresarse directamente sobre el modelo de análisis. Entonces, la variante notacional sobre los modelos de análisis-diseño establece :

- Se respeta la notación de objetos de interface, entidad y control.
- Se respeta la notación de conocimiento o parte_de con flechas indicando aridad en el primer caso y nombre más aridad en el segundo.
- Las flechas simples (sin nombre y dirigidas) que indican colaboración de un objeto a otro en la metodología standard ahora pasan a ser flechas con labels. Esto es, como la flecha indica una colaboración de otro objeto es adecuado indica cual colaboración se requiere, es decir, cual es la responsabilidad del objeto colaborante. En otras palabras, el propio método.

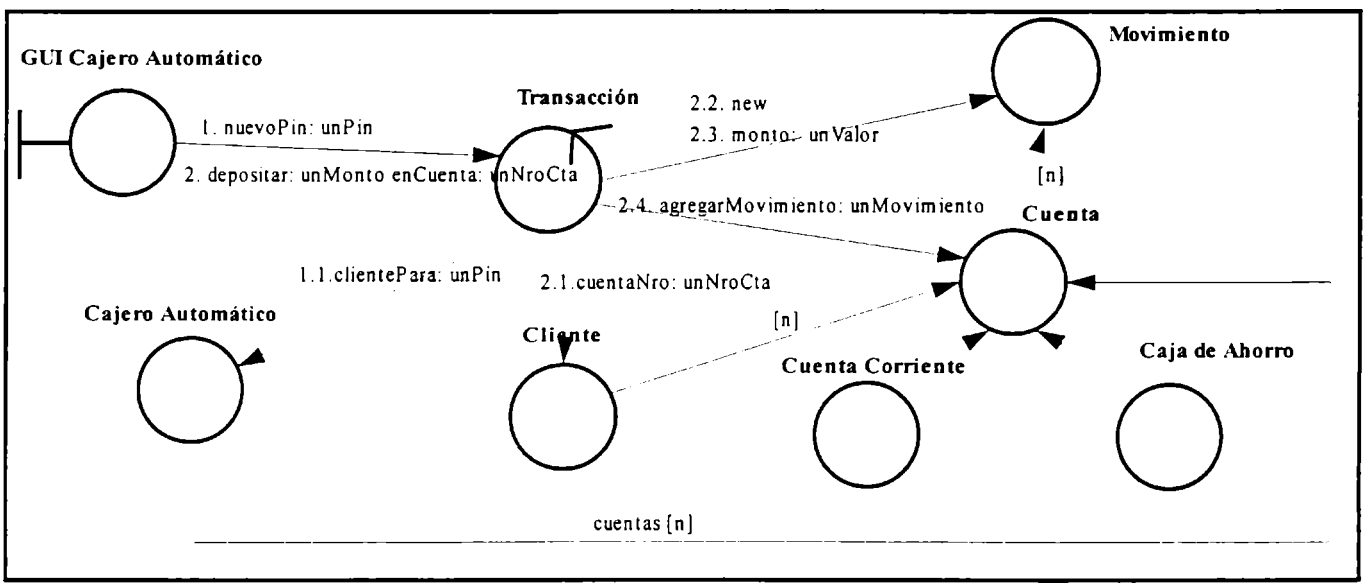
- Un objeto aparece una sola vez dentro del view del modelo. A diferencia con la estándar, donde puede aparecer varias veces. Esto puede burlarse si el modelo es complejo colaboracionalmente hablando.
- Las labels de las colaboraciones son antepuestas por un número. Dicho número no es más que un índice que indica el orden en que se dispara el método. Y en el caso que un método etiquetado i requiere para su desarrollo el disparo de otros n métodos, se realiza por cada uno de los métodos etiquetados con subíndice i.1, i.2, ... , i.n.

Así, por ejemplo, un use case que tratase el deposito de efectivo en un cajero automático podría verse en la notación estándar de la siguiente forma:



Queda claro que poco dicen las flechas colaboracionales entre objetos, aún teniendo en cuenta la simplicidad del ejemplo y la descripción textual del use cases que sustenta el modelo.

En tanto, en la versión alternativa, la situación quedaría de la siguiente forma:



CONCLUSION

En la experiencia y opinión personal utilizando OOSE hay solo un punto desfavorable a la hora de conducirse con la metodología. Y este punto viene dado por el hecho de que, si no se es cuidadoso, cada use cases tiende a vislumbrar un objeto de control. Pero lo peor del caso es que estos objetos tienden a ser objetos dioses, es decir, que toman el control de toda la ejecución del caso de uso evitando el uso de delegación. Es muy recomendable, luego de modelar todos los use cases, intentar hacer un mapa global de objetos y maximizar la aplicación de patterns sobre los mismos.

Para el desarrollo de la tesis se utilizará la notación establecida en la variante reducida. De cualquier manera, cuando los modelos de análisis de los use cases son extremadamente complejos, esta notación se torna engorrosa (a no ser que se utilice más de una representación gráfica para cada invocación a los objetos con muchos accesos). Por esa causa, los diagramas de interacción justifican su reaparición en escena y serán usados por conveniencia.

Una Política de Testing del Producto, Inserción y Seguimiento en el Mercado

Testing

La robustez y la carencia de errores de la aplicación no basta con un fuerte testing de componentes y de funcionalidad por parte del desarrollador. Se necesita un testing mucho más estructurado y concreto para garantizar una mas fiable inserción en el mercado, donde los errores, tanto de código como de interpretación, pueden hacer desequilibrar la propia inserción y la consecuente estabilidad del producto en el mercado.
Una política de testing tiene como objetivo identificar los tres tipos de fallos posibles, localizados en : modelos de análisis (inapropiadas consideraciones), modelos de diseños o clases (diseños erróneos) y el propio código (errores o código incorrecto).

Antes de empezar cualquier testing y considerando que todo el desarrollo se condujo por use cases, podría considerarse hacer un chequeo inicial de funcionalidad. El más simple de todos. El punto es que luego de hacer tanto diseño y tanta -fundamentalmente- implementación, se pierde un poco de vista la inicial tarea de especificar los límites y las necesidades funcionales de la aplicación. A tales efectos aparecieron inicialmente los use cases. Pero ahora un chequeo a su descripción textual resulta una buena práctica para detectar si hay incongruencias significativas y funcionalidades descartadas o modificadas ortogonalmente que harían fracasar por completo el proceso de testing al no existir matching entre lo que los use cases especifican y lo que el sistema retorna ante cada interacción con los actores.

Un testing **adecuado** no es aquel que prueba todos los casos posibles, lo cual resultaría exhaustivo y poco práctico, sino el que cubre la cantidad precisa de casos como para demostrar el suficiente grado de robustez en la aplicación final.
Aprovechando que el proceso de desarrollo de la aplicación ha sido *use case driven approach*, contamos con la ventaja de tener todo un modelo de use cases que indican que es lo que debe hacer la aplicación y como debe llevar a cabo tales acciones. Y esto resulta la base para establecer los casos de testing para la aplicación. De esta manera el tipo de test que se realiza se dispara de acuerdo a la funcionalidad esperada en vez de basarse en las interacciones entre objetos. Este último tipo de test focaliza su interés en la interacciones entre los métodos a partir de un servicio en particular desde una clase en particular. Es decir, en este último caso el testing se realiza por cada clase y posicionándose en cada método o servicio público que brinda la clase. En primero de los casos se considera test horizontal (sobre la funcionalidad), mientras que el segúndo es vertical (sobre la clase).

¿Cómo construir los test cases ?

El flujo de eventos que conforma un use cases está lleno de entradas de datos. Cada una de esas entradas de datos puede contener un rango determinado de valores. De esta manera :

- Establecer textualmente el test case en dos fases :
    - Estímulo : Describir el propio use cases instanciando una combinación posible de valores para cada una de las entradas de datos,
    - Resultado Esperado : Describir el estado final luego de la ejecución del test cases.

De lo anterior se desprende que por cada use case se obtiene al menos un test case.
Notar que no es necesario tomar **todas** las combinaciones posible de valores para la entradas, sino aquellas que el tester considere significativas (aparece factor humano inevitable).
De la misma manera, no todos los use cases necesitan ser considerados. Para lograr clasificar los use cases acorde a su importancia en el proceso de testing se puede realizar una tabla con dos columnas : *frecuencia de ejecución* y *factor crítico* que indiquen, por cada use cases, un valor del 1 al 10. Con este simple esquema se identifican cuales son los use cases vitales para nuestro proceso de testing.

Ahora... ¿quién escribe los test cases ?. En principio, el propio developer/s podría encargarse de tal tarea, necesitando luego una considerable interacción con el o los managere para controlar que los valores y casos considerados se asemejan a la realidad del dominio. Si el manager resulta ser un experto y ha tenido una muy fuerte participación y colaboración en los modelos de requerimiento, análisis y diseño, el mismo podría encargarse de la elaboración de los test cases sin colaboración de los developers.

Algunos puntos a considerar en la elaboración de test cases :

- Reducir el número de test cases tanto como sea posible.
- En una batería de testing es posible generar automáticamente test data, pero no test cases debido a que su fuente radica en la ambigüedad textual de los propios use cases.
- Formalmente, la estructura de cada test cases debería considerar algo como : <estado inicial, eventos, estado final>

Testing focalizado en el Desarrollador

El desarrollador debería trabajar permanentemente a conciencia. Esto es, tener constantemente en su mente las consecuencias futuras de no hacer un buen testeo del código que está a punto de considerar finalizado, ubicándose en el rol del usuario final que acaba de adquirir o está a punto de adquirir la aplicación.
En este punto el desarrollador puede considerar o no los test cases preparados (si que existen a esta altura del proceso). De cualquier manera, algunos puntos a considerar a la hora de probar cada nueva funcionalidad que se va incorporando a la aplicación (siempre considerando que el desarrollo es *use case driven approach*) :

- Realizar el testing funcional de la nueva funcionalidad acorde a la descripción textual del use case que representa dicho flujo en el sistema (utilizar los test cases si ya existen). Si el use case es *abstracto*, repetir el testing para cada use case *concreto* que lo referencie.
- Probar casos extremos que involucren datos o hechos poco usuales (si existen test cases, el developer no debe preocuparse al respecto).
- Intentar en todo momento ser realista. Esto significa que el testing debería realizarse como un proceso establecido sobre datos concretos y no sobre información insulsa ingresada por el propio desarrollador. De esta manera, no solo se testea que el código no tiene falencias sino que cumple las espectativas funcionales que especifica el use case que se esta verificando.
- Si se están construyendo ventanas inteligentes (aquellas que detectan todo tipo de errores en la edición : falta de datos, falta de correspondencia, etc.), testear todas las combinaciones posibles de datos donde 1 o más de ellos permanezcan nulos.
- Identificar las ventanas que comparten información o bien que contienen datos relacionados o dependencias entre si y realizar el testeo detallado en el primer punto con ambas ventanas abiertas.

Y fundamentalmente no considerar el trabajo terminado en este punto. El testing recién comienza. Los grandes cambios pueden llegar a sucederse en la próximas iteraciones junto con los manager y más aún, con los usuario finales.

Testing focalizado en Manager Funcional del Proyecto

Como el desarrollo fue conducido por use cases, tenemos en su descripción textual nuevamente, la base para fundar el testing con nuestro manager/s. La misma puede considerarse suficiente, o bien aplicar la política descripta más arriba para calificar los use cases y obtener los test cases.

Las incongruencias no deberían ser materia común en este punto ya que las descripciones de los use cases tanto como el business model de la aplicación fueron desarrollados junto a él o los managers, de manera tal que incompatibilidades funcionales indicarían un mal trabajo a la hora en que se instanció el proceso de análisis.

Algunas consideraciones al disparar el testing sería :

- En este punto, el testing debería conducirse solo por los requisitos del propio manager. De ninguna manera el desarrollador debería intervenir sugiriendo o aconsejando. Se supone también en este punto, que el manager ya conoce la funcionalidad de la aplicación y no solo porque intervino en el proceso de análisis sino también porque ya tuvo acceso a la primer versión del manual de usuario que el propio desarrollador se encargo de construir.
- El desarrollador debería apuntar todo cuanto descubra que el manager malinterpreta o no considera. Eso puede indicar : falta de interés en la funcionalidad, nombres mal asignados o con interpretación errónea o bien no haberse incluido el detalle en el manual de usuario.
- El desarrollador debería adaptar cada parte de la aplicación que resulte incómoda al manager, aunque él mismo este convencido que la actual manera es la correcta. Pensar

que el punto de vista del manager se acerca mucho más a la realidad que quiere y necesita el usuario final que el que tiene el propio developer de la aplicación.

Inserción del Producto en el Mercado

Con la aprobación funcional de los managers, el producto está listo para salir al mercado. Pero aquí se enfrentan dos corrientes claramente diferenciadas : ¿el producto debe lanzarse directamente como aplicación final ? o bien ¿el producto debe lanzarse como un prototipo inicial ?
Por más que la aplicación haya sido realizada bajo los fundamentos más genéricos y fundada en un profundo estudio del dominio y sus necesidades, las aplicaciones genéricas terminan siendo (en su gran mayoría) un rotundo fracaso. Así, el primer caso cuenta con la simplicidad de dar por concluido el trabajo y de esperar tan solo la buena performance de la fuerza de ventas para ubicar el producto en los mercados. Simplista pero peligroso.
Por otro lado, la segunda alternativa, parece ser la más larga de recorrer, pero asegura un menor riesgo y por ende, una cota mayor en la probabilidad de éxito del producto.

DIMBO apunta a un segmento de mercado muy amplio pero claramente identificado : comerciantes y pequeñas empresas. Al ser amplio, la probabilidad de encontrar clientes interesados se incrementa. Por otro lado, y por la misma amplitud del segmento, la intersección de necesidades de los individuos dentro del mercado tiende a ser un conjunto no demasiado grande.
En consecuencia, DIMBO intenta satisfacer a todos y puede que no satisfaga a nadie. Pero... ¿dónde está la respuesta ? : En el propio mercado. Por esa razón, la segunda política se solidifica.
Ahora solo falta establecer como podría encaminarse dicha política, es decir bajo que estrategias.

Estrategias

DIMBO podría tomarse unos doce meses para hacerse conocer en el mercado al que apunta. Esto implica que versiones trials o demos de la aplicación podrían distribuirse gratuitamente a los efectos de promocionar la aplicación. proveyendo en el primer caso funcionalidad temporal con fecha de expiración.
Cada demo debería estar acompañada del propio manual del usuario y, fundamentalmente, de la información necesaria para que el usuario potencial tenga en claro que la herramienta puede adaptarse a sus necesidades ya sea total o parcialmente. De esta manera, no solo logramos un rico feedback de nuevas o diferentes necesidades del mercado, sino que podemos asegurarnos un nuevo cliente para nuestra aplicación.
DIMBO debe posicionarse como un prototipo. Las necesidades de los usuarios testers pueden determinar luego que DIMBO automáticamente pase a ser una aplicación final en vez de una demo. Al menos para algunos usuarios. Pero ese posicionamiento es vital para el

ciclo de vida que intenta enfrentar DIMBO. Así. los usuarios finales sabrán en todo momento que están tratando no con el producto final en el cual ellos van a invertir su dinero, sino en un modelo experimental que puede ser mejorado y extendido a su propio antojo, si es que así lo desea. La extensión y adaptabilidad se sustentan en un muy bajo costo desde el punto de vista de los desarrolladores debido a la tecnología utilizada, puramente constituida de objetos)

El segmento identificado para disparar el plan de marketing para la inserción de DIMBO en el mercado podría abarcar los siguientes subsegmentos :

1.
   Fuente : Listas de correos electrónicos de personas relacionadas al marketing y a la propia actividad del marketing. Se puede obtener desde los grupos de discusión nacional o desde grupos de noticias relacionados al dominio.
   Estrategia : Mail de presentación y dirección de ftp donde conseguir la demo del producto en forma gratuita.
2.
   Fuente : Cámaras de comercio de diferentes ciudades.
   Estrategia :Lograr conferencias gratuitas para presentar el producto ; Insertar la presentación del producto en conferencias relacionadas al marketing.

Seguimiento

Hay dos tipos de seguimientos que se deben realizar luego de la puesta en marcha de DIMBO en el mercado :

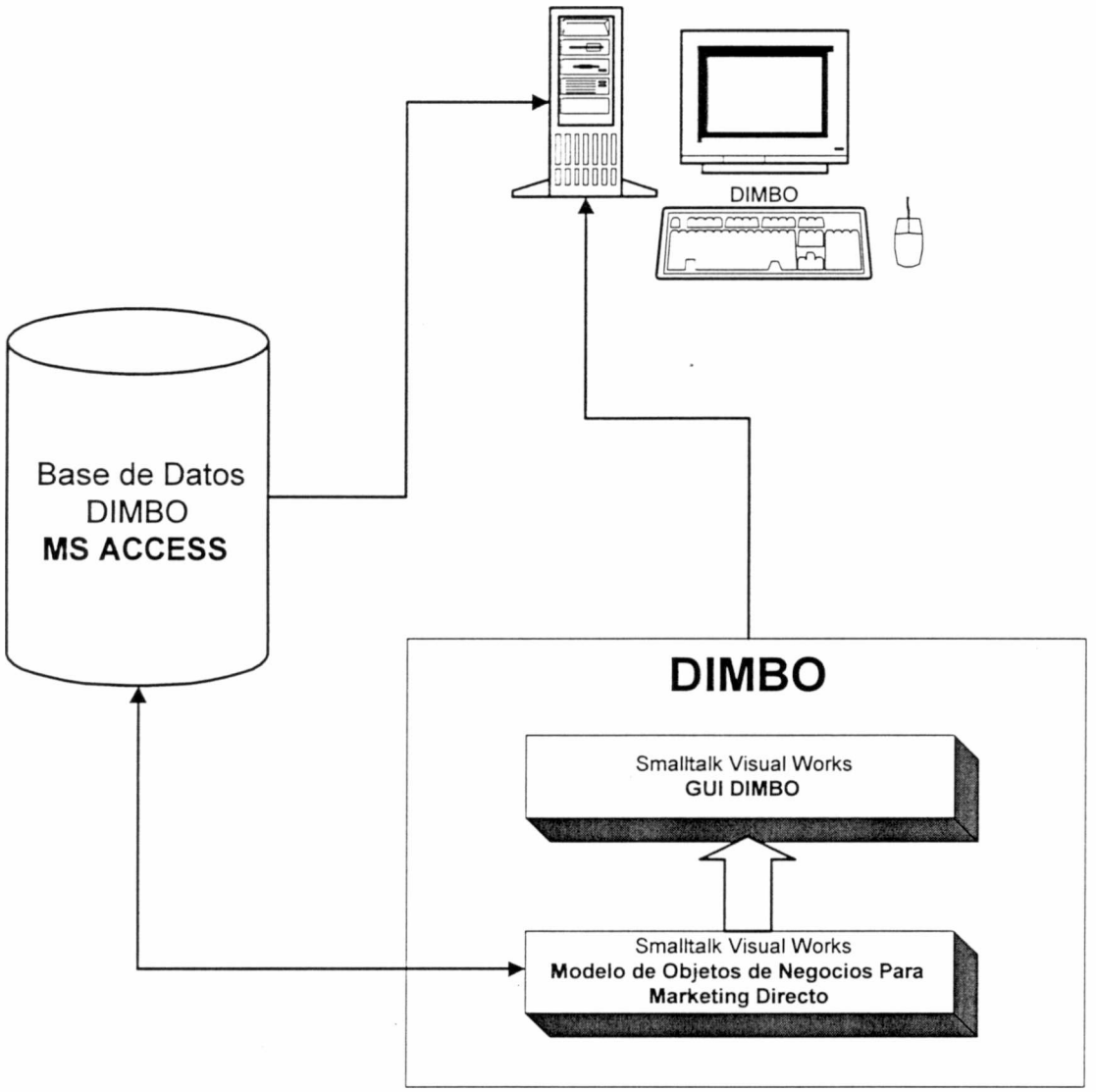
- Rastreo de Nuevas y Obsoletas Necesidades
- Rastreo de errores, incongruencias o comportamientos no esperados

La primera es la más costosa de conseguir porque requiere un feedback potente por parte de los usuarios. Así, independientemente que DIMBO sea adaptado a necesidades particulares de ciertos usuarios, se puede captar cuales son las necesidades globales que escaparon al proceso de estudio del dominio.
El segundo punto son la base de los nuevos releases que se realicen sobre DIMBO para corregir los errores que la aplicación incluye. Dichos errores pueden ser fruto de bugs en el código, como de malas interpretaciones del dominio o bien que la funcionalidad no es la que se esperaba realmente por parte de los usuarios. Todos ellos deben ser rápidamente solucionados (para demostrar solidez en el propio proceso de desarrollo) y entregadas al cliente potencial de la forma más veloz y práctica posible (vía patches, por ejemplo) para que el mismo pueda continuar con el uso de la demo (o de la propia aplicación final) sin desilucionarse del sistema.
La forma de obtener el feedback puede ir de las más costosas (seguimiento diario telefónico) hasta las más simples y eficientes (usar emails para obtener reporte de errores. incompatibilidaes e incongruencias).

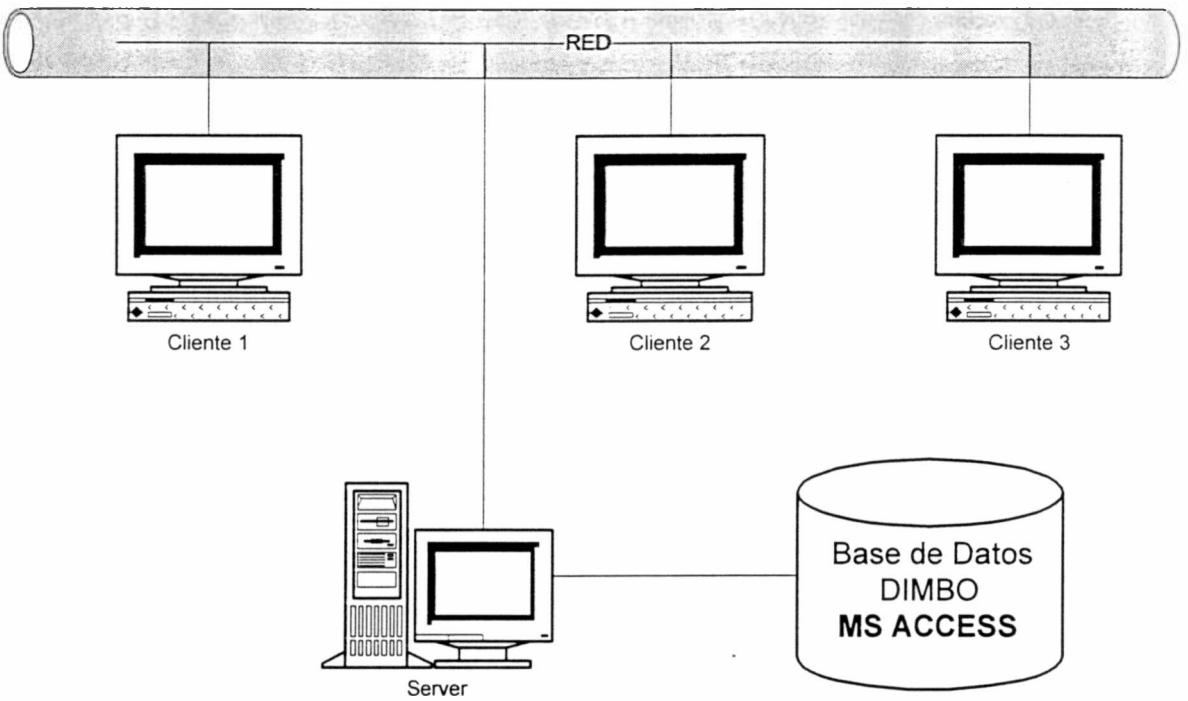
Aspectos de la Aplicación

Arquitectura de la Aplicación

DIMBO es tiene una arquitectura client-server con la particularidad de que el motor relacional puede correr tranquilamente en la misma máquina. Esto es, la base de datos utilizada (MS Access) recide en la misma computadora donde se ejecuta la imagen cliente de Smalltalk Visual Works.



La actual arquitectura soporta el funcionamiento multiusuario en diferentes computadoras y el acceso centralizado a la base de datos de Ms Access podría configurarse para ser accedido directamente sobre la red o sobre un server dedicado. Un esquema que represente lo anterior luciría de la siguiente manera :

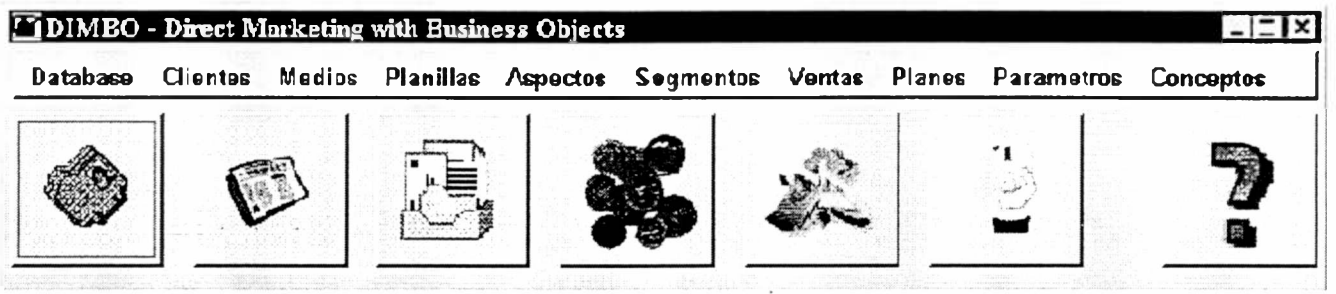


DIMBO presenta soporte para manejo de acceso por usuarios a la aplicación, pero la actual implementación no hace uso ni restricción de dicho esquema. Si bien la aplicación exige un usuario y una password para conectarse a la base de datos, cualquier información puede ingresarse para tal conección.

Manual funcional de DIMBO

1.1 DIMBO. La aplicación. Estándares de interface de la aplicación

DIMBO. Direct Marketing With Business Objects tiene como objetivo proveer al pequeño y mediano comerciante una herramienta capaz de controlar, dirigir y ejecutar las actividades primordiales del marketing directo.



Los principales conceptos del área son cubiertos por esta aplicación: Clientes, Medios de Venta/Distribución, Segmentos de Mercado, Segmentadores de Mercado, Ventas, Ofertas, Planes y Actividades de Marketing, Parametrización de Clientes y Conceptos ofrecidos al mercado.

La semántica de interface es lo suficientemente sencilla como para ser utilizada sin excesivos conocimiento previos o experiencias con herramientas de este estilo. El usuario inicialmente debe conectarse a la base de datos donde se sostiene la información de la aplicación. Para tal efecto cuenta con la opción de menú *Database* donde encontrará una subopción *Login* que lo conectará a dicha base de datos. Actualmente la aplicación -aunque lo solicita en una dialog box- no requiere password alguna, con lo cual bastará un simple <enter> para conectarse a la base de datos. Por otro lado, al dejar de utilizar la aplicación la operación inversa de *Logout* debería ser ejecutada desde el mismo menú.
Las características generales de la aplicación muestran cada ventana de edición con dos botones de *Aceptar* y *Cancelar*. Así mismo, los browsers cuentan además con dos botones de *Agregar* y *Borrar*. La funcionalidad de los mismos se desprende de los propios verbos que los representan. Es importante aclarar que mientras el botón de *Aceptar* no este grisado, la información que se este editando en pantalla no estará persistida en la base de datos. De esta manera, si se intenta cerrar una ventana con el botón de *Aceptar* no grisado, la aplicación le avisará que los actuales datos editados están a punto de ser descartados. En caso de aceptarse este aviso, se producirá el mismo efecto que presionar el botón *Cancelar*, el cual vuelve atrás todos los cambios iniciados sobre el objeto que se este editando en la ventana seleccionada.
Aquellas ventanas que no contengan ni *Aceptar* ni *Cancelar* significa que las operaciones que se realizan sobre los datos serán persistidas o automáticamente o bien con el *Aceptar* de la ventana padre desde donde se la abrió.

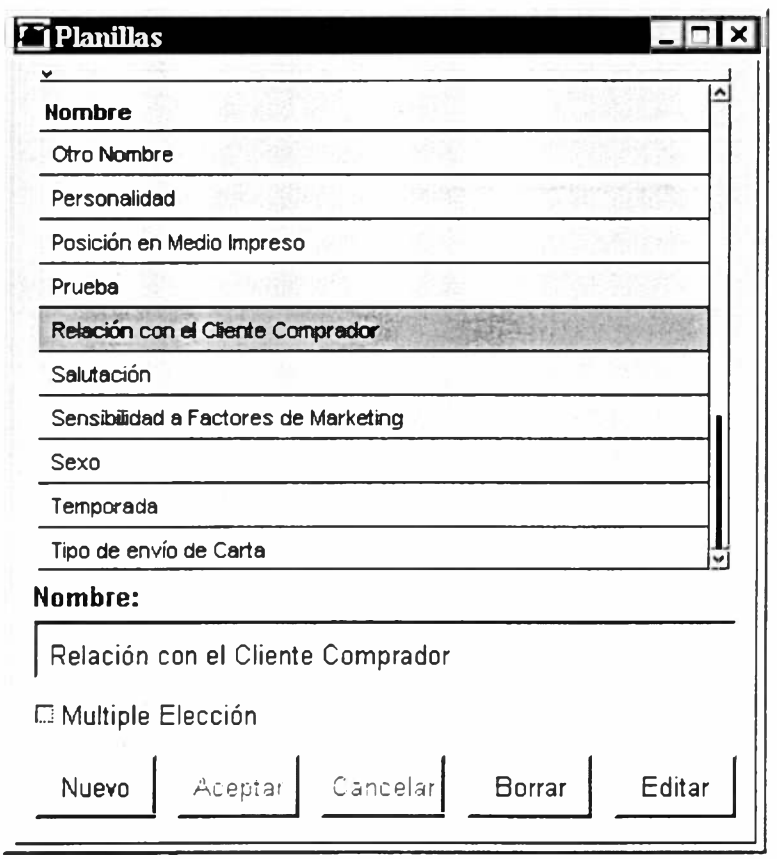
Los botones que aparecen en el Launcher principal se corresponden respectivamente a: Browser de Clientes, Browser de Medios, Browser de Planillas, Browser de Ventas, Browser de Planes de Marketing Directo y Browser de Conceptos.

Cada browser muestra una tabla de datos llamada dataset. Cada dataset refleja los aspectos más importantes de cada objeto en cuestión. Los datos de estos objetos se editan en los editores embebidos o bien desde otra interface a la cual se accede presionando el botón de *Editar*. Posicionándose sobre cualquier punto del dataset y presionando el botón derecho del mouse se puede acceder a dos opciones más: *columns sorter* y *columns selector*. La primera de ellas permite seleccionar las columnas por las cuales se desea aplicar el ordenamiento

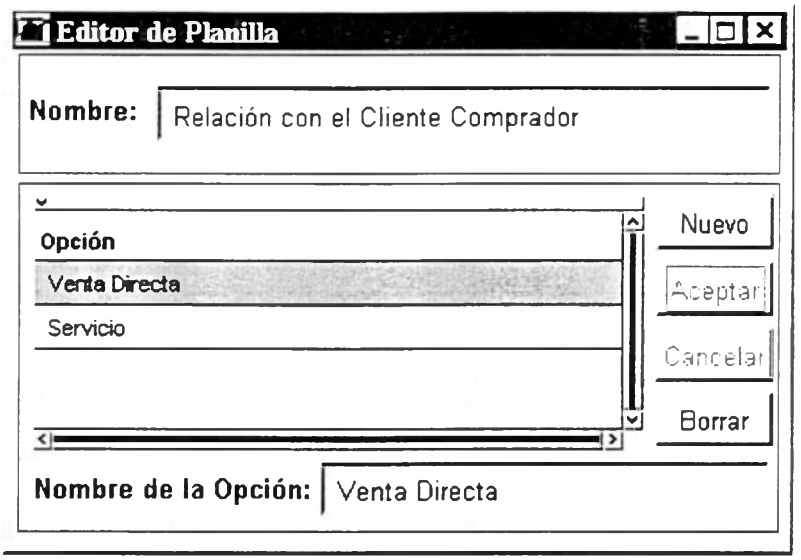
(sorting) del dataset. La segunda permite selecciona el orden en que se muestran las columnas de derecha a izquierda así también como eliminar temporalmente aquellas que no deseen visualizarse.

1.2. *Manejo de información tipificada.*

DIMBO posee la capacidad de tipar información a los efectos de no perder completitud en las acciones de segmentación y definición de aspectos de clientes. Esto significa que los usuarios podrán definir, extender y modificar las cantidad de tipos que la aplicación maneja respecto de un aspecto determinado. Por ejemplo, si un usuario desea manejar información acerca del sexo de sus clientes, este aspecto constituye un tipo dentro de su aplicación. Cada tipo en DIMBO es conocido como una planilla. Por cada planilla, entonces, el usuario predetermina un conjunto de valores posibles, los cuales -como se mencionaba anteriormente- pueden ser modificados, extendidos o borrados en forma dinámica según el deseo del usuario (siempre y cuando no se lo este utilizando en algún aspecto tipificado).
La principal ventaja de esta tipificación radica en el hecho de que nunca los usuarios tendrán información semánticamente similar pero textualmente diferente, perdiendo de esta manera completitud en la segmentación. Es decir, si el aspecto tipificado es el sexo y sus valores posibles son *masculino* y *femenino*, no existirá jamás un cliente que en dicho aspecto referencie a -por ejemplo- varón o mujer o male o como se le ocurra al operador de turno ingresar la información. De esta manera, a la hora de segmentar, un filtrado totalmente exacto es conseguido gracias a esta prestancia.

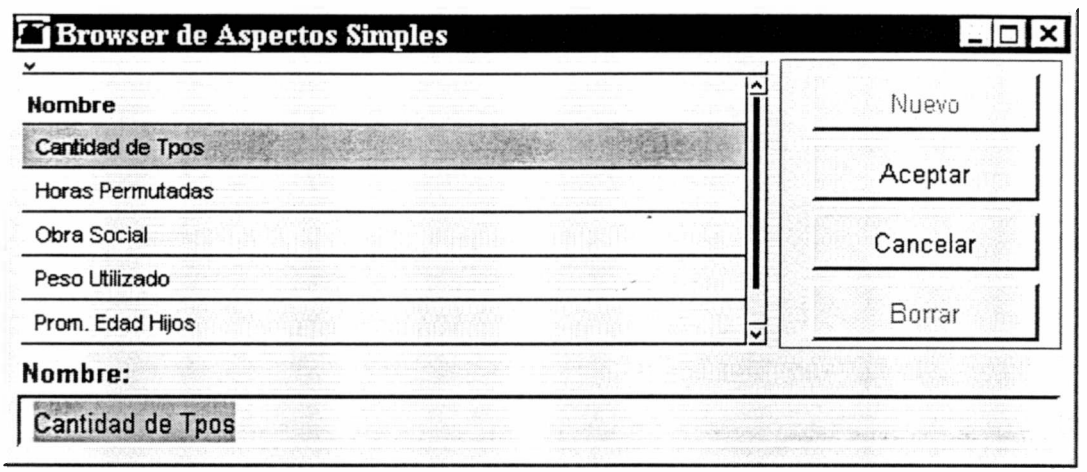


Algunas planillas existen de antemano para cubrir ciertos aspectos tipados de la propia aplicación (tal como el sexo de los clientes). Pero adicionalmente existe la posibilidad de que los usuarios extiendan la cantidad de conocimiento sobre cada cliente (ver 1.3), generando nuevos aspectos que pueden ser valores simples o bien tipados como es el caso de las planillas. En este caso, pueden reusarse planillas existentes o bien instanciarse nuevas planillas con sus respectivas opciones. Estas últimas se generan por intermedio del botón *Editar*:

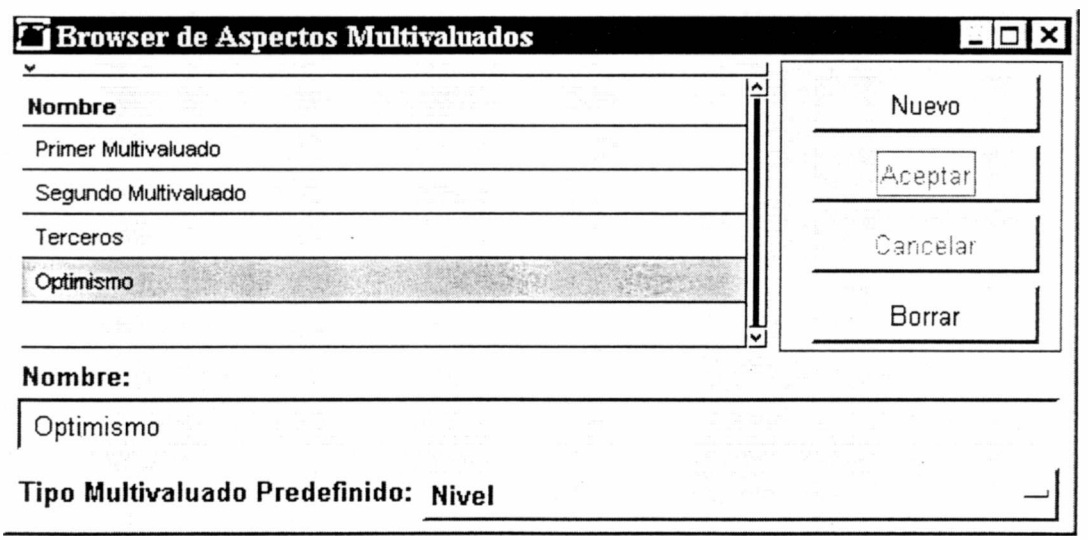

Editor de Planilla
Nombre:
Relación con el Cliente Comprador
Opción
Venta Directa
Servicio
Nuevo
Aceptar
Cancelar
Borrar
Nombre de la Opción:
Venta Directa

1.3. *Manejo de aspectos generados dinámicamente. Simples y Multivaluados.*

La información que se maneja en el contexto del marketing directo puede ser muy variada. Especialmente cuando se trata de clientes y fundamentalmente cuando se manejan diferentes contextos dentro del dominio del marketing. Por esa razón, una aplicación que permita solo un predeterminado número de aspectos sobre los clientes (fuente de toda razón y justicia del marketing), limitará su existencia a un mínimo cambio de requerimientos por parte de los usuarios.
Para cubrir esta falencia, DIMBO incorpora los aspectos dinámicos. Esta característica brinda al usuario la posibilidad de poder definir a su preferencia nuevos aspectos que desea conocer acerca de los clientes. Estos aspectos pueden ser simples datos (string de datos) o bien ser tipados con planillas (ver 1.2).



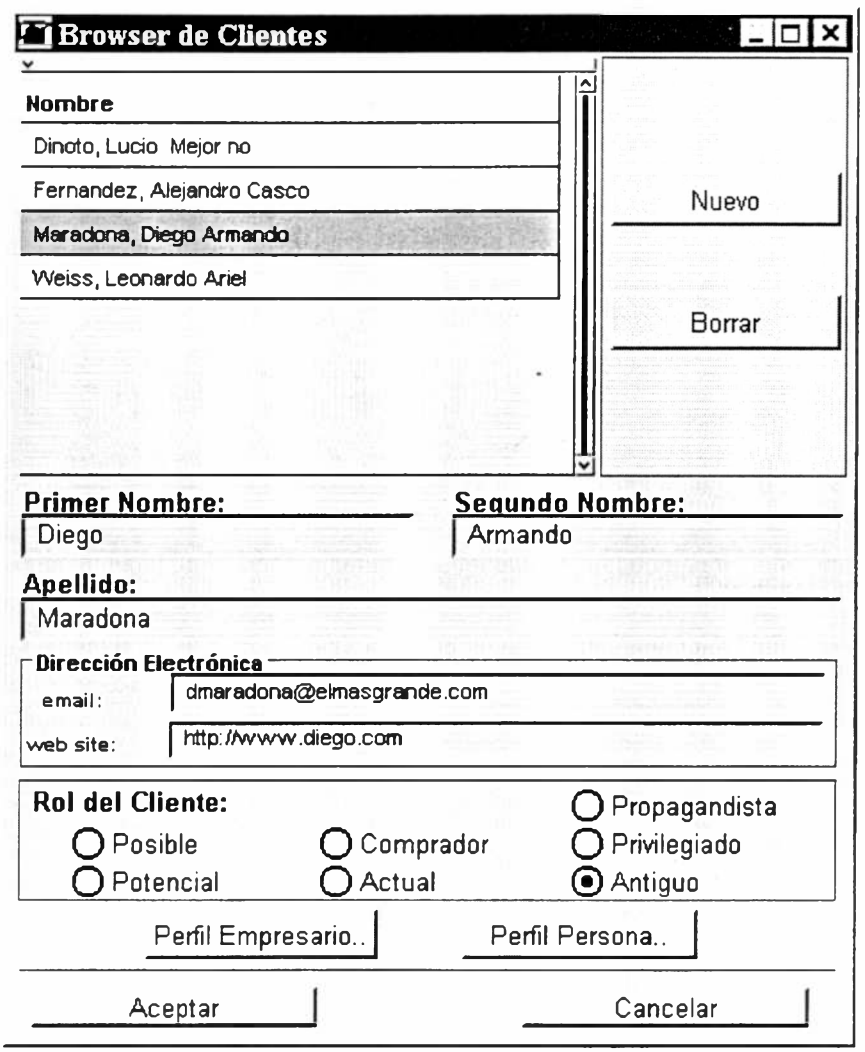
Los aspectos simples que muestra la interface anterior permiten al usuario ingresar cualquier tipo de información textual. Por otro lado, los aspectos multivaluados requieren que se indique el tipo de planilla sobre el cual se aceptarán opciones, tal cual muestra el browser siguiente:



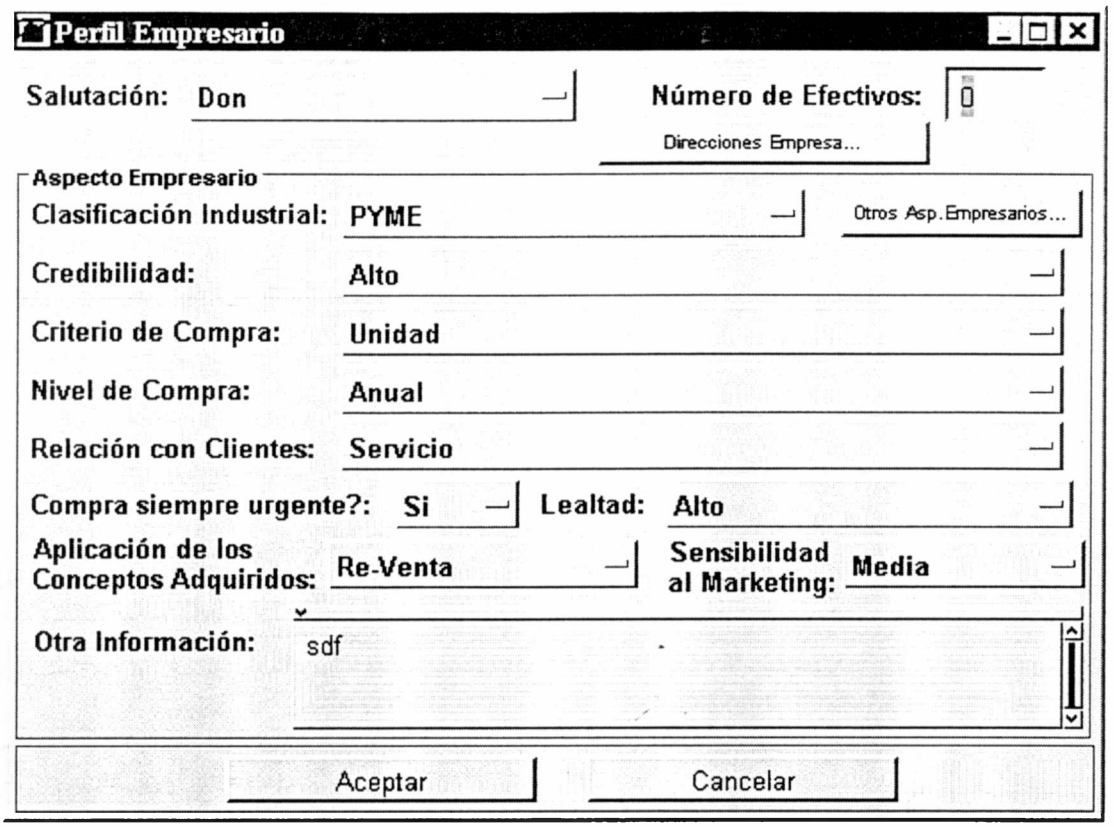
Otra vez, la segmentación resulta ser uno de los procesos beneficiados con esta particularidad de DIMBO, ya que también es posible segmentar clientes indicando el valor que se desea en cada unos de estos aspectos, indiferentemente si son simples o multivaluados.
En la sección 1.4, el usuario podrá notar como estos aspectos dinámicos son agregados como aspectos de clientes existentes.

1.4. ***Base de Clientes. Aspectos Empresario y Personal. Aspectos Tipados y aspectos Dinámicos.***

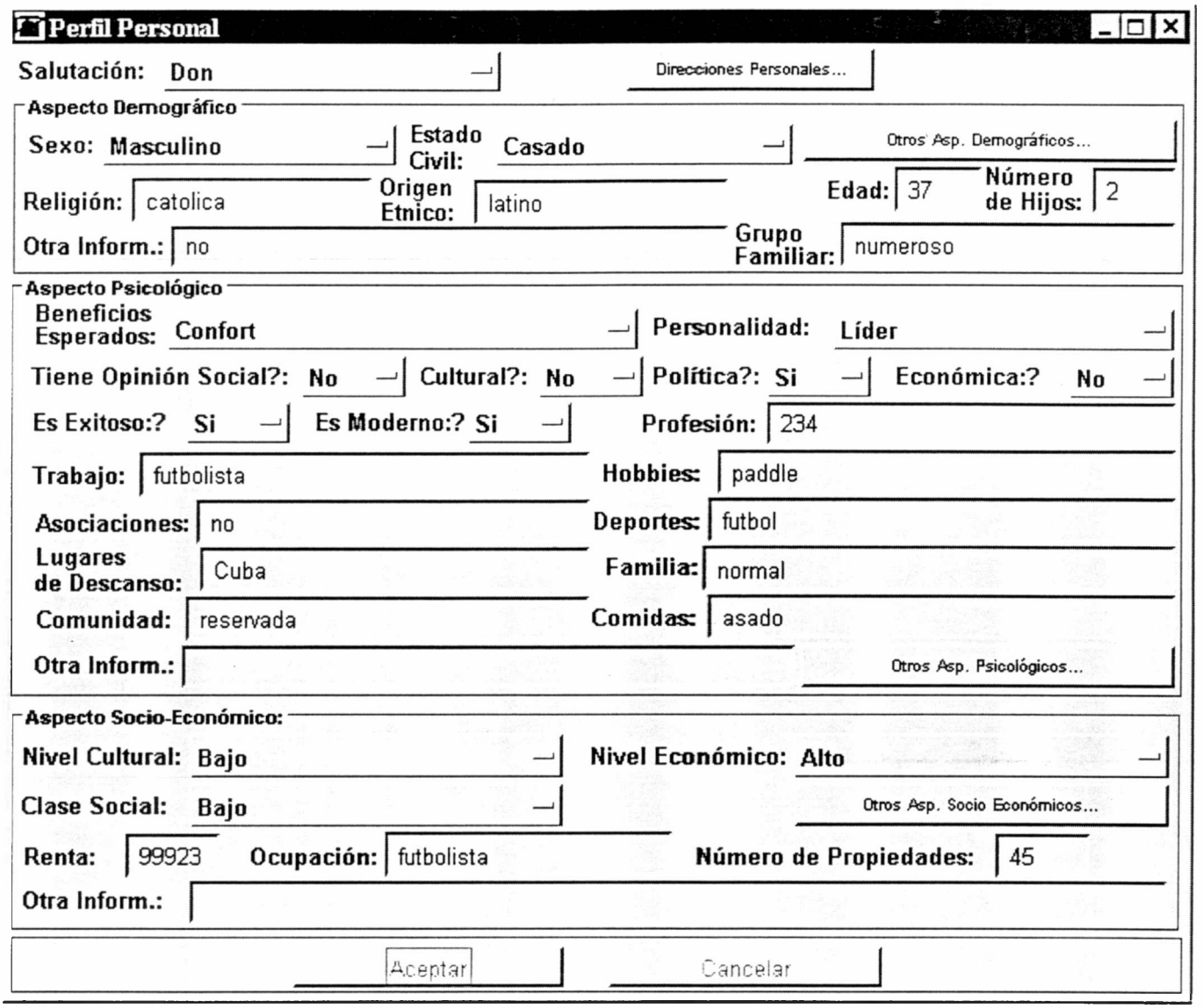
DIMBO se construyó bajo la premisa de poder saber todo cuanto se desee conocer acerca de los clientes. De esta manera, la información que se sostiene sobre los clientes provee una buena base para la segmentación y la parametrización de los mismos.



Los clientes tienen dos perfiles diferentes sobre los cuales el usuario extiende el conocimiento sobre los mismos: un perfil empresario y un perfil personal:



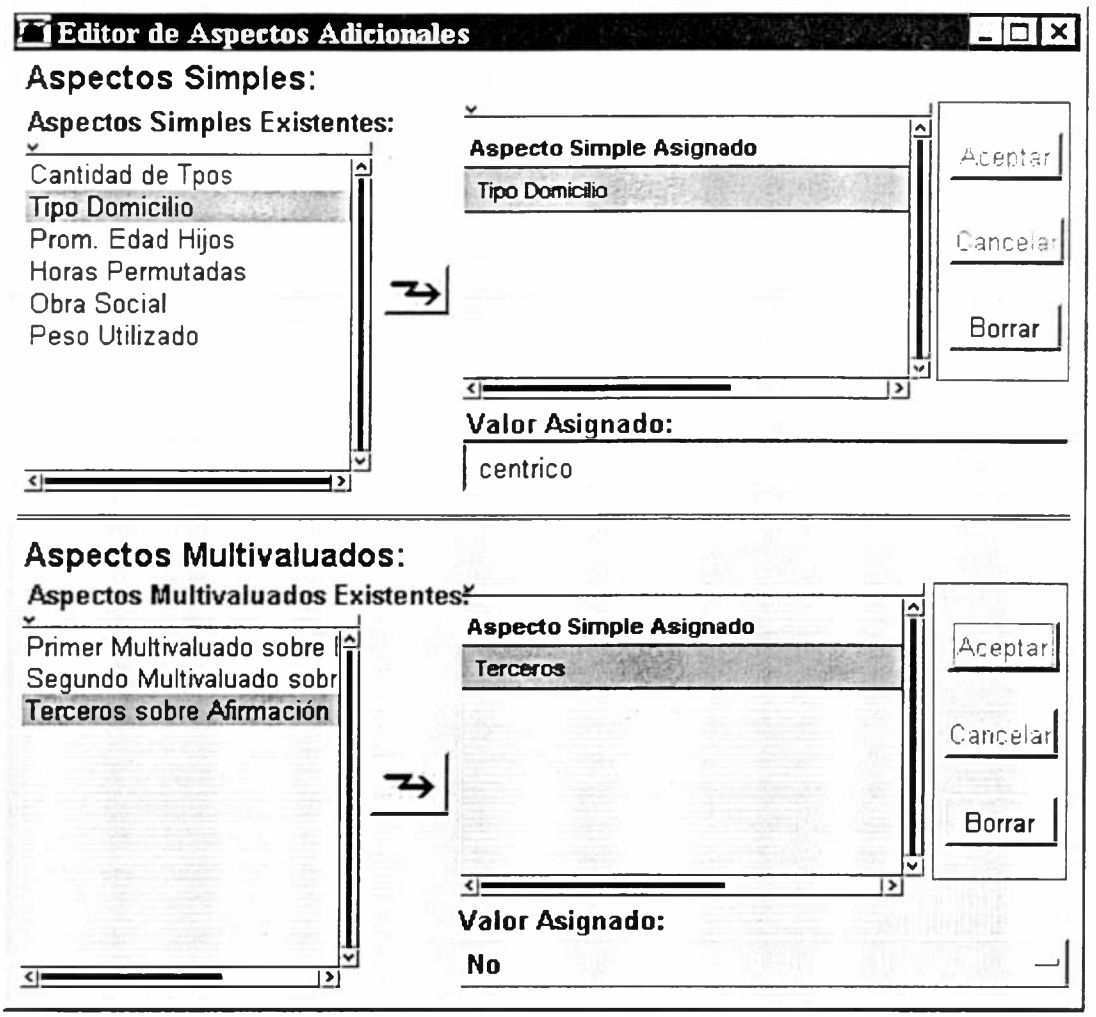
Notar como en el perfil empresario como en el personal aparecen informacion tipada (ver 1.3). Las posibles opciones que toma cada uno de esos aspectos (por ejemplo Lealtad) son las opciones predeterminadas en las planillas mencionadas anteriormente en el punto 1.3.



El perfil personal cuenta con tres marcadas secciones: el perfil socio-económico, el psicológico y el demográfico.

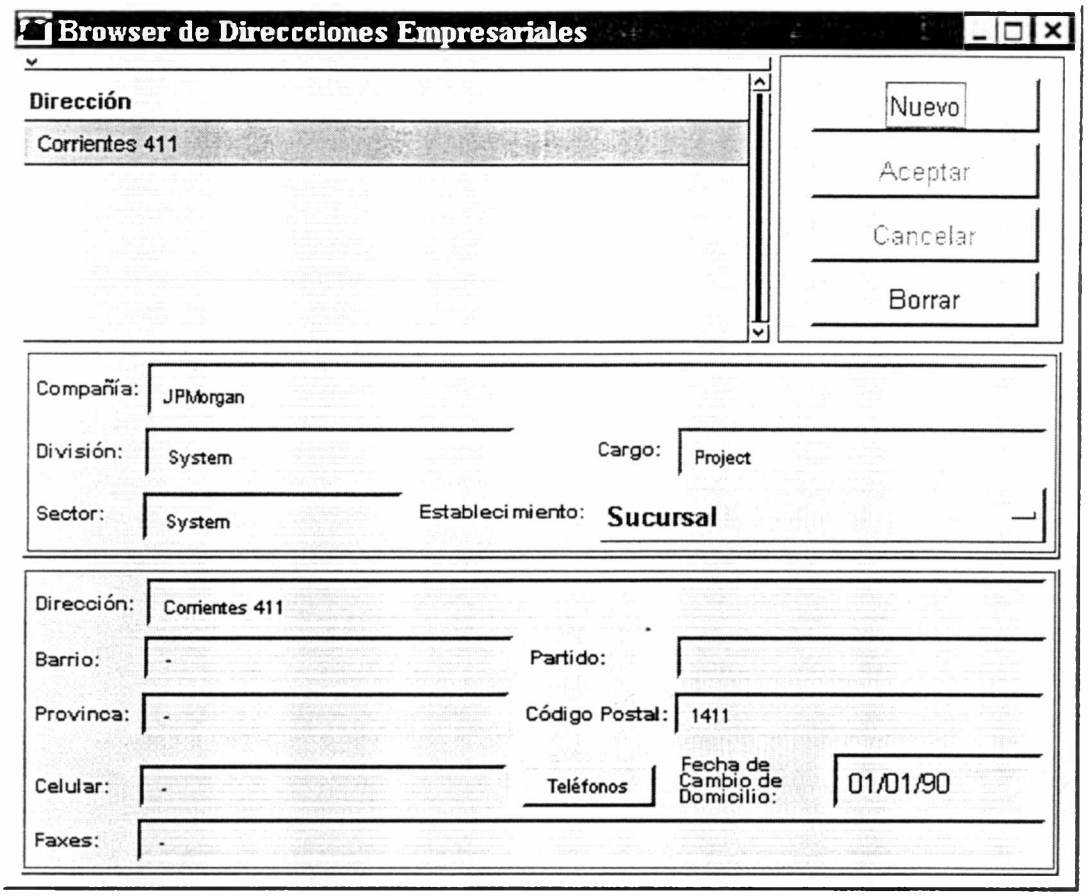
Toda la información no tiene porque se completada inicialmente y puede irse mejorando, actualizando o incorporando a medida que el manager conoce más y más acerca de sus clientes mediante los propios planes de marketing que instancia con DIMBO.

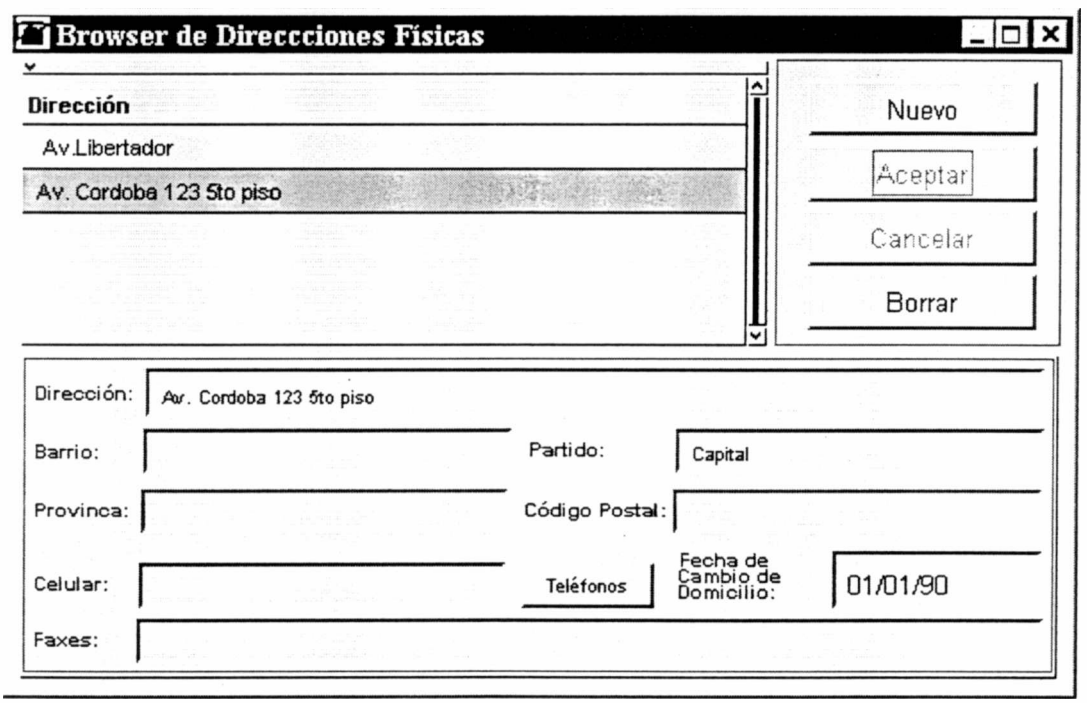
Cada uno de los cuatro aspectos discriminados, tanto en el perfil personal como en el empresario, cuentan con un botón que menciona *"otros aspectos..."*. Esto permite al usuario, por cada uno de los cuatro aspectos, agregar dinámicamente los aspectos simples o multivaluados (ver 1.3) y sus sendos valores para ese cliente en particular. Como un ejemplo, el resultado de presionar el botón *"Otros aspectos empresarios"* sería el siguiente:



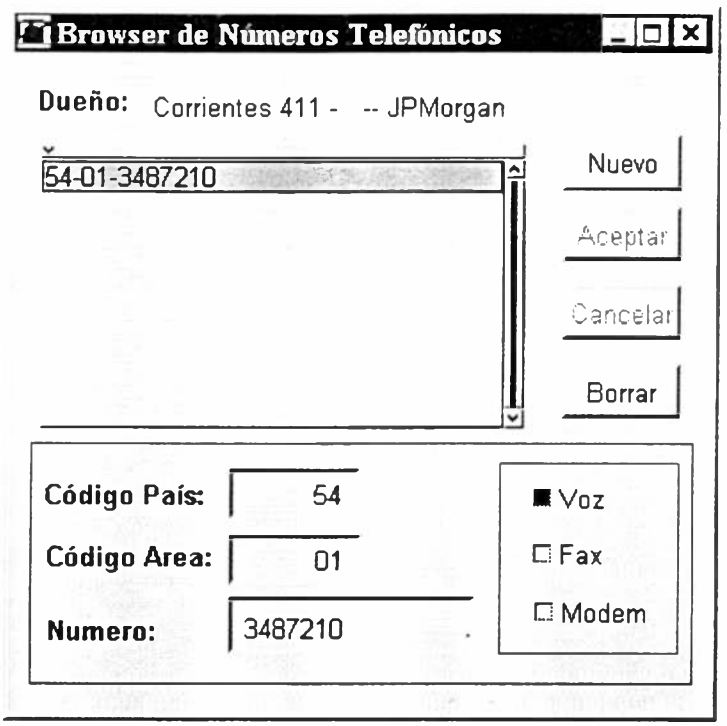
Notar como todas las instancias posibles de ambos aspectos (simple y multivaluado) son ubicadas en las listas de la izquieda, permitiendo al usuario arrastrarlas a la derecha por intermedio del botón-flecha. De esta manera, los aspectos dinámicos van adjuntándose sobre el cliente que se está editando con su respectivo valor. Esta valoración es, o simple (como en la parte superior de la ventana) o multivaluada (como en la inferior). En el primer caso, un simple contenido textual es definido como el valor de ese aspecto, mientras que en el segundo una de las opciones posibles de la tipificación seleccionada (ver 1.2) debe ser elegida.

Por último, los browsers de perfiles permiten asentar información de las direcciones -tanto empresaria como personal- de los clientes:



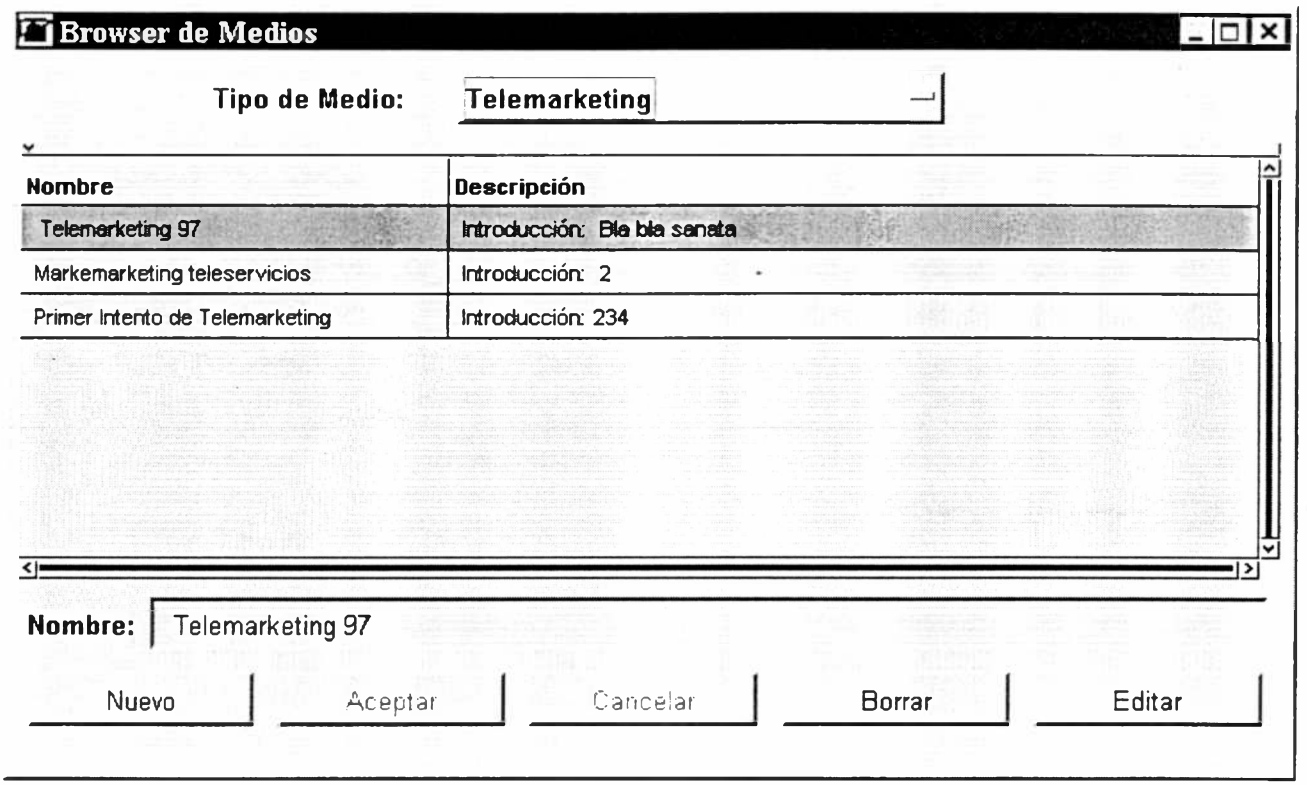


Cada uno de estos browsers permite asignar teléfonos a las respectivas direcciones:

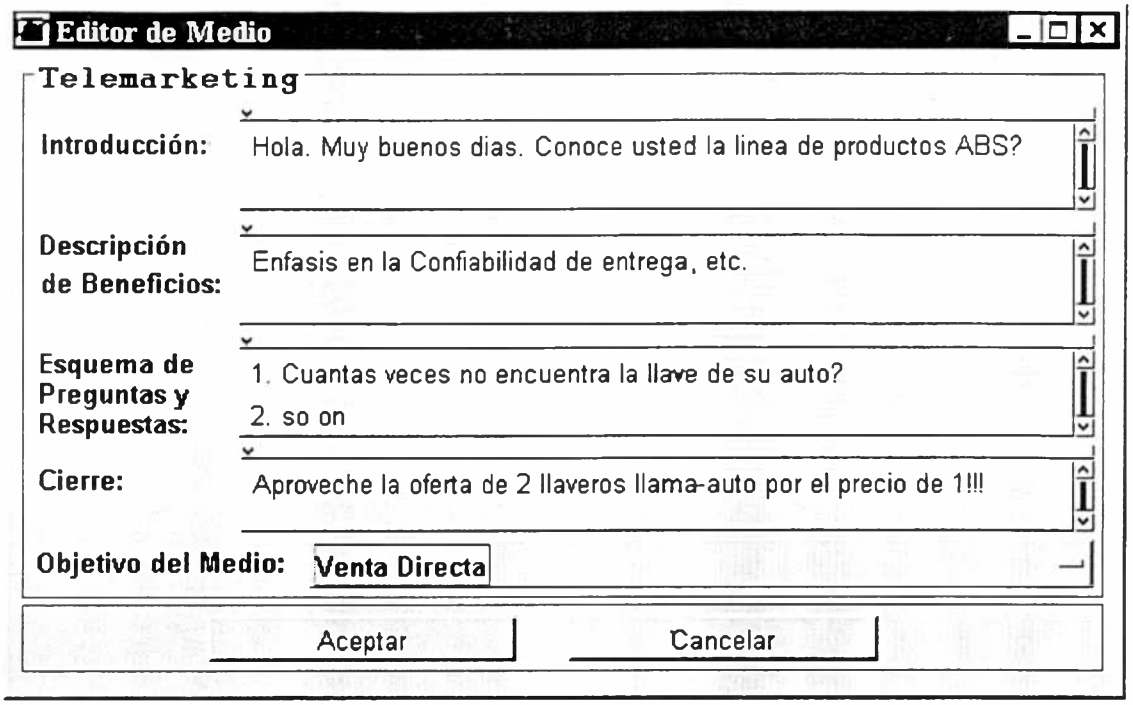

Browser de Números Telefónicos
Dueño: Corrientes 411 - -- JPMorgan
54-01-3487210
Nuevo
Aceptar
Cancelar
Borrar
Código País: 54
Código Area: 01
Numero: 3487210
Voz
Fax
Modem

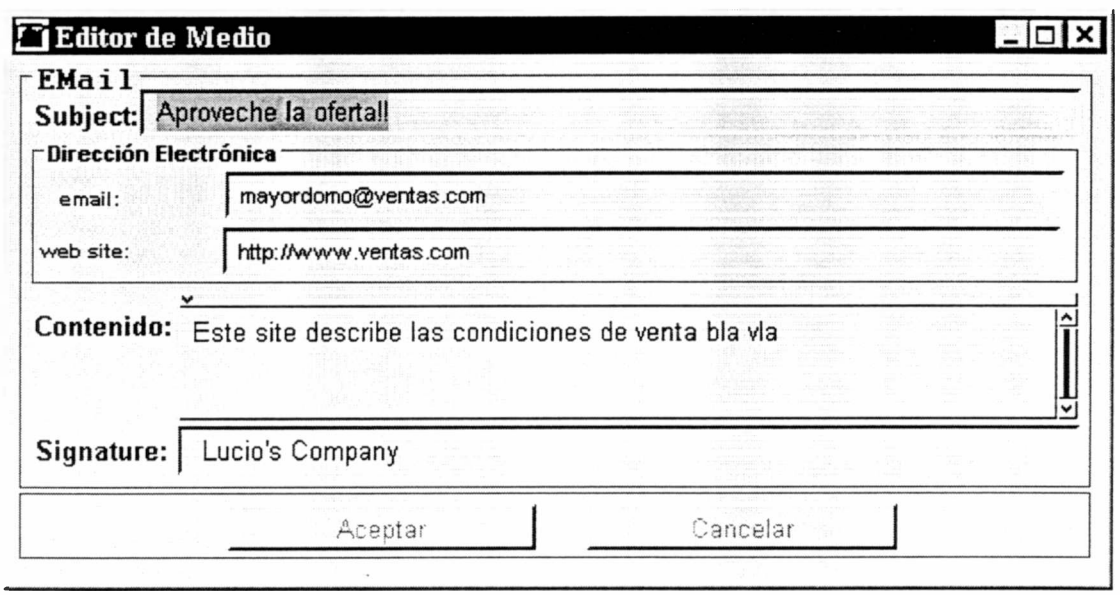
1.5. *Medios de Distribución.*

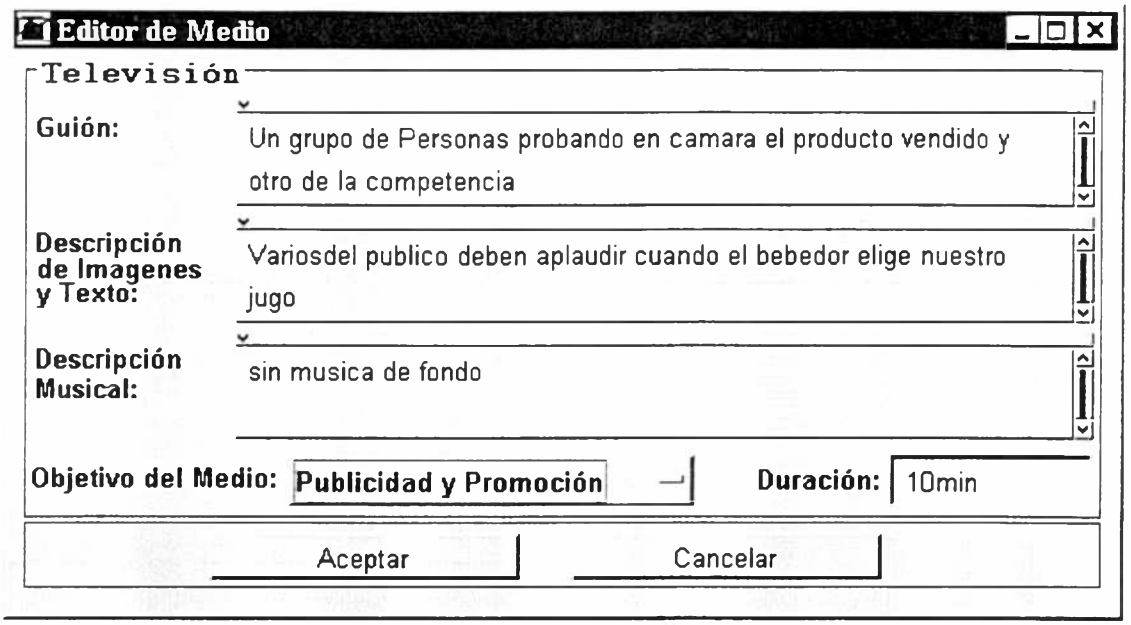
Los medios de distribución constituyen los rieles de comunicación entre los conceptos (la empresa) y los clientes. Los medios son asignados luego a cada plan de marketing que se instancia con la aplicación. Según su fuente (escrita, visual, etc.), los medios pueden contener diferente información.
DIMBO ha previsto que el marketing directo no solo consiste de mailing. Hay un variada cantidad de medios que pueden utilizarse para hacer marketing directo y pueden ser: Telemarketing, Televisión, Radio, Correo, Correo Electrónico, Venta Directa, Periódico, Fax y Revista.

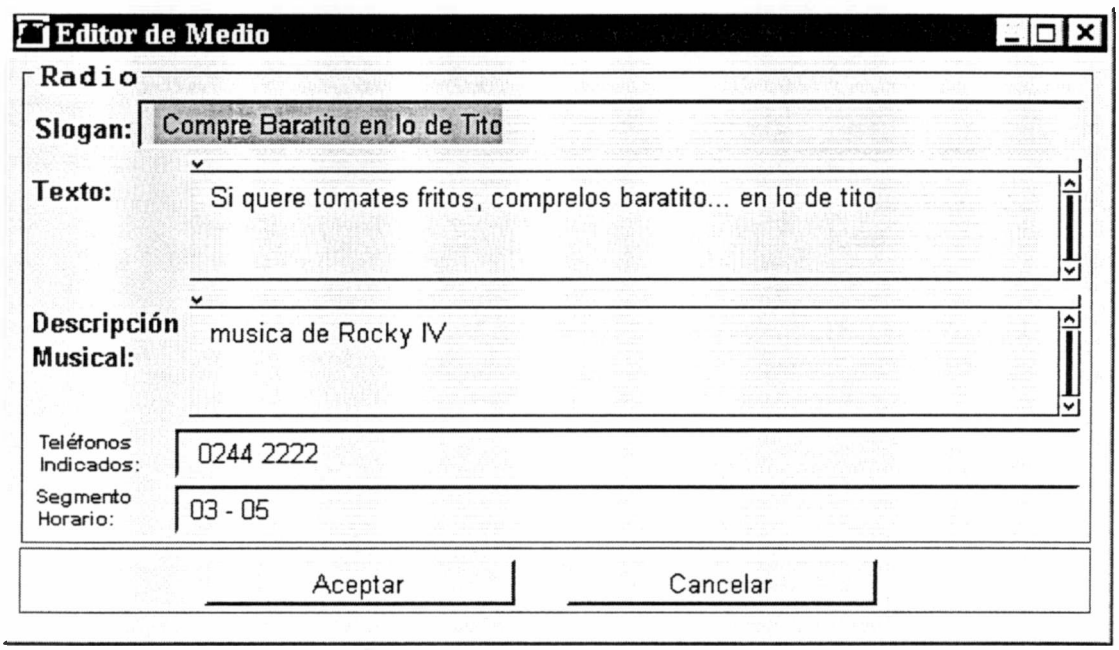


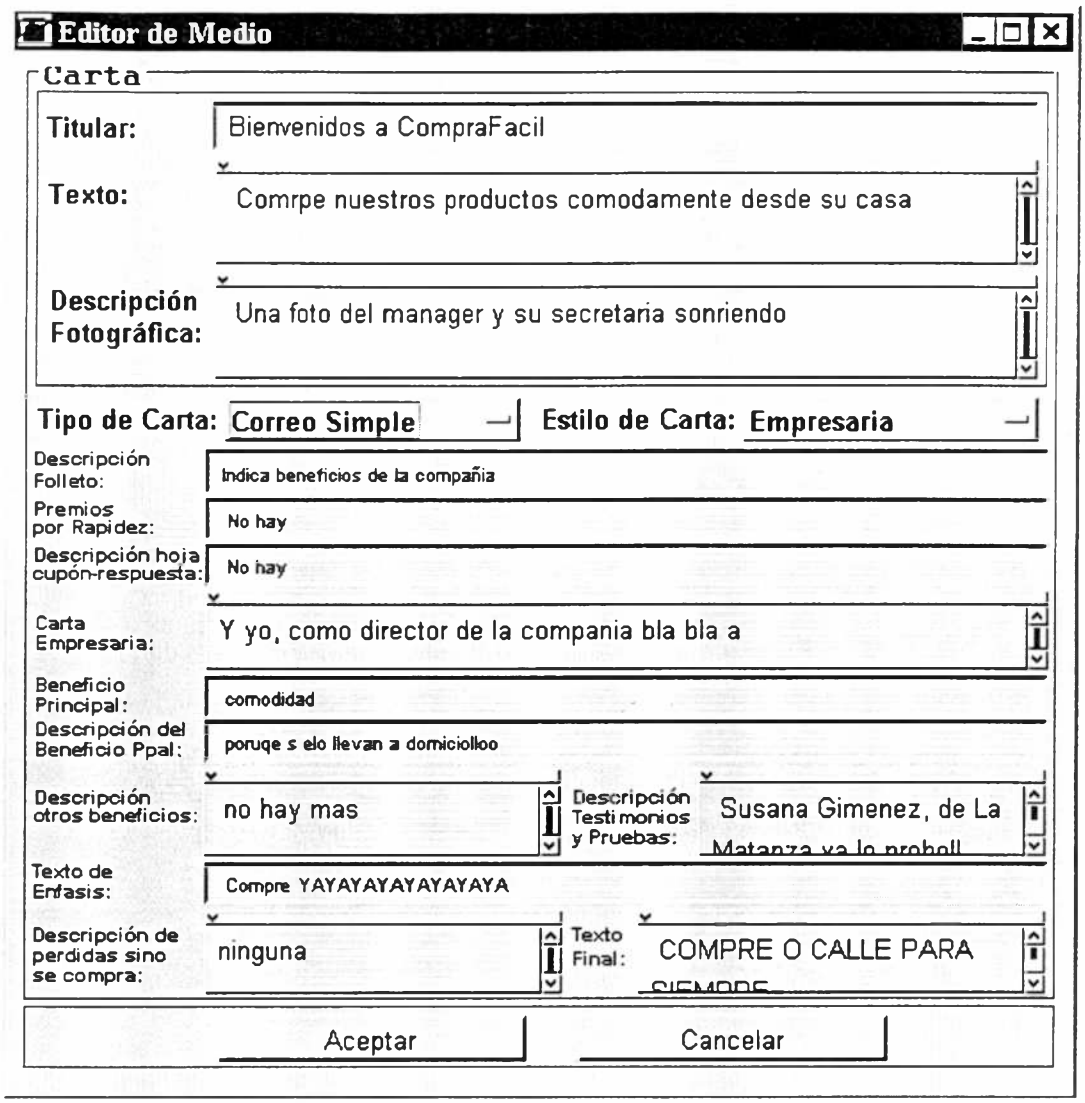
Seleccionando el menu-botón de la parte superior de la pantalla, los diferentes medios existentes para el tipo pueden visualizarse en el dataset de la interface.
Mediante el botón *Editar*, los aspectos de cada tipo de medio pueden ser configurados por el cliente:

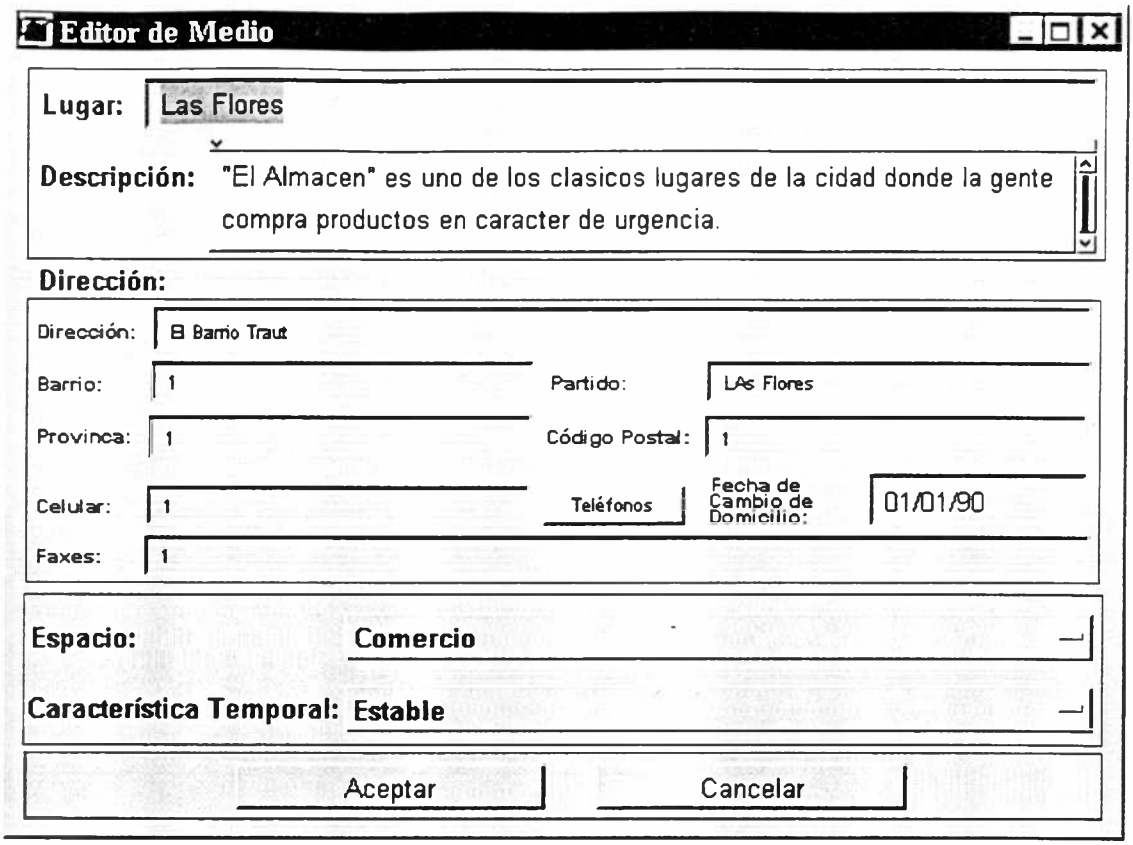

Editor de Medio
Telemarketing
Introducción:
Hola. Muy buenos dias. Conoce usted la linea de productos ABS?
Descripción de Beneficios:
Enfasis en la Confiabilidad de entrega, etc.
Esquema de Preguntas y Respuestas:
1. Cuantas veces no encuentra la llave de su auto?
2. so on
Cierre:
Aproveche la oferta de 2 llaveros llama-auto por el precio de 1!!!
Objetivo del Medio:
Venta Directa
Aceptar
Cancelar

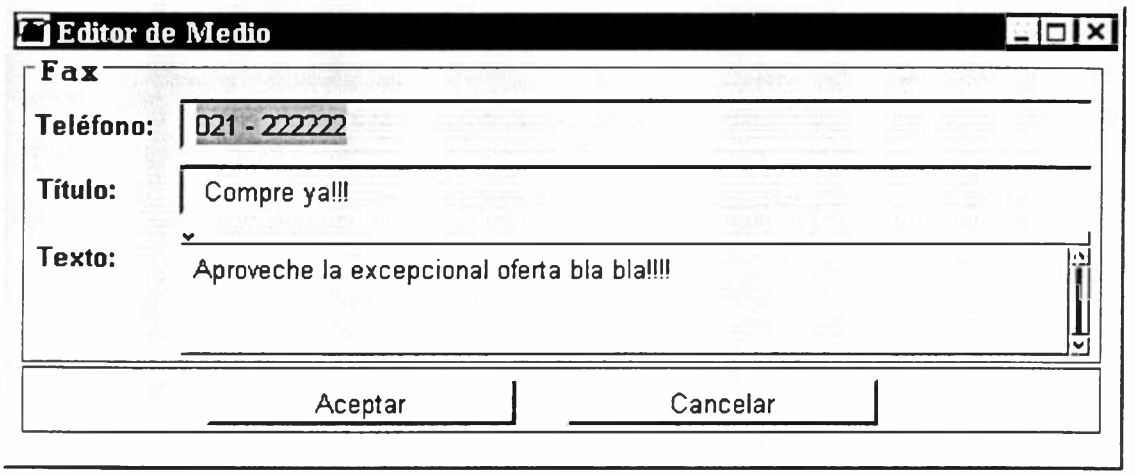

Editor de Medio
EMail
Subject:
Aproveche la oferta!!
Dirección Electrónica
email:
mayordomo@ventas.com
web site:
http://www.ventas.com
Contenido:
Este site describe las condiciones de venta bla vla
Signature:
Lucio's Company
Aceptar
Cancelar

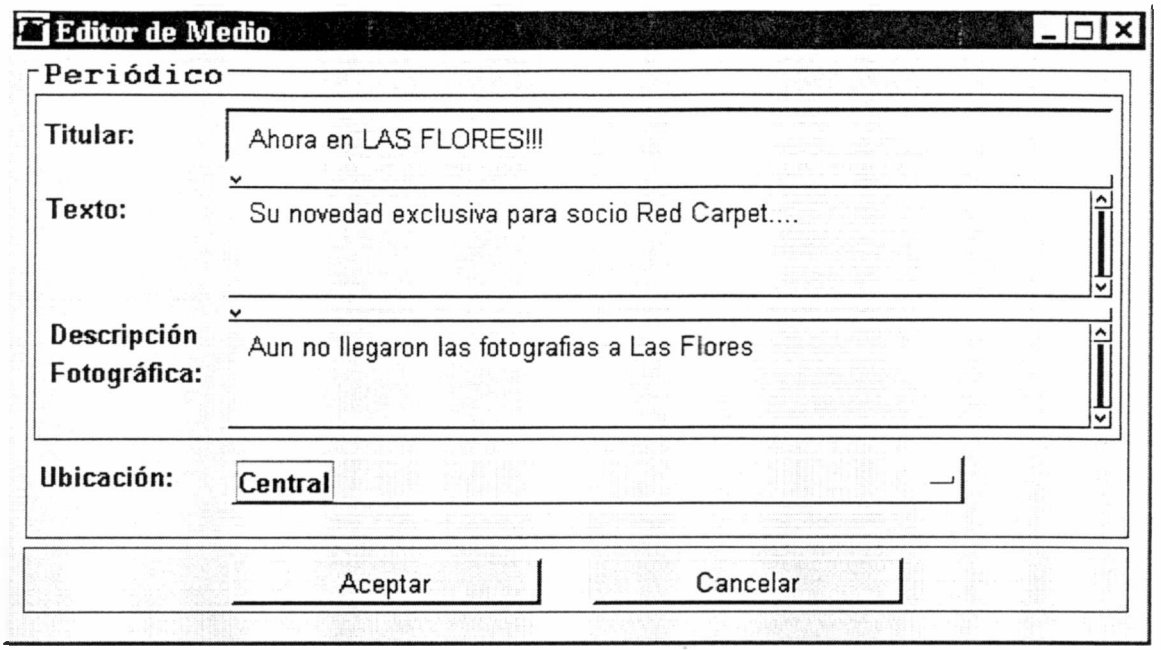

Editor de Medio
Televisión
Guión:
Un grupo de Personas probando en camara el producto vendido y otro de la competencia
Descripción de Imagenes y Texto:
Variosdel publico deben aplaudir cuando el bebedor elige nuestro jugo
Descripción Musical:
sin musica de fondo
Objetivo del Medio:
Publicidad y Promoción
Duración:
10min
Aceptar
Cancelar


Editor de Medio
Radio
Slogan:
Compre Baratito en lo de Tito
Texto:
Si quere tomates fritos, comprelos baratito... en lo de tito
Descripción Musical:
musica de Rocky IV
Teléfonos Indicados:
0244 2222
Segmento Horario:
03 - 05
Aceptar
Cancelar


Editor de Medio
Carta
Titular:
Bienvenidos a CompraFacil
Texto:
Comrpe nuestros productos comodamente desde su casa
Descripción Fotográfica:
Una foto del manager y su secretaria sonriendo
Tipo de Carta:
Correo Simple
Estilo de Carta:
Empresaria
Descripción Folleto:
Indica beneficios de la compañia
Premios por Rapidez:
No hay
Descripción hoja cupón-respuesta:
No hay
Carta Empresaria:
Y yo, como director de la compania bla bla a
Beneficio Principal:
comodidad
Descripción del Beneficio Ppal:
poruqe s elo llevan a domiciolloo
Descripción otros beneficios:
no hay mas
Descripción Testimonios y Pruebas:
Susana Gimenez, de La
Texto de Enfasis:
Compre YAYAYAYAYAYAYAYA
Descripción de perdidas sino se compra:
ninguna
Texto Final:
COMPRE O CALLE PARA
Aceptar
Cancelar


Editor de Medio
Lugar:
Las Flores
Descripción:
"El Almacen" es uno de los clasicos lugares de la cidad donde la gente compra productos en caracter de urgencia.
Dirección:
Dirección:
El Barrio Traut
Barrio:
1
Partido:
LAs Flores
Provinca:
1
Código Postal:
1
Celular:
1
Teléfonos
Fecha de Cambio de Domicilio:
01/01/90
Faxes:
1
Espacio:
Comercio
Característica Temporal:
Estable
Aceptar
Cancelar

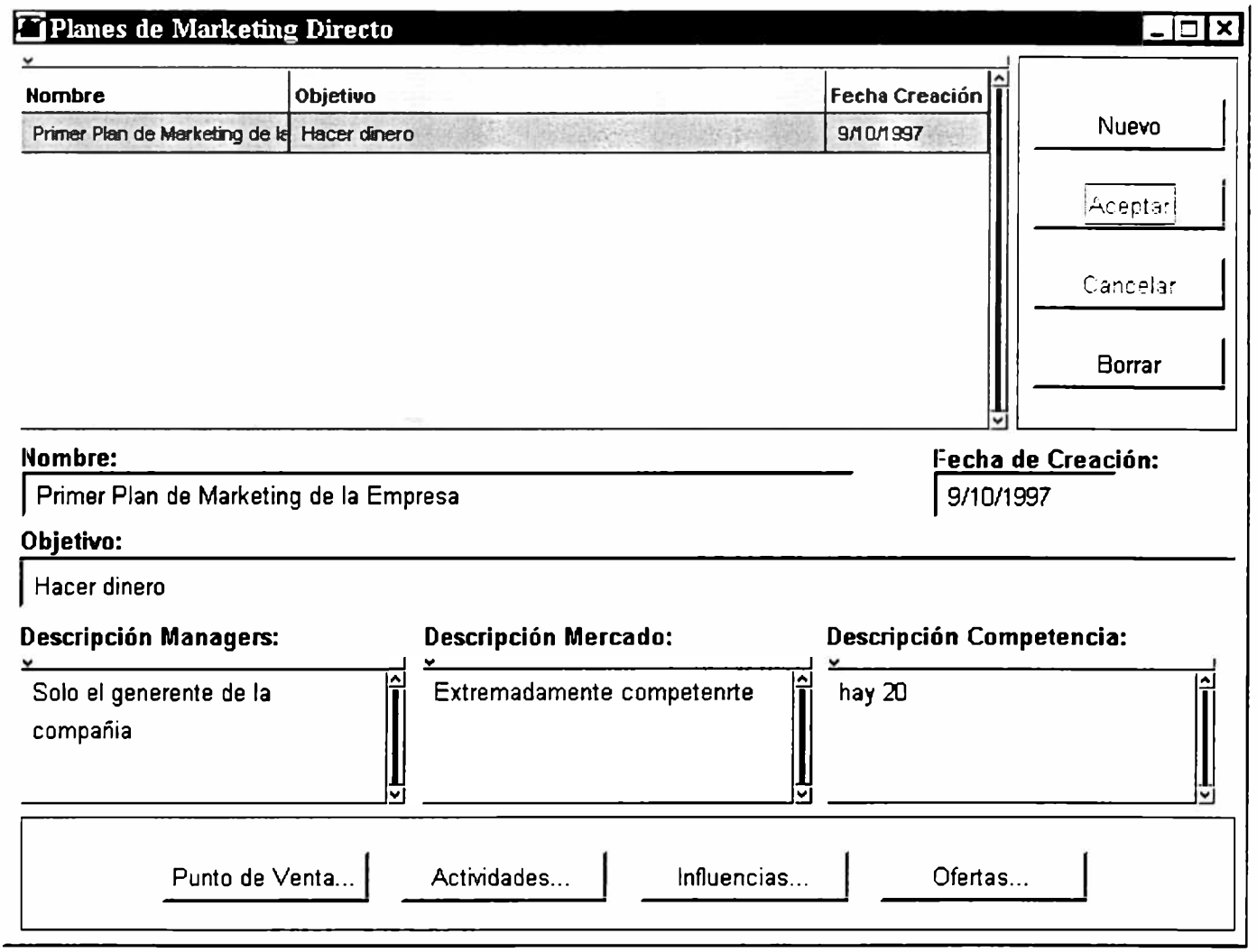

Editor de Medio
Fax
Teléfono:
021 - 222222
Título:
Compre ya!!!
Texto:
Aproveche la excepcional oferta bla bla!!!!
Aceptar
Cancelar


Editor de Medio
Periódico
Titular:
Ahora en LAS FLORES!!!
Texto:
Su novedad exclusiva para socio Red Carpet....
Descripción Fotográfica:
Aun no llegaron las fotografias a Las Flores
Ubicación:
Central
Aceptar
Cancelar


Editor de Medio
Revista
Titular:
Aprovechel!!
Texto:
Acerquese a nuestras oficinas....
Descripción Fotográfica:
sin fotos
Posición:
Sin Posición Concreta
Afinidad Concepto-Revista:
Medio Alto
¿Publica la competencia?:
No
Temporada:
Invierno
Porcentaje ocupado sobre la hoja:
10
Aceptar
Cancelar

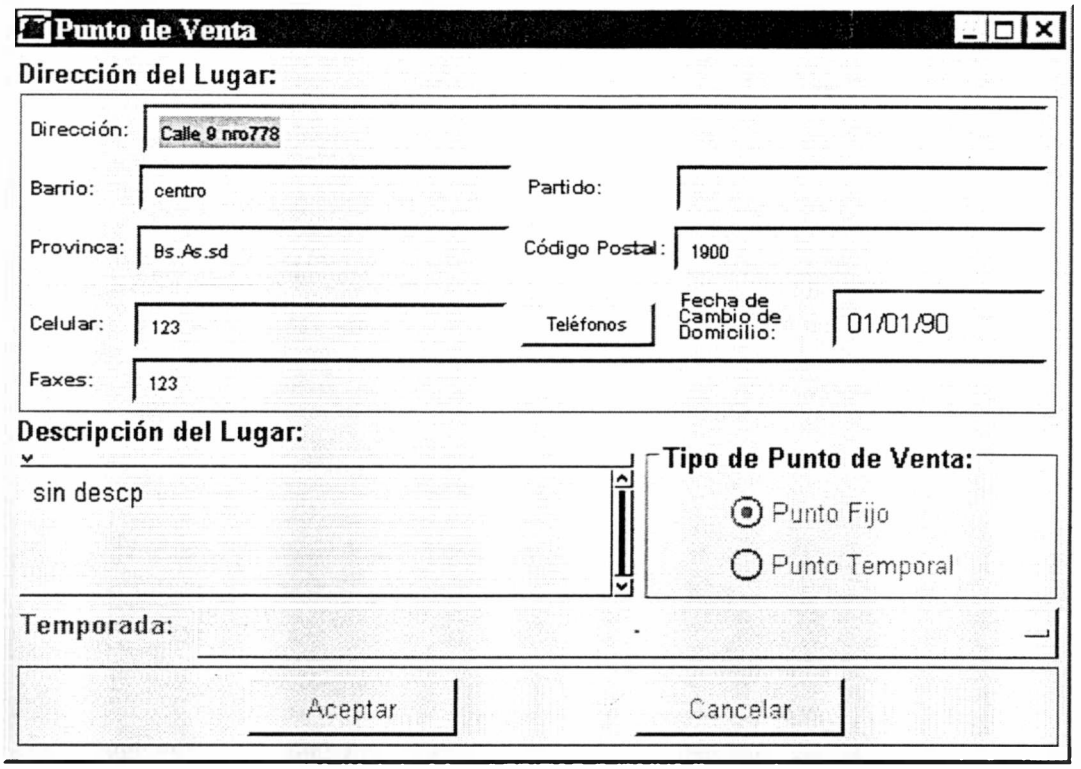
1.6. Generación de Planes de Marketing. Influencias. Actividades. Puntos de Venta. Ofertas asociadas.

Cada planeamiento en torno al marketing directo que lleva a cabo el comerciante usuario de DIMBO, debería ser identificado, dirigido, controlado y ejecutado acorde a la información que se maneja en las instancias de planes de marketing:

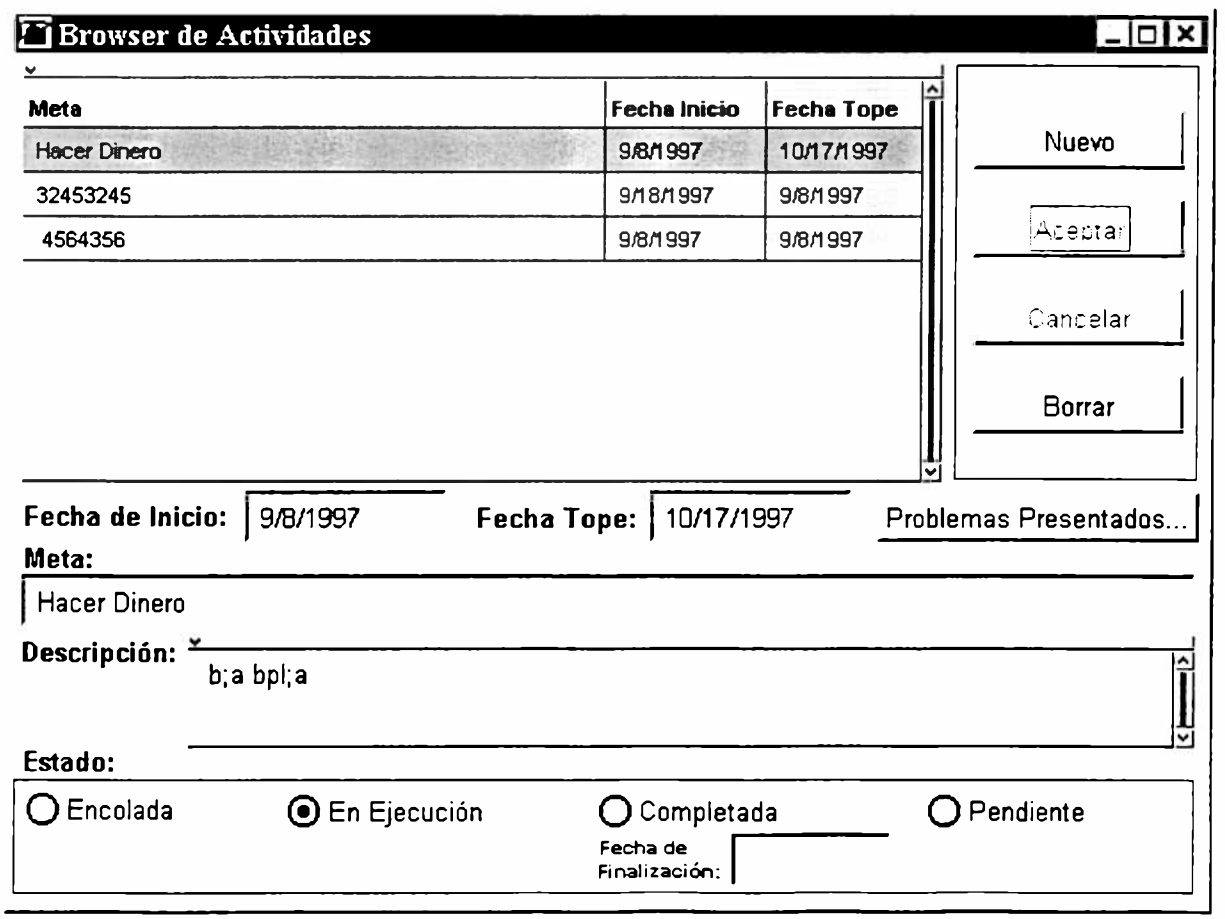


La Información sostenida alrededor de cada plan de marketing consta de cuatro puntos:

- El Punto de Venta donde se dispara el plan, tal cual lo muestra la interface siguiente. Tener en cuenta que el tipo de punto de venta (temporal o fijo) se establece en el momento de creado. Posteriomente, el mismo permanece inmodificable. En el caso de ser de tipo temporal, un aspecto tipado puede ser completado indicando la estación del año.



- Las Actividades del plan donde los problemas que se van sucediendo pueden ser claramente documentados. También un status de la actividad es establecido con la particularidad de que, en caso de estar completada, la fecha de finalización debe ser adjuntada al status de la actividad.

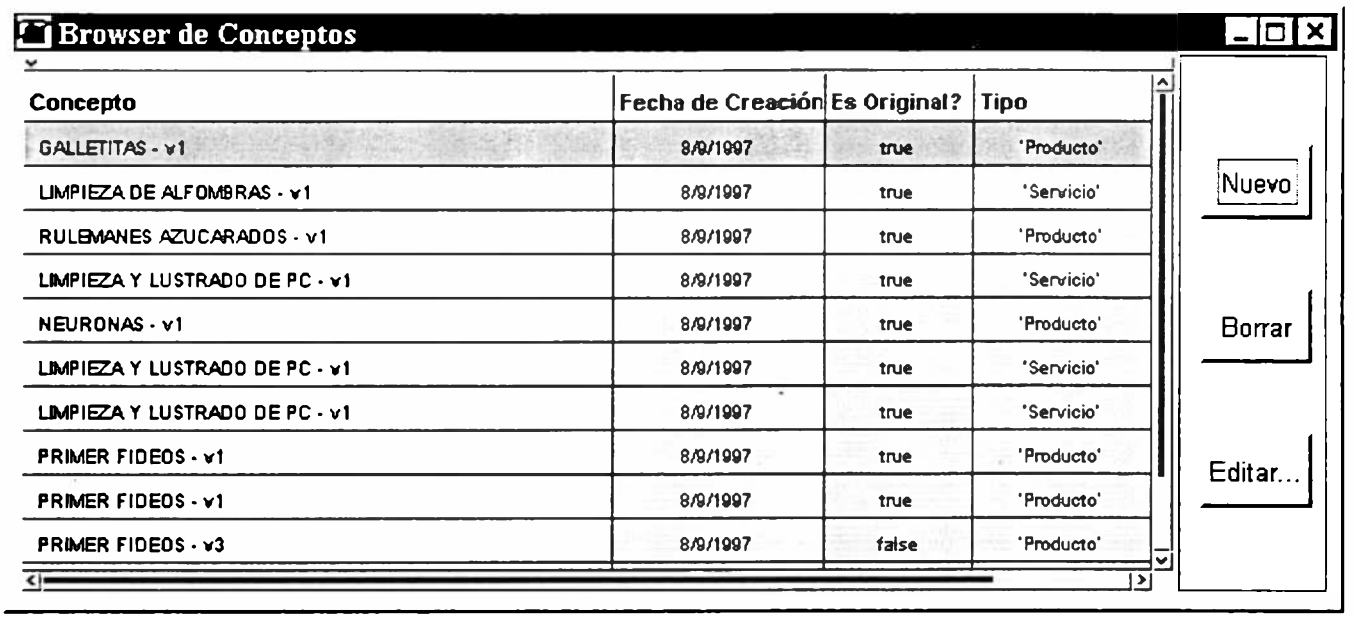


- Las Influencias que el propio plan recibe en su ciclo de vida:

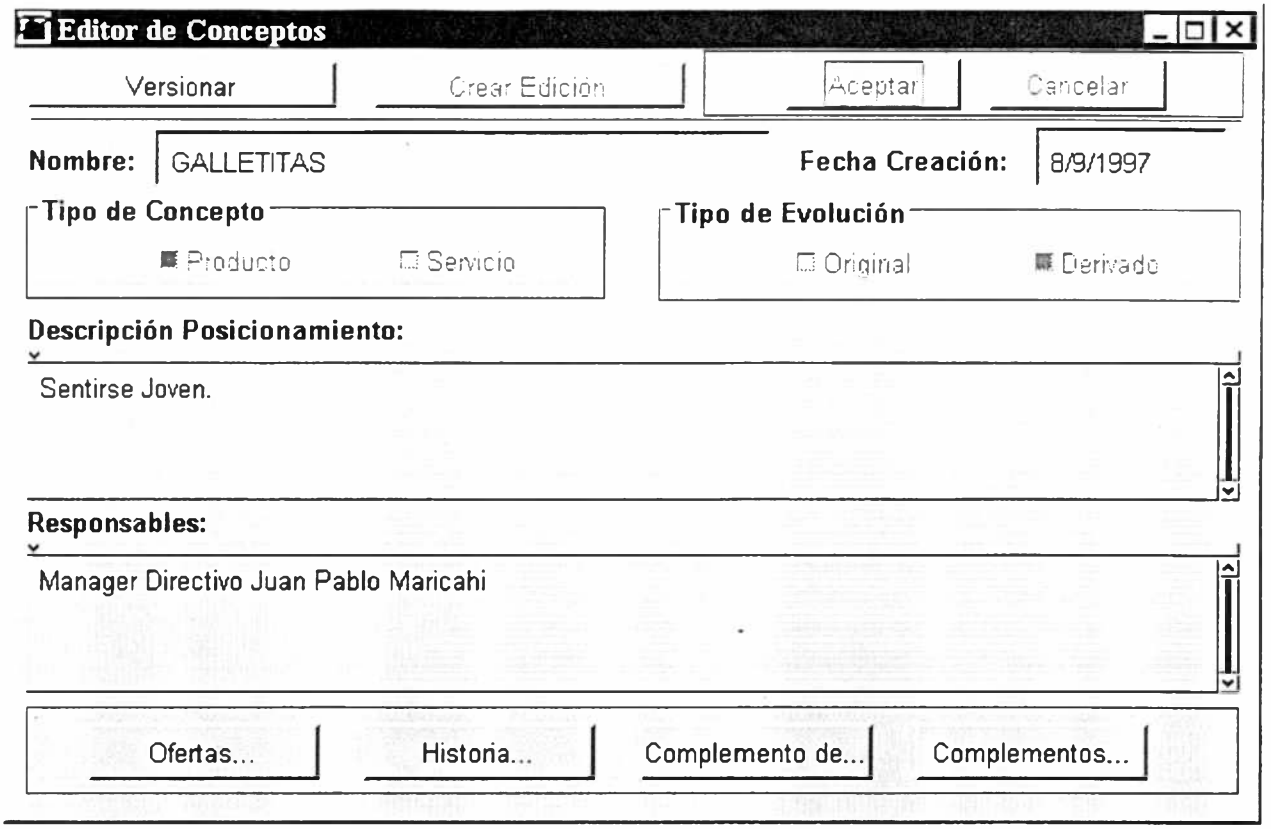
- Y las ofertas, punto el cual se referencia en 1.8.

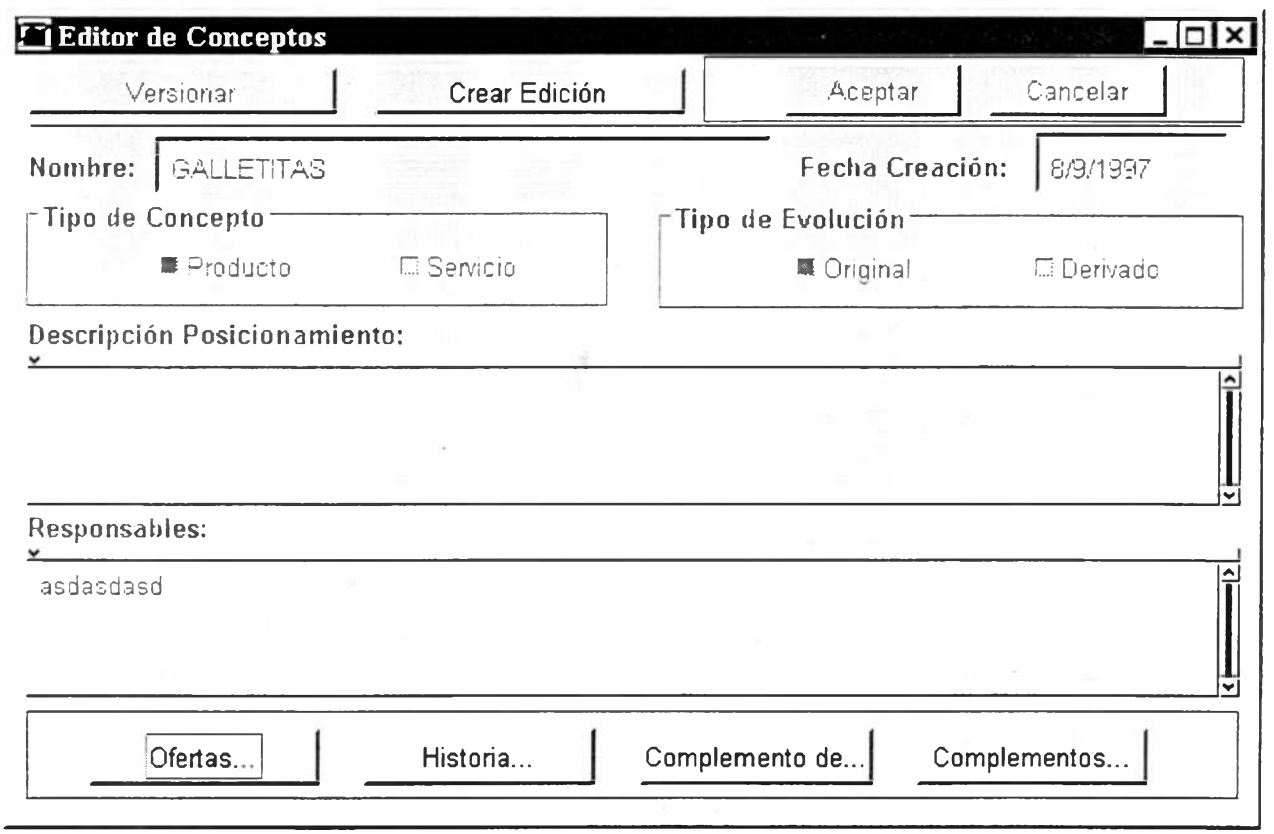
1.7. Instanciación de Conceptos. Productos y Servicios. Versionamiento y línea de evolución de conceptos. Historia. Edición y Versión de conceptos.

Los conceptos son la base de sustento del comercio. El manager instancia conceptos que pueden ser tanto productos como servicios:

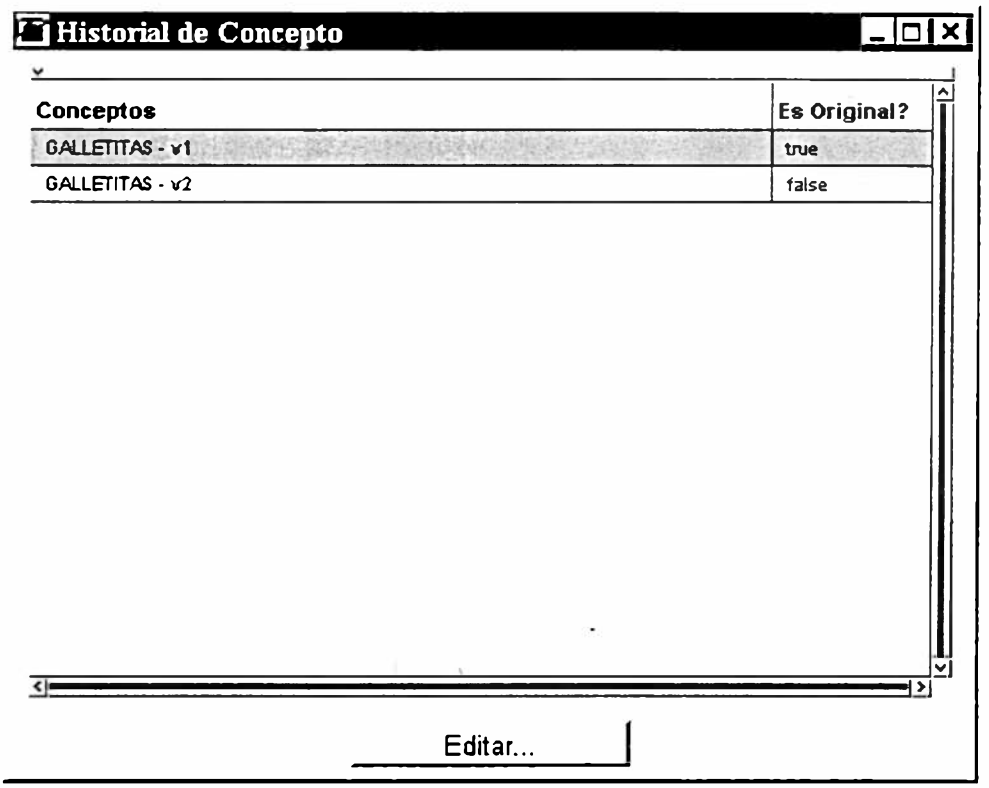


Cada concepto puede ser original o bien ser un eslabón en la evolución del mismo (notar como la descripción del concepto indica su versión: v1, v2, etc.).
La cadena de evolución va indicando como el concepto ha sido mejorado, adaptado, modificado a lo largo de su existencia (ver Versioning Pattern en la documentación de la tesis).
Los conceptos pueden estar *versionados* o *en edición*. El primer caso se corresponde a conceptos que ya no pueden ser modificados y permanecen eternamente en ese estado (en el que fueron versionados). Sin embargo, esto no implica que no puedan ser utilizados en ofertas actuales o futuras. Por el contrario, aquellos que están en edición permanecen abiertos a cambios constantes. A partir de una edición puede realizarse una nueva version y "cerrar" el concepto. Por el contrario, una versión de un concepto puede ponerse en edición edición como un nuevo concepto para ser ofrecido al mercado, el cual es modificado a antojo del usuario sin afectar a su versión origen. Por último, cuando se crea un nuevo concepto (original), el mismo es iniciado en estado de edición.
Cada concepto puede ser editado vía el botón *Editar*. Según el concepto esté en edición o esté versionado, los aspectos editables aparecerán grisados o disponibles para modificación, tal cual lo muestran sendos editores:

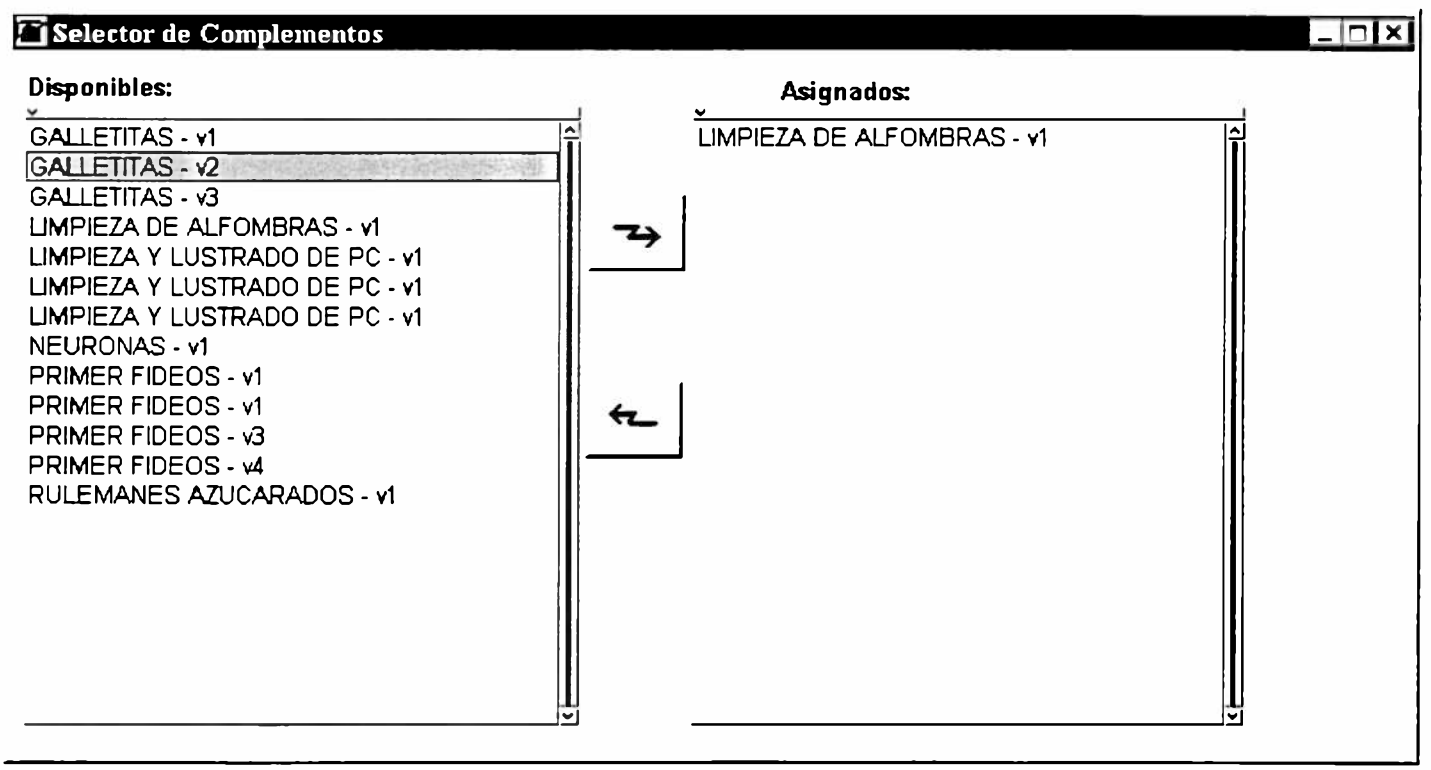




Las ofertas que abarca el concepto editado pueden ser alcanzadas por intermedio del botón del mismo nombre.
También, desde cualquier eslabón de la cadena que represente el concepto editado, todo el historial del concepto puede ser visualizado mediante el botón *Historia*:



Regresando al editor de conceptos, cabe destacar que un concepto puede llevar consigo conceptos complementarios que hacen las veces de "venta enganchada". Los mismo pueden ser alcanzados y modificados (esto último siempre y cuando el concepto esté en edición) por medio del botón *Complementos*:

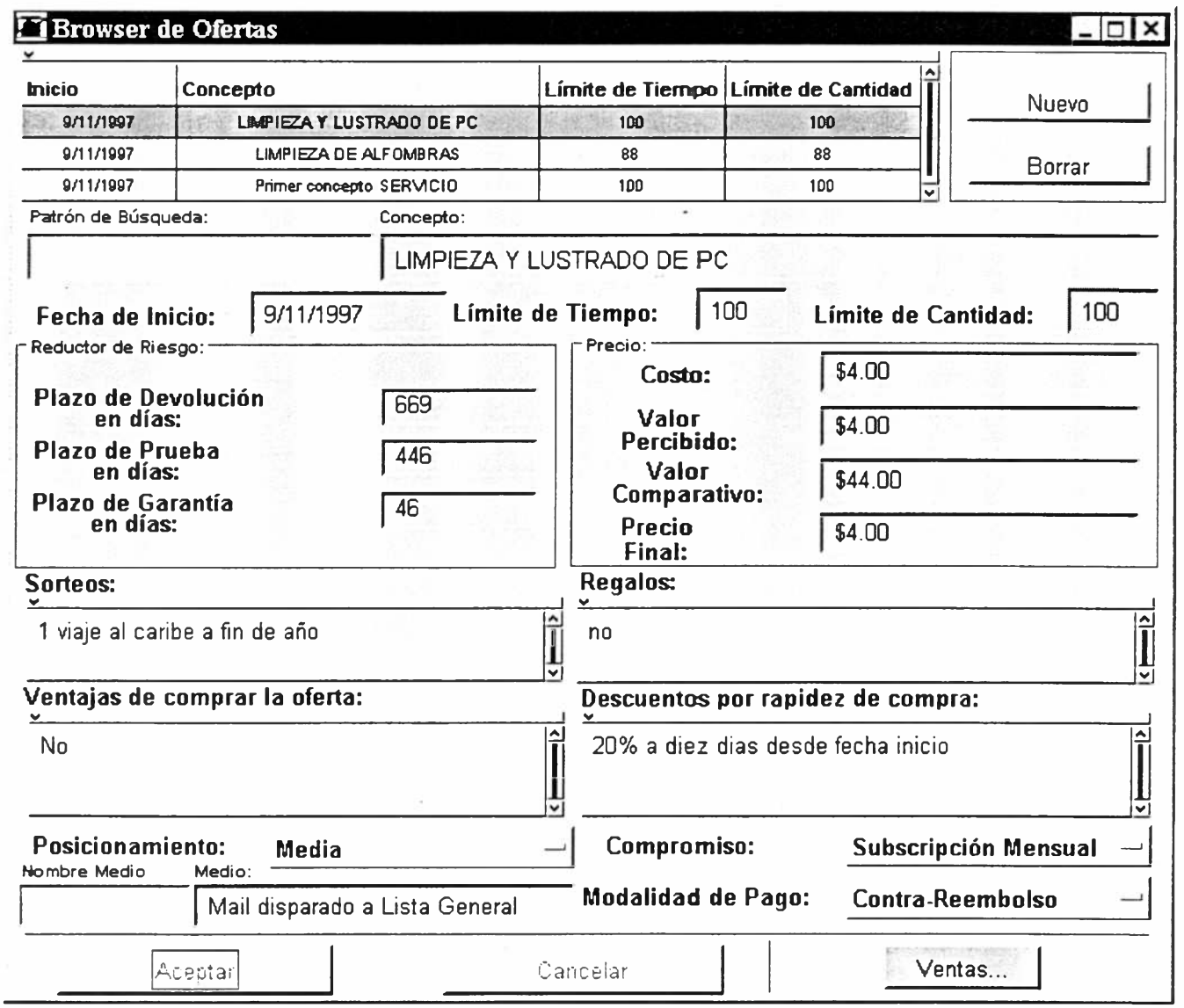


Así como un concepto tiene complementos, el mismo puede formar parte de los complementos de otros conceptos. Esto puede ser identificado por medio del botón *Complemento de*:


Viewer de Complementos
Complemento
LIMPIEZA DE ALFOMBRAS - v1
NEURONAS - v1

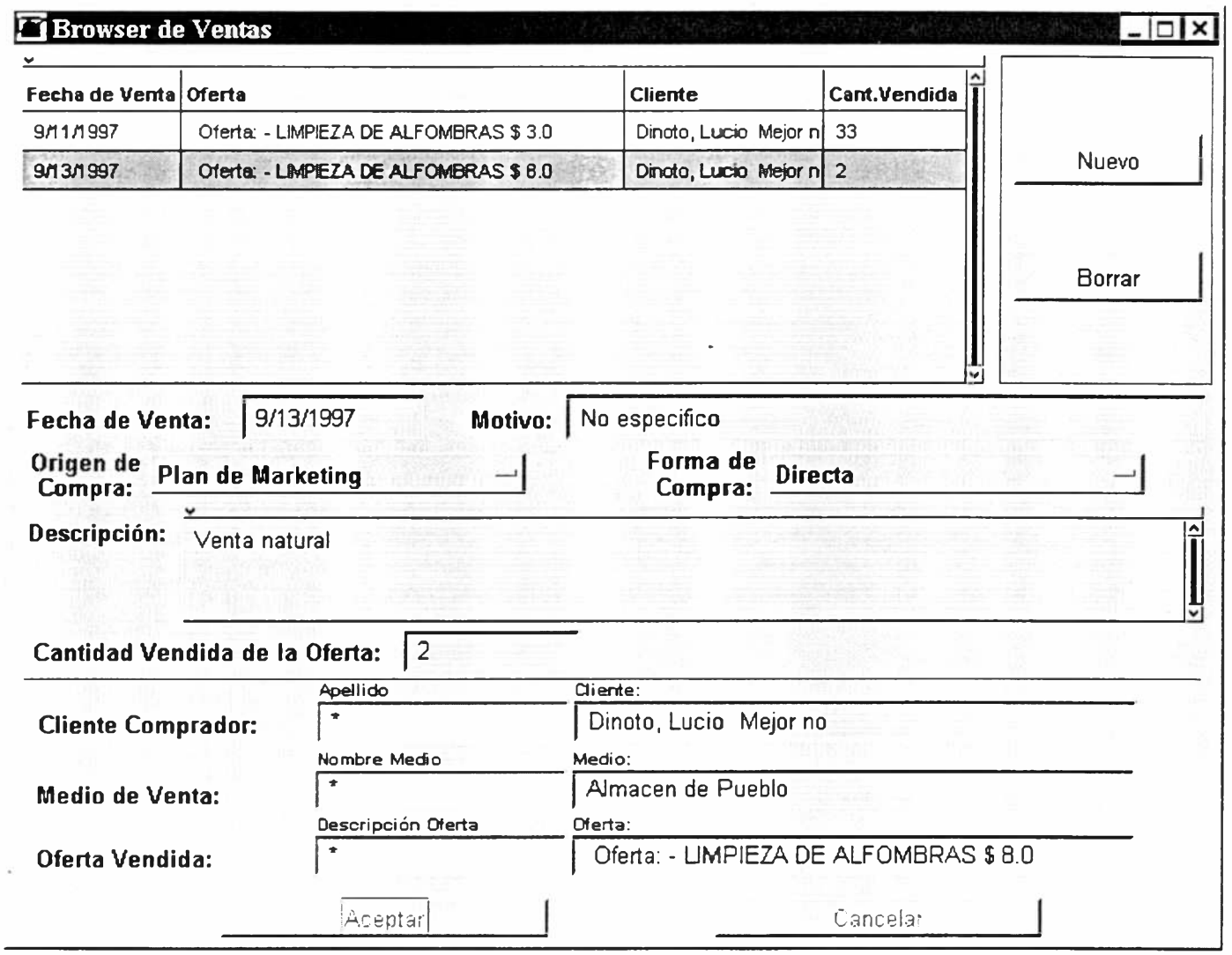
1.8. *Creado de Ofertas sobre conceptos. Precio y medio asociado.*

Desde cada plan de marketing (ver 1.6), es posible determinar las ofertas asociadas con el propio plan. Notar aquí que la forma de seleccionar tanto el concepto (ver 1.7) como el medio, debe hacerse en la parte entitulada *Patrón de Búsqueda*. En tal sector basta con utilizar una combinación de asteriscos (*) y letras para que DIMBO le retorne una lista de posibles opciones matcheadas con el patrón ingresado. Luego, una vez seleccionada del menú ofrecido una de esas opciones, el campo en cuestión es completado automáticamente. Finalmente, desde este browser pueden accederse a las ventas asociadas con la oferta seleccionada en el browser.



1.9 *Ventas. Incorporación de ventas sobre ofertas particulares.*

Una venta se realiza sobre una oferta en particular (recordar que las ofertas se definen en el contexto de un plan de marketing en particular). Un browser especializado en las ventas es utilizado para asentar las mismas:

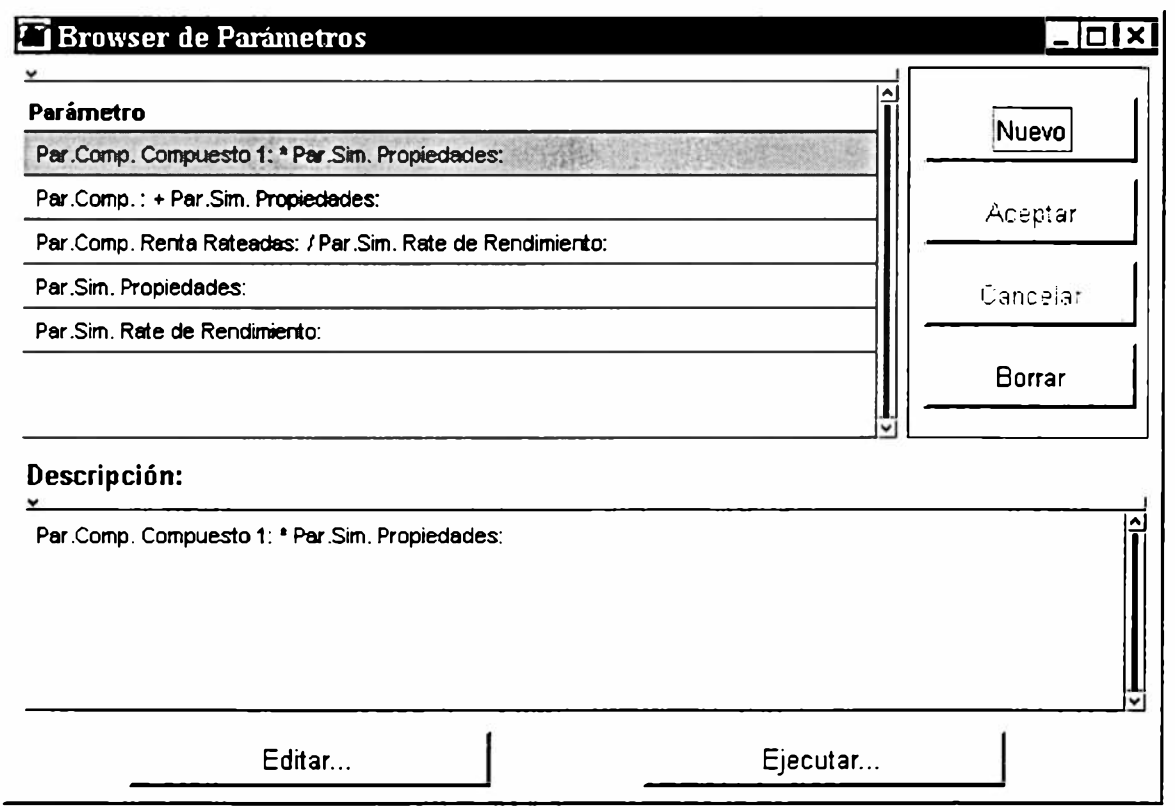


La forma de completar los aspectos de Cliente, Medio y Oferta es similar a la descripta en el punto 1.8. Tales aspectos deben ser indefectiblemente completados para poder dar de alta una venta.

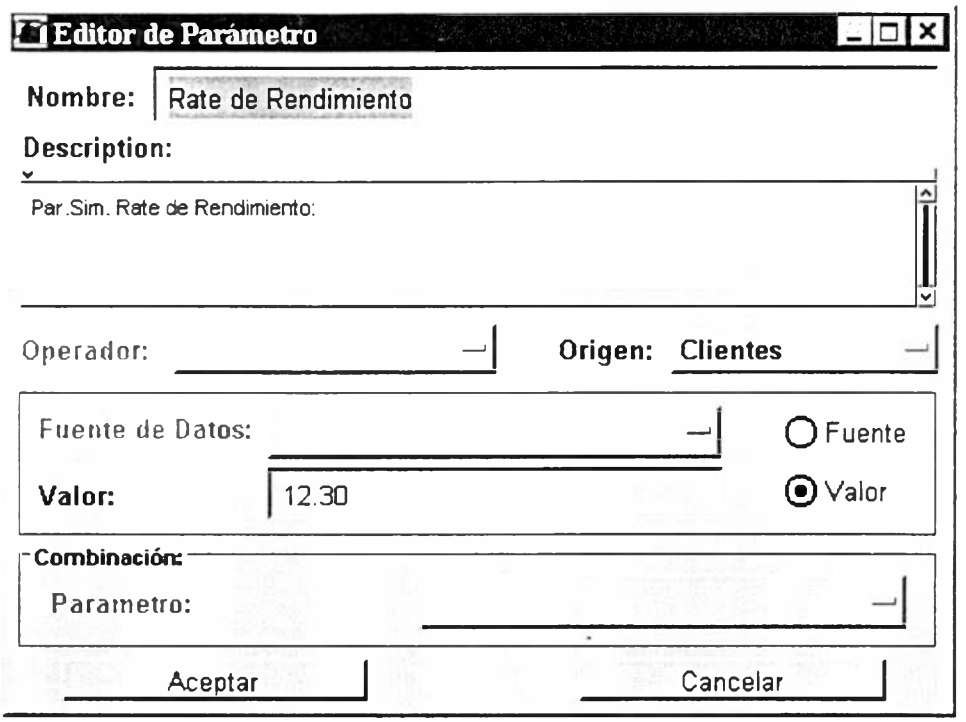
1.10. ***Parametrización de Clientes, Conceptos, Ventas y Ofertas. Fórmulas de parametrización. Obtención y composición de fórmulas.***

La parametrización de clientes es otra de las virtudes que ofrece DIMBO al permitir al manager poder combinar aspectos numericos de sus fuentes junto con operadores matemáticos simples para satisfacer sus deseos de valorizaciones de clientes. Esto significa que en formas únicamente limitadas al incentivo matemático del manager, este último puede obtener una valoración de cada uno de sus clientes en forma dinámica.
Cualquiera sea la fórmula o combinación de fórmulas que se apliquen, estas se dispararán sobre todos los clientes de la aplicación y asignará a cada cliente un único valor (entero o real). El usuario dará luego interpretación de mercado a esa valoración obtenida.
Los clientes pueden ser valorizados por varias fuentes de información que lo referencien, más allá de sus propios aspectos numéricos tales como edad, número de hijos, número de propiedades. Así entonces, intervienen en esta valoración valores tales como cantidad de ventas realizadas sobre el cliente, días desde la última compra, etc. Una vez más, estas pueden ser combinadas entre ellas y con otras constantes numéricas para obtener así las fórmulas deseadas. Estas fórmulas, son conservadas en la aplicación para ser reusadas o bien combinadas nuevamente para obtener nuevas parametrizaciones.

El browser de parámetros existentes en la aplicación se alcanza en la siguiente interface:

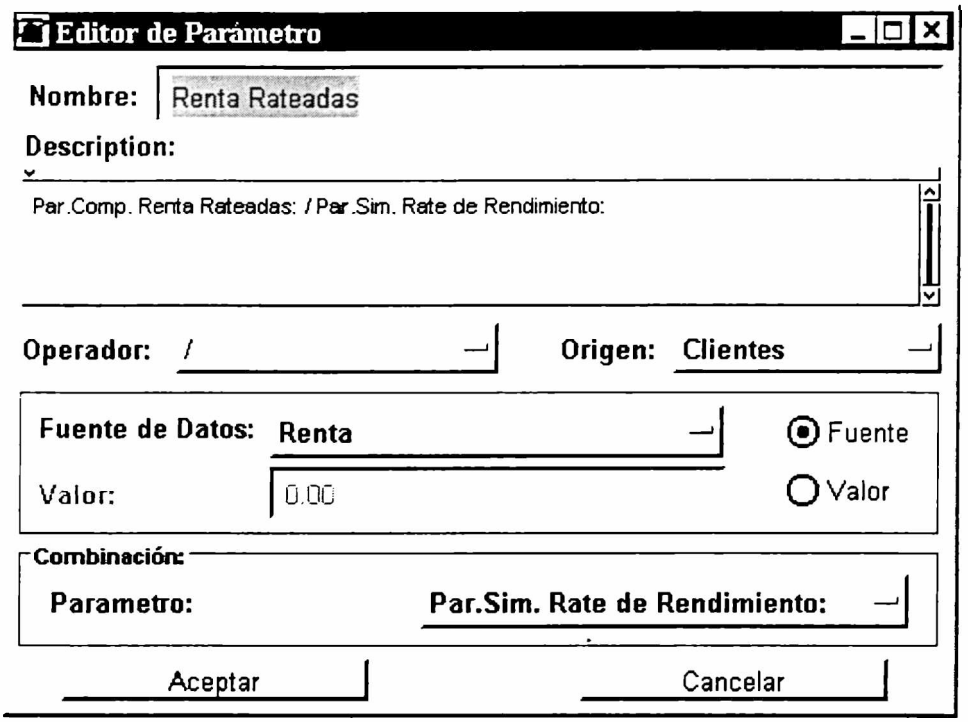


El parámetro seleccionado puede ser editado con el botón *Editar* accediendo a la siguiente interface:



La estructura de la interface anterior indica un tipo de parámetro de los denominados simple. Al momento de crear un parámetro en el browser anterior por intermedio del botón *Nuevo* , DIMBO le preguntará al usuario si desea un parametro *simple* o *compuesto*. Los simples son aquellos que representan o un valor constante (como el ejemplo arriba) o bien un valor desde una fuente de datos. La fuente de datos se selecciona desde el botón de *Origen* (Clientes en el ejemplo). Estos origenes pueden ser tanto clientes, como ventas, compras y ofertas.
En este editor puede elegirse uno de dos opciones: valor o fuente. En el primer caso, se espera un valor numérico que hace las veces de constante para representar el parámetro. En el segundo, según el origen elegido, se deberá seleccionar un aspecto del propio origen, predefinido de antemano como parametrizable.

El otro caso en cuestión trata los parámetros compuestos, donde la interface habilita otras opciones como las que muestra la vista siguiente:



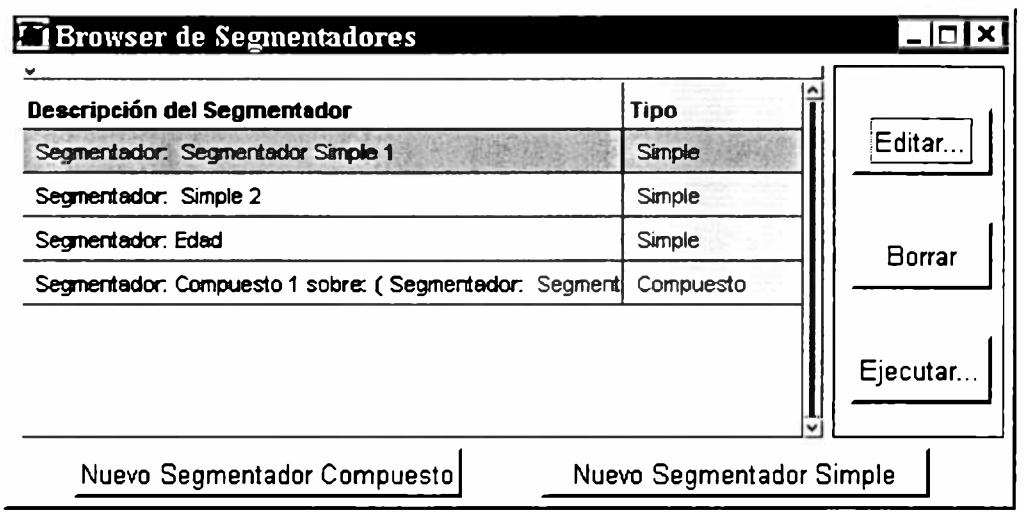
Notar que en la interface de arriba está seleccionado un origen (clientes) y la fuente de ese origen (renta). Esto constituye un parámetro que será matemáticamente combinado con otro parámetro preexistente (y alcanzable desde el browser de parámetros descripto más arriba) vía el operador matemático de división (/). Es decir, que este parámetro evaluará los clientes asignándoles un valor numérico fruto de dividir la renta de los clientes por el parámetro simple llamado *Rate de rendimiento*.

Observar también que el browser presenta una descripción que no es editable y que simboliza la descripción del parámetro editado.

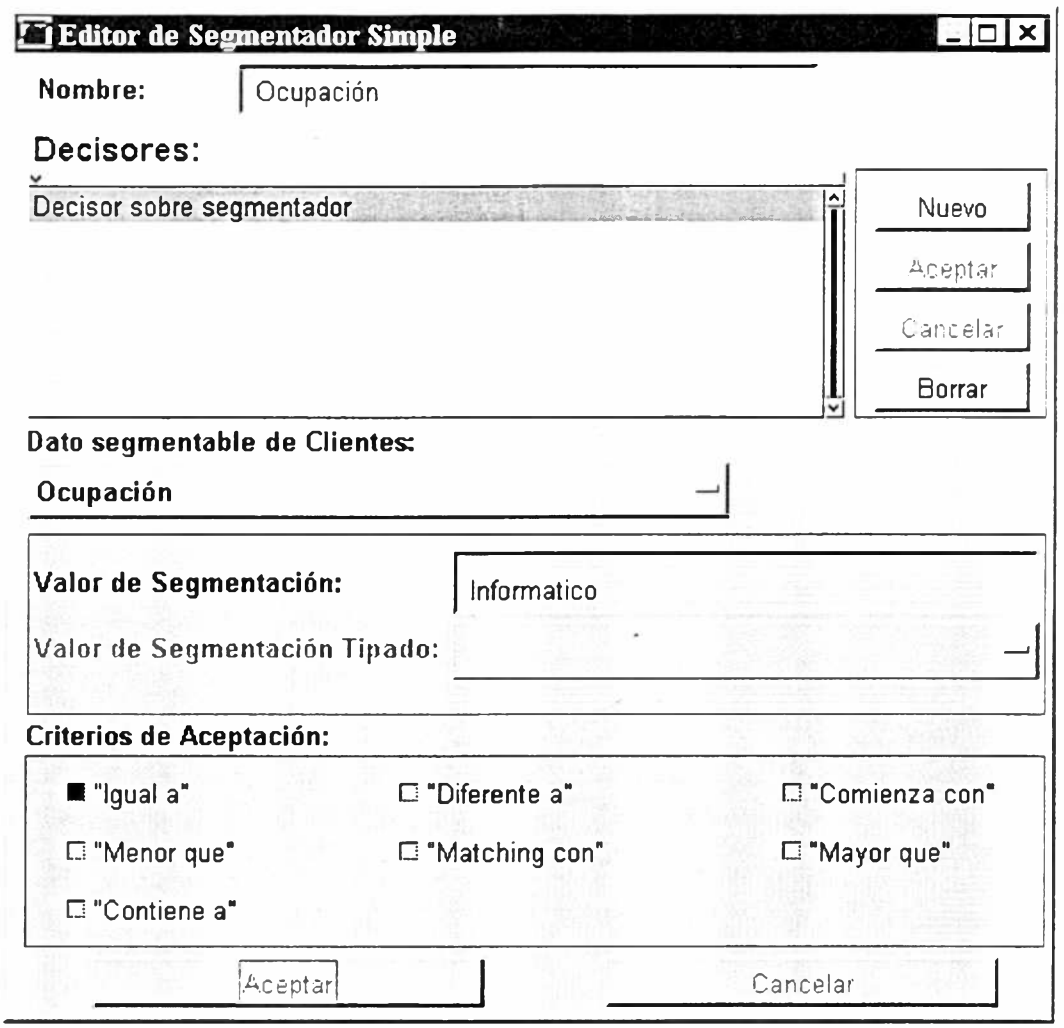
Por último -y regresando al browser de parámetros- la opción *Ejecutar* dispara el parámetro seleccionado retornando el listado de los clientes ordenados de mayor a menor según el resultado de la propia valoración.

1.11. ***Segmentación de Clientes. Obtención y composición de aspectos segmentables. Instanciación de segmentadores de mercado.***

La segmentación de clientes brinda al manager la posibilidad de disparar flujos comerciales sobre conjuntos selectos de clientes reunidos bajo premisas oriundas en las propias necesidades que los flujos satisfagan. De esta manera el factor de riesgo disminuye incrementando la posibilidad de éxito sustentada en el hecho de dirigir la oferta con la mayor fuerza posible sobre el más fuerte de los potenciales grupos consumidores.
DIMBO permite la segmentación de clientes en base a la especificación de uno o varios segmentadores combinados que se aplican en conjunto sobre el universo de clientes.
Existen de antemano un conjunto amplio de aspectos segmentables que llevan consigo el valor deseado por parte del manager y un conjunto de criterios a aplicar a la hora de comparar los datos reales de la base de información con la opción de segmentación.
El browser de segmentadores de DIMBO muestra una interface como la siguiente:

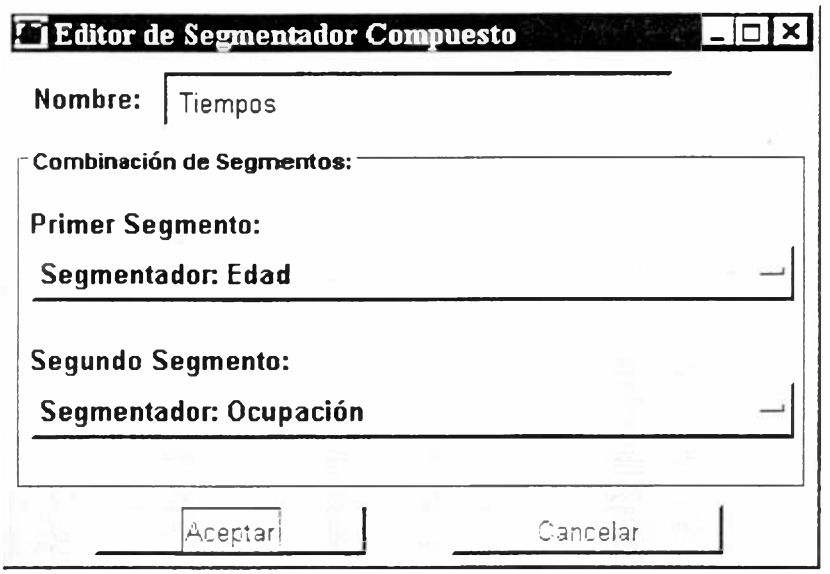


Observar que los segmentadores pueden ser simples o compuestos. En el primer caso, contendrá solo un aspecto segmentable de cliente con su respectivo valor. En el segundo combinará dos segmentadores preexistentes en uno común. Estos segmentadores son persistentes en la aplicación y pueden ser reusados y recombinados libremente.
El botón *Editar* abrirá la ventana de edición del segmentador, dependiendo si este es simple o compuesto:

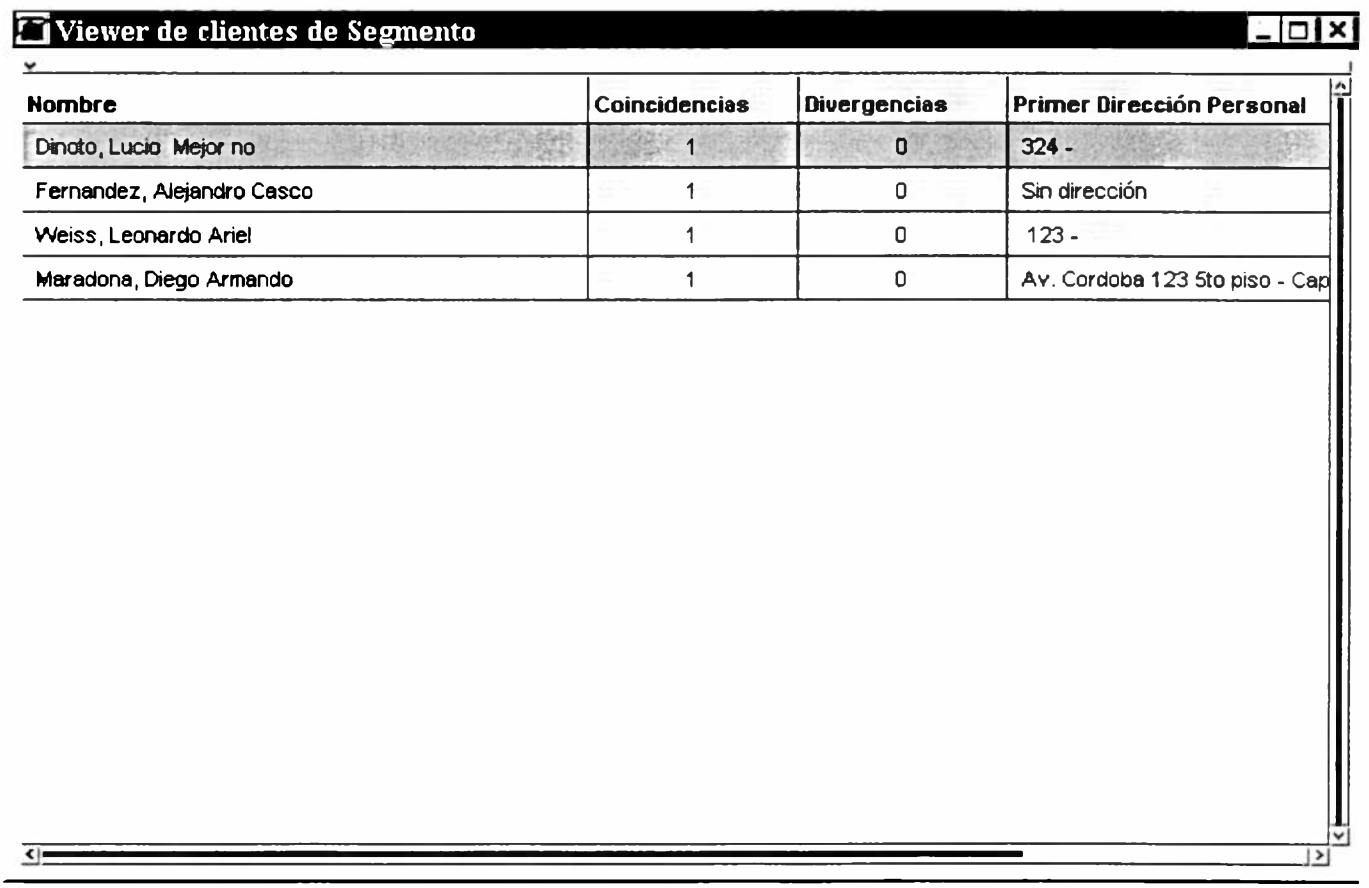


Es importante destacar que algunos segmentadores tienen su origen tipado, es decir, sus posibles valores resultan de los valores predefinidos en su plantilla. Así, por ejemplo, si se segmenta por sexo, el menu-botón disponible en el editor anterior sería el denominado *Valor de Segmentación Tipado* donde las opciones de sexo previamente definidas aparecerán como únicas opciones posibles. Por otro lado, los criterios pueden ser seleccionados en forma múltiple, exigiéndose al menos uno por vez.
Cada segmentador, como se observa en el browser, contiene uno o más decisores. Estos decisores conforman el conjunto de aspectos sobre los cuales el segmento reflejará el matching.

En el caso de segmentador compuesto, la interface luce como la siguiente:



Todo cliente que está siendo evaluado por un segmentador será incluido en el segmento a generar con un valor de aciertos. Por cada decisor que el dato segmentable y el dato se segmentación se cumplan bajo el o los criterios seleccionados, DIMBO incrementará en uno el número de matching de aciertos del cliente. El segmento entonces es obtenido por el cliente ordenado de mayor a menor y es generado luego de presionar el botón *Ejecutar* del browser de segmentadores y acceder al browser de segmentos que se indica a continuación:



Nombre	Coincidencias	Divergencias	Primer Dirección Personal
Dinoto, Lucio Mejor no	1	0	324 -
Fernandez, Alejandro Casco	1	0	Sin dirección
Weiss, Leonardo Ariel	1	0	123 -
Maradona, Diego Armando	1	0	Av. Cordoba 123 5to piso - Cap

Plataforma Técnica y Extensiones

Detalles Técnicos

Los requerimientos mínimos para obtener una aceptable performance de la aplicación deberían sustentarse al menos en un procesador 486DX4 con 16 megabytes de memoria, aunque lo ideal se acercaría a una Pentium con 32 megabytes de memoria.
DIMBO cuenta con la particularidad de poder correrse en un ambiente centralizado. Esto es, no es necesario un server exclusivo para el soporte relacional de la aplicación. De esta manera, un mismo procesador bajo windows 95/NT soporta tanto el motor relacional MS Access como la aplicación Visual Works. Este faceta se logra vía la conexión ODBC que incorpora Microsoft Access a partir de la versión del mismo para Windows 95.
Cualquier otro motor relacional, de mayor o menor envergadura, podría haber sido elegido para este desarrollo. Es más, como Visual Works se abstrae del motor, podría migrarse la base de datos actual a otro motor (DB2, SQL server y hasta Xbase), que DIMBO seguiría funcionando normalmente. Sin embargo, se eligió Ms Access debido a su simplicidad y a ser un producto altamente conocido.

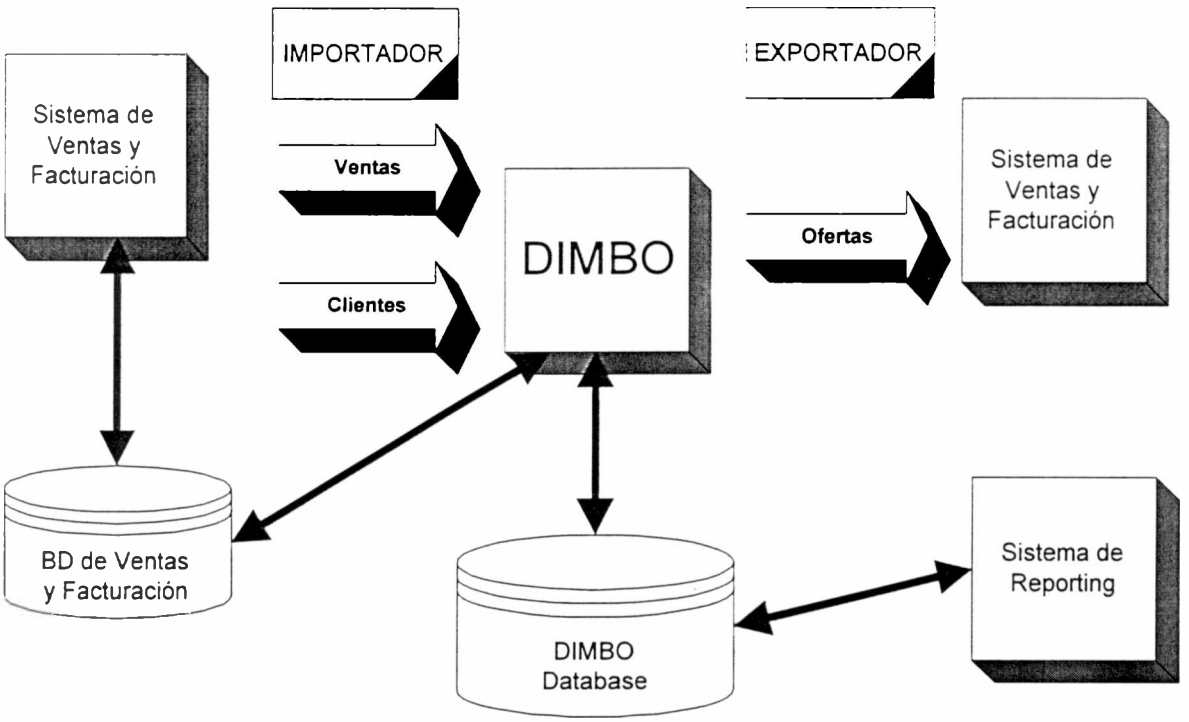
Optimizaciones y mejora de performance

Ninguna de las prestaciones que brinda DIMBO están sujetas a Ms Access ni a cualquier otro soporte relacional. Esto puede debilitar en ciertos aspectos la performance de la aplicación. Puntualmente, la mayoría de las bases de datos relacionales proveen cualidades tales como *store procedures*, que no son otra cosa que código SQL precompilado, lo cual acelera notablemente la velocidad de ejecución del código comparado a una corrida de SQL standard. Esta característica se asemeja a los *módulos* que provee Ms Access, los cuales también pueden incluir código *Visual Basic*. Tal capacidad puede resultar útil, por ejemplo, a la hora de recuperar objetos complejos (cuando se muestran en las rows de un dataset una combinación o join de objetos del modelo). Si este tipo de interface es constantemente requerida por la aplicación, la performance puede verse degradada, con lo cual las implementación de store procedures acelera notablemente los tiempos de respuesta.
Otro característica son los *triggers* de insert y de update de las tablas. Los triggers son segmentos de código SQL que se disparan posteriormente a un update o a un insert -según se defina- en la tabla en cuestión. De esa manera, por ejemplo, se garantizan accionares sobre otros datos de la aplicación, manteniendo rules predefinidas con los datos del sistema. Esto tiene una ventaja que radica, una vez más, en la performance. Pero su gran desventaja yace en el hecho de que se pierde la localizabilidad que intentamos ganar con los modelos de objetos, ya que cierto tipo de responsabilidades que deberían ocurrir en el dominio del modelo de objetos, escapan del mismo para ajustarse en la propia base de datos. Es decir, lo malo de esto es que de repente pueden incorporarse business rules en los triggers en vez de hacerlo en el modelo, donde realmente debería estar. Por ejemplo, DIMBO maneja

versiones de conceptos de los cuales el modelo permite que una versión de la línea de evolución sea la default. Esta característica viene indicada directamente por una columna en la tabla VersionObject. Si una nueva versión comienza a ser la default, el modelo debería localizar la versión actualmente calificada como default y quitarle esa propiedad. Pero otra alternativa de solución podría ser que un trigger se encargara de realizar esto al momento de recibir un update o un insert con la columna default seteada. Esta solución es simple, efectiva y eficiente, pero escapa totalmente del modelo de objetos y tiene consecuencias directas sobre el mismo, por lo cual, debería ser desestimada. Sin embargo, un uso conveniente de los triggers podría ser acorde, por ejemplo, para la recolección estadística de datos que se generan en DIMBO, donde tales estadísticas podrían a ser usadas, por ejemplo, por otro sistema (Excel, Word, Sistemas de Reporte, etc...) para generar gráficos.

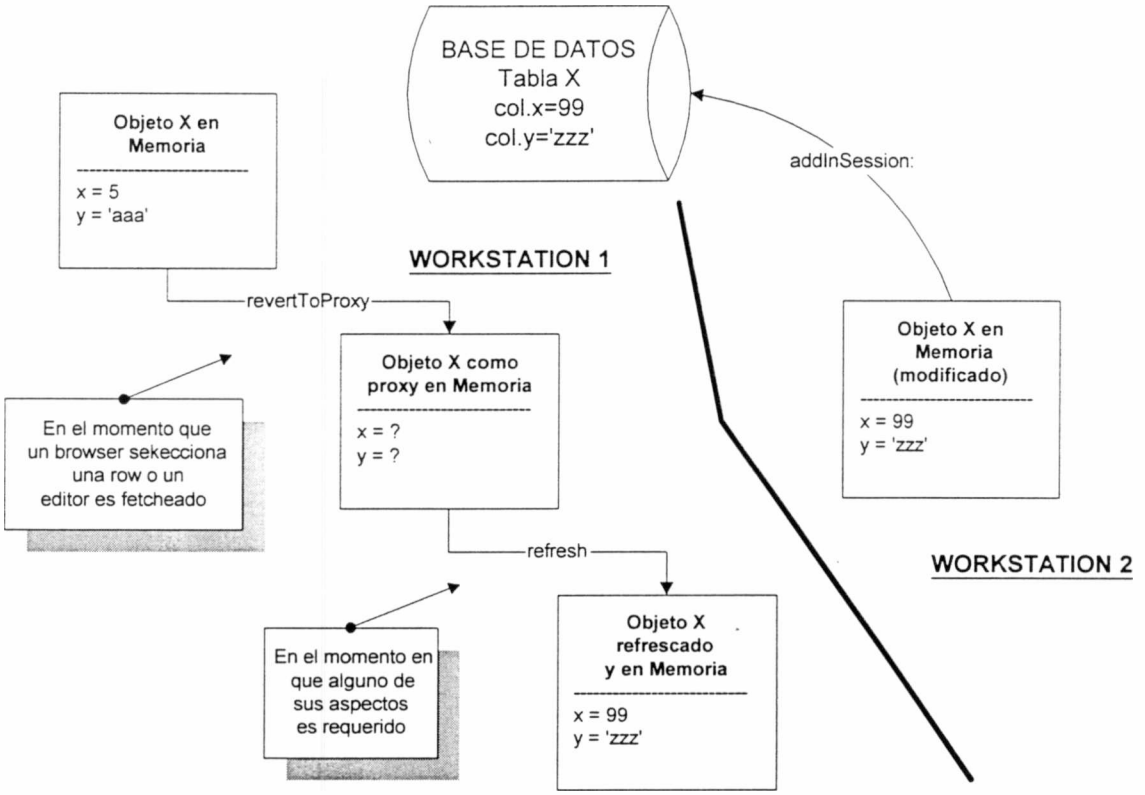
Extensiones

Hay dos puntos que DIMBO no considera y que serían determinantes para su consumo en el mercado : comunicación con otros sistemas y reportes. En ese inciso se intenta dar una pauta para atacar ambos puntos.
Para obtener un reporting efectivo y de alta calidad habría que : o desarrollar todo un modelo genérico para soportar reporting en Visual Works o utilizar herramientas específicamente desarrolladas para tal efecto.
Considerando que los datos de la aplicación viven en un modelo relacional que se accede desde cualquier utilidad vía SQL standard, la segunda solución parece ser la más conveniente. Y allí radica la ventaja (la única ?) de tener el modelo de objetos mapeado a un modelo pseudo-relacional, ya que una infinidad de utilidades (especialmente de reporting) puede ser utilizada sin problema alguno.
Por otro lado una aplicación como DIMBO "alone in the business" puede llegar a ser una molestia más que una utilidad. A continuación se brinda un diagrama genérico de los datos que DIMBO debería compartir, exportar e importar de otras aplicaciones. Otra vez, como los datos se soportan en esquema relacional, la convivencia e interacción con otras aplicaciones, en principio, no es un tema de preocupación mayor debido a la compatibilidad de las fuentes. La única extensión a realizarse sobre el modelo de objetos es la capacidad de importar y exportar objetos vía tablas relacionales.



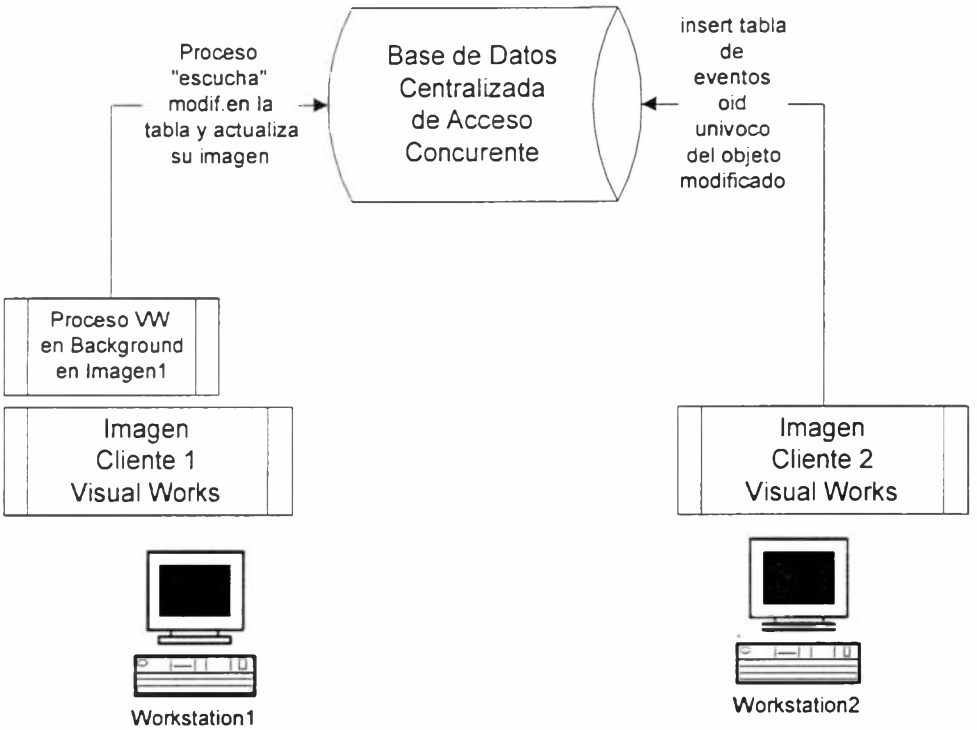
Extensión Ambiente Distribuido

Extender lo existente a un ambiente distribuido de acceso concurrente no sería un inconveniente para DIMBO, que en principio necesitaría solo una extensión en su ambiente la cual se detalla más adelante.
Por un ambiente distribuido se supone que la base de datos se encuentra atendida por un server especial, eventualmente dedicado, que permite acceso concurrente desde varias workstations. Esta es una virtud del propio ODBC. Y los resguardos de un acceso concurrente están salvados y soportados por el Object Lens (por ejemplo, en el lockeo antes de disparar un addInSession : , protocolo que realiza un update o insert del objeto en su respectiva tabla). Object Lens es el framework que provee Smaltalk para soporte relacional.
La extensión que se mencionaba anteriormente consiste en ser capaz, desde el modelo, de ver siempre lo último que existe en la base de datos. Por ejemplo, si un usuario desde una workstation abre el browser de conceptos y otro usuario desde otra máquina da un alta de un nuevo concepto (o una modificación), el primero debería ser capaz de ver dicha modificación, al menos, en el momento de su edición directa donde una modificación podría intentar hacerse sobre información antigua. Este "refresh" del objeto en memoria con los nuevos datos de su tabla podría dispararse cada vez que se seleccione una nueva row (para el caso de browsers) o el value holder que sostiene el objeto editado (para el caso de editores simples). La manera de refrescar el objeto es volverlo a proxy, con lo cual Lens se encargara de desproxiarlo al próximo acceso, realizando de este modo el refresh.



Lo anterior lleva consigo una pequeña carga y baja de performance, ya que ante cada selección o edición de un objeto habría que llevarlo a proxy para su posterior refresh.

Otra alternativa es lograr una comunicación entre las diferentes imágenes clientes de la aplicación.
Esto podría lograrse usando la propia base de datos, indicando en una tabla el OID del objeto modificado en cada estación. Así, en cada imagen un proceso background podría estar corriendo analizando las altas de esta tabla y convirtiendo a proxy cada instancia de las mencionadas en la tabla que este cargada en la imagen local.



Mejoras

Algunas características que han sido acotadas en esta versión de DIMBO son, por un lado, la de permitir que los datos tipados (como beneficios esperados, etc..) permitan múltiple valoración. Esto es, por cada dato que se clasifica tipado, permitir asignar un conjunto de valores en vez de solo uno, como es en la actualidad. Notar que la característica multivaluada está soportada en el modelo, ya sea tanto en los TypeClassContainer como en las TypeInstance. Las cuales permiten múltples opciones en su protocolo. Esto significa, que la única extensión necesaria yace en la interface, donde habría que generar un chooser genérico (aspect subcanvas) para los TypeInstances que permita múltiple selección y pegarlo luego en todas las interfaces que referencien dicho tipo de campo. Como la multiplicidad de un tipo puede cambiar dinamicamente (basta con que el usuario modifique una instancia de TypeClassContainer en el browser de planillas), el uso de del chooser simple o multiple, debería ajustarse dinámicamente.

El otro punto a mejorar de la actual implementación de DIMBO se funda en la actual creación de los ShoppedData. Las instancias de las subclases de esta clase (para Client, Sale, Concept, Supply) son utilizadas como los buceadores dentro del modelo a la hora de trabajar con valores parametrizables o segmentables. Actaalmente existen sendos builders (shoppers) que harcodean cada uno de los aspectos "buceables" y van creando y persistiendo las instancias de los ShoppedData. Este código se disparó una sola vez, persistió los shopped data y no vuelve a ejecutarse. Obviamente, el propio developer se

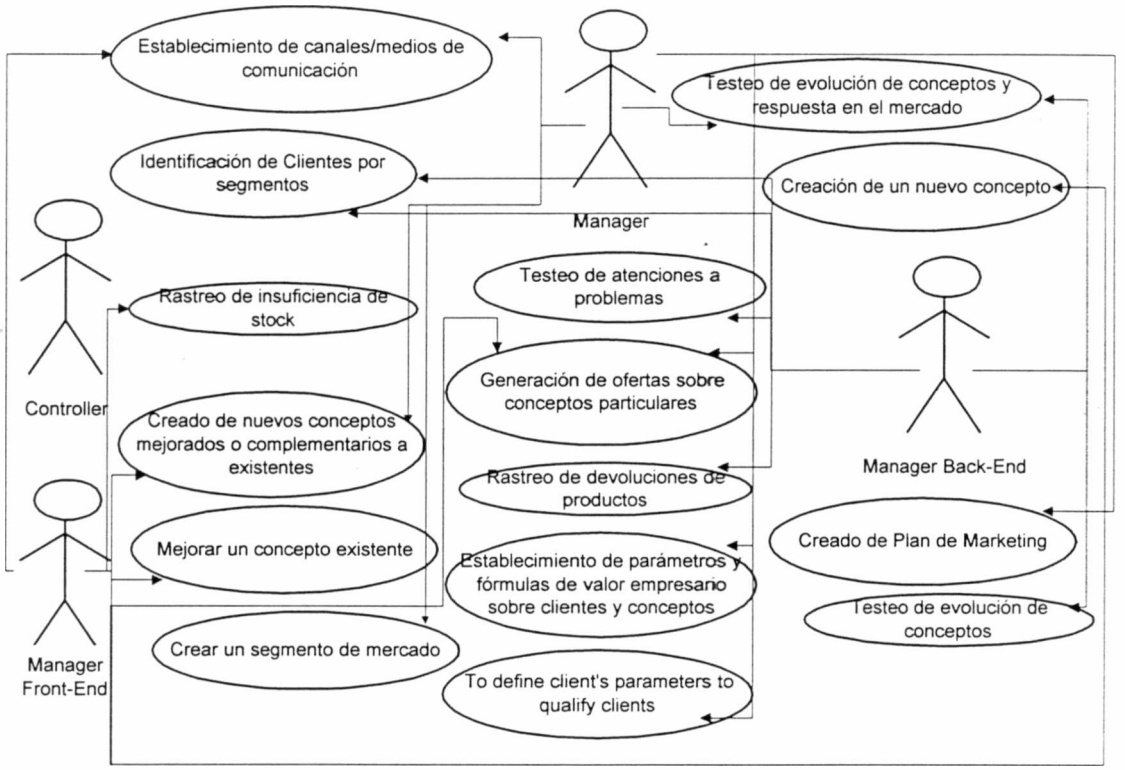
encargo de hacer esto y dicha tarea no compete ni competerá nunca al usuario o los usuarios finales.
La primer y principal desventaja de este approach es que si cambia el comportamiento de algunos de los objetos shoppeados, como ser Client, Sale, Supply o Concept, se deberá modificar el código de los shoppers, identificar cuales shopped datas se ven afectados, modificarlos y persistirlos.
Pero el actual modelo corre con la ventaja que los shoppers pueden ser despegados del mismo y poner cualquier otra cosa en su lugar que cumpla al menos su funcionalidad y nadie se verá afectado en lo más mínimo.
Una solución práctica y elegante para este tipo de problemas consiste en armar un browser de shopped datas para que puedan ser editados, modificados y borrados a antojo del developer, en forma dinámica y sin tener que modificar código. En realidad, como se generan datos para segmentación y calificación, deberían existir dos tipos diferentes de browser o bien discriminar las altas indicando cual será su utilización. Dichos browsers deben permitir seleccionar el source object desde donde se esta "buceando" y permitir que el developer edite el código bloque para acceder al dato que contiene el source object.
Claro está que este tipo de prestaciones que brindaría DIMBO son solo a nivel developer y que si bien pueden formar parte de la propia aplicación, tales browsers no le competen al usuario y por ende no debería accederlos bajo ninguna circunstancia.

Otras dos características menores que surgieron en el proceso de testing fueron, por un lado la posibilidad de implementar algún mecanismo de timing o scheduler de manera tal que se avise al usuario de las actividades que están pendiente o encoladas y que la fecha tope de vencimiento ha llegado; y por otro lado, la posibilidad de tener roles múltiples para los usuarios.

Procesos de Análisis, Diseño e Implementación

Procesos de Análisis, Diseño e Implementación :

Todo el proceso de ingeniería de la aplicación fue conducido por OOSE (Object Oriented Software Engenieering - Ivar Jacobson 1992). La captura funcional de requerimientos así como la definición de los límites de la aplicación fueron delineados mediante los use cases. A continuación se presenta el modelo de use cases reducido que se encuentra ampliamente detallado en el apéndice A. Este paso creo las bases para todas las actividades que se desarrollaron en el proceso que conforma la ingeniería de software de la aplicación.



Por otro lado, el proceso de análisis cuenta con dos fases bien distinguidas : los modelos de negocios y los modelos de objetos. Los primeros se realizaron en conjunto con el manager o experto del dominio, sirviendo de base para construir los segundos. Los modelos de negocios se abstraen de la implementación y tan solo especifican en términos de objetos como funciona el negocio en la realidad. Los segundos, incorporan los detalles relacionados con sistemas informáticos. Los mismos pueden ser analizados en el apéndice B.

Por último, se brinda un modelo de diseño-implementativo que resume los aspectos fundamentales del diseño de objetos para DIMBO. Los mismos han sido descriptos con sus atributos, relaciones y responsabilidades básicas y pueden hallarse en el apéndice C.

Mapeo de Objetos a Tablas

A continuación se describen todas las políticas utilizadas para el desarrollo de la tesis a la hora de mapear el modelo de objetos a un modelo relacional. Cabe acotar que bajo, ningún punto de vista, la consistencia del mapeo está sujeta a un determinado manejador de base de datos.
Si bien las políticas empleadas son en general aplicables a cualquier mapeo, existen otras muy particulares que se han descubierto en la implementación de esta tesis y están sujetas en cierto modo al lenguaje que se esta usando y al mapeo standard que provee el propio Visual Works, es decir, ObjectLens. Todas las apreciaciones de performance, características, peculiaridades y recomendaciones se enfocan en el caso práctico de utilización de Lens-Visual Works, lo que no quita que las mismas sean aplicadas sobre otra tecnología.

BASES

¿Desde dónde empezar?

Si no se tiene un legacy system, como se supone en este artículo, siempre ,y bajo ninguna excepción, empezar modelando los objetos. Jamás al revés, es decir, partiendo desde un modelo relacional. De ese modo aseguramos al menos que de uno de los lados, el que nos interesa, es puro. ¿Pero hasta que punto es puro el modelo de objetos? Es cierto que tener persistencia tiene su costo : el modelo de objetos es el encargado de pagarla. Pero siendo cuidadoso y ocultando el comportamiento agregado para persistencia puede evitarse el engorrosamiento de la interface de los objetos.
De cualquier manera, el modelo de objetos persistentes no será totalmente reusable para un entorno ideal donde la preocupación por la persistencia es nula (¿Lo será?, ya que aún siquiera los manejadores de bases de datos de objetos brindan persistencia a costo cero, es decir, en forma transparente al modelo).
Está claro que el modelo relacional no es un modelo relacional puro, sin embargo, al usar esta plataforma estamos abriendo nuestros objetos (o nuestros datos desde este punto de vista) al amplio dominio de compatibilidad de lo relacional. Aunque esto último, que pregonan muchos autores, desliza la siguiente pregunta : ¿Tiene sentido que otro entorno utilice un modelo relacional que puede llegar a ser ineficiente desde otra óptica que no sea el mapeo a objetos?

Una clase a una o varias tablas

Permanentemente, en el intento de mapear clases a tablas, se busca poder asociar una clase a una tabla. Y basicamente para lograr localizabilidad, modificalidad y fundamentalmente eficiencia, ya que a la hora de comunicarse con un objeto, no es necesario disparar joins sobre varias tablas, sino tan solo recuperar la información desde una de ellas.
Pero lo anterior solamente ocurre si los atributos persistentes del objeto en cuestión contienen datos simples (string, boolean, etc.) que soporte también el RDBM. Pero esto es

un caso muy trivial que no trae complicación alguna. Se desprende, entonces, que una clase conteniendo atributos con colecciones de otros objetos deberá mapear al menos a dos tablas. Y también, que, si un objeto referencia a otro por un atributo persistente, su tabla deberá contener una clave foránea indicando la clave primaria de la tabla del objeto referenciado.

Colecciones de objetos

Retomando la discusión anterior, ¿Por qué al menos dos tablas? : Sencillo, si la colección contiene más de un tipo de objeto, no pueden encajar en una tabla, por tanto se necesitan tantas tablas como tipos diferentes de objetos.

¿Quién es la clave primaria?

Es importante aclarar que toda tabla tiene un atributo **oid** de tipo char (o varchar) que no es otra cosa que la clave primaria de la tabla. Esta característica de unicidad determina un valor diferente que se asigna a cada objeto persistente.
Algunos manejadores de bases de datos proveen esta generación numérica en forma automática por cada tabla en que se lo establezca. Hoy por hoy, la mayoría de los manejadores relacionales soportan esta característica.
Pero un approach alternativo implementado en JPMorgan-Bs.As es el siguiente :
Desde el propio ambiente de desarrollo se maneja una tabla en la propia base relacional donde se deja establecido cada clase persistente y su contador de instancias generadas en la base. De esta manera, y desde el propio modelo de objetos, cuando se da un alta de un objeto en su/s tabla/s , se identifica a cuales tablas se refiere y se obtiene para cada una de estas cual es el próximo número de oid disponible para asentar en la base de datos, incrementándose luego tal valor en la base de datos de las tablas.

Costo a pagar en general

- Los objetos se abren para persistir. Esto es, Lens requiere que se generen accessors y mutators para cada objeto persistente del objeto
- Las variables de instancias de los objetos deben ser definidas en el mismo orden textual que la propia tabla relacioal y obviamente buscar la equivalencia de tipos entre el ambiente de objetos y el relacional (por ejemplo, [string, varchar], [boolean, bit], [integer, int], etc.)

Un objeto persistente referencia a otro objeto persistente

Es un caso de relación uno a uno. Se puede dar por una relación de parte_de o de tiene_a.
Ejemplo :
Un Client tiene una ElectronicalAddress.
Solución con Objetos :
El objeto Client tiene una variable de instancia que referencia a una ElectronicalAddress.
Solución relacional :

Existe una tabla Client que no posee ninguna referencia a la tabla ElectronicalAddress. Sin embargo, esta última tabla posee una columna que referencia a su padre, es decir, a Client. Y esa columna no es más que la clave primaria de Client, o sea la columna oid.
Es importante evitar agrupar todo en una tabla, ya que hace que los cambios en las clases en cuestión sean menos localizables.
Consecuencia en el modelo de objetos :

En el structure type que define Lens para el link entre el objeto y la tabla, debe **tiparse** la variable de instancia que referencia al ElectronicalAddress indicando este último tipo de objeto.

Un objeto persistente referencia a una colección de otros objetos persistentes

Es un caso de relación uno a muchos. La relación se sustenta en el modelo de objetos con una instancia de Collection.
Ejemplo :

Un BuyerClient tiene un conjunto de Purchase que ha realizado a través del tiempo.
Solución con Objetos :

El BuyerClient tiene una variable de instancia que es una colección de objetos Purchase.
Solución relacional :

Estructuralmente similar a la solución uno a uno, pero con otro manejo de la persistencia. La tabla para el BuyerClient no referencia ninguna colección de otros objetos. En tanto, Purchase mantiene una columna que referencia a su padre BuyerClient.
Por ejemplo, dos instancias de Purchase pertenecientes al mismo BuyerClient, serán instancias de la tabla con igual foreign key referenciando al mismo parent.
Consecuencia en el modelo de objetos :

El objeto Purchase debe mantener una variable de instancia para referenciar al padre. Obviamente, esta variable **debe** ser transparente al protocolo del objeto.
Desde Lens, la variable de instancia que referencia la colección debe sostener un IdentitySet y será tipada en el structure type con el tipo de objeto que contiene (Purchase). Cuando una instancia de Purchase modifica su parent (por ejemplo, en la creación de una instancia desde la BD), el padre (BuyerClient) es notificado del cambio y agrega en su IdentitySet de Purchase's la nueva instancia. De cualquier manera, este comportamiento es automáticamente generado por Lens conjuntamente con los accessors.

Un objeto persistente referencia una colección de objetos que pueden ser de diferentes clases

Es también un caso de relación uno a muchos.
Ejemplo :

Un composite puede tener en la colección de sus sucesores tanto objetos base como composites.

Solución con objetos :

En el modelo puro de objetos tal relación se sustenta con una simple collection que sostiene los diferentes objetos.

Solución relacional :

Cada diferente tipo de objeto que sostiene la colección debe ser manejado independientemente. Existirá un tabla para el objeto base que contenga una columna para referenciar al padre del cual él forma parte de su colección. Por otro lado el composite deberá tener otra columna para referenciar a instancias de su misma tabla y otra columna para representar la colección de objetos base que agrupe.

Lo anterior se debe a que Lens necesita conocer de antemano a que tabla se mapea un atributo.

Consecuencias en el modelo de objetos :

Se pierde polimorfismo en la representación interna del objeto. Hay que tener tantas variables de instancias representando colecciones como tipos de objetos diferentes puede sostener el modelo original. Es necesario, entonces, brindar un protocolo transparente de manera tal que los que interactúen con este objeto no noten el detalle implementativo. Tal protocolo debe unificar las collections a una sola, pero considerando un eventual orden que puede haber entre todos los elementos. Si el orden es significativo, puede agregarse cada elemento a las respectivas colecciones junto a un índice que se va incrementando a medida que las diferentes collections van recibiendo nuevos elementos.

A igual que el caso anterior, es necesario tiparse las colecciones con sus respectivos valores (Base y Composite, en este caso).

Ciertos objetos referencian a una colección de otros objetos y estos referencia a los primeros

Es una relación n a n.

Ejemplo :

Una Persona puede referenciar varias Direcciones y las Direcciones pueden pertenecer a varias Personas.

Solución con Objetos :

Tan solo una variable de instancia en Persona que referencia a una collection de Dirección. Lo mismo se repite desde Dirección hacia Persona.

Solución con relacional :

Ambas tablas sostienen una columna adicional para referenciar al parent. Similar al caso 1 a n.

Consecuencia en el modelo de objetos :

No agrega consecuencias a las ya planteadas.

Una jerarquía debe ser persistente y la clase abstracta posee atributos comunes

Ejemplo :

Una Address puede ser una ElectronicalAddress o una PhysicalAddress.

Solución con Objetos :

Se desprende del planteo.

Solución con relacional :

Una posibilidad es representar a la clase abstracta en una tabla con su propio oid, y las dos restantes tablas representando a las clases concretas con un atributo foráneo a una row particular de la tabla abstracta. Este approach solo es recomendado cuando la clase abstracta es muy dinámica, es decir, sufre cambios prologadamente. La justificación de esta sugerencia radica en evitar joins a la hora de recuperar objetos desde las tablas, lo que hace más eficiente la interacción del mundo de objetos con el mundo relacional. Por otro lado, utilizando el approach mencionado, los posibles y frecuentes cambios de la clase abstracta, solo afectan a una sola tabla y no a las concretas.

Entonces, la solución que se plantea es definir tantas tablas como clases concretas persistentes existen y agregar a esas tablas columnas para representar los atributos de la clase abstracta

Consecuencias en el modelo de objetos :

El tipado de las variables se realiza en el structure type de las clases concretas y no en las abstractas.

Un objeto persistente tiene una colección ordenada de otros objetos

Ejemplo :

Un objeto mantiene una lista muy grande objetos. Dicha lista la mantiene ordenada debido a constantes consultas sobre ella. De no estar ordenada, las consultas serían ineficientes.

Solución con Objetos :

Usar una SortedCollection.

Solución con relacional :

Similar a la solución uno a muchos. Pero en el motor relacional crear un índice para la tabla sosteniendo la colección construido de manera similar a como se implementó el sort block en el modelo de objetos. Asignar este índice como el primario. De este modo, cuando se recupera el objeto persistente, el identity set referenciando a la colección estará ordenado implicitamente de la manera buscada.

La otra alternativa es implementar un accesor particular que toma el identity set armado desde el modelo relacional y retomarlo ordenado, wrapeando de esta manera el acceso al identity set. El problema de este último approach es el costo de ordenar grandes listas. Por eso, solo se recomienda en casos de listas pequeñas

Consecuencias en el modelo de objetos :

Sin consecuencias diferentes a las planteadas.

Un objeto persistente tiene un atributo que es un block

Ejemplo :

Un bussines object tiene una rule que se evalúa con parámetros externos.

Solución con Objetos :

Con un Block se representa el comportamiento de la rule y se le envía value : con el argumento mencionado

Solución con relacional :

En la tabla que representa la clase que tiene el bloque se almacena un atributo del tipo char o varchar. Aquí se persiste el sourceCode del bloque. Esto es, la clase tiene en si una variable de instancia que sostiene el source de un bloque. Cuando un accessor se dispara sobre una instancia para obtener el bloque de la misma, el string se convierte en bloque mediante el mensaje ##########.

Consecuencias en el modelo de objetos :

El bloque debe ser regenerado cada vez que se accede al mismo. Esto puede evitarse con un nuevo atributo en la clase que haga caching del bloque. De cualquier manera, el modelo sufre la consecuencia de mantener información adicional.

Una clase abstracta persistente define una colección de objetos persistente. Por ende, las subclases heredan este atributo y las instancias no son compartidas

Ejemplo :

Una clase abstracta Bank define una colección de Accounts. Bank tiene tres subclases : InternationalBank, NationalBank y StateBank. Obviamente, una instancia de Account pertenece a una y solo una de estas tres clases concretas.

Solución con Objetos :

Trivial. Las subclases heredan y reusan sin complicación alguna.

Solución con relacional :

El problema está en el tipado. Cuando se mapea es necesario indicar de que clase es cada objeto persistente. Y si se lleva a cabo el esquema de mapeo uno a muchos (indicada más arriba), se llega a que cada Account en la tabla debe tener un link con su padre para representar la colección. ¿Pero quién es su padre?. La abstracta no puede serlo, y mucho menos discriminar por una de las tres subclases ya que cuando recupere una instancia de una subclase de Bank no se recuperarán las accounts que referencia. Aún más, hay dos posibilidades : o se levantan todas (si el mapeo se hizo con el Bank en cuestión) o no viene absolutamente ninguna (ya que no hay link con el Bank en cuestión).

La solución más simple con lens consiste en agregar a Account tantas columnas como clases haya que puedan referenciarla. Por tanto, en este caso, Account tendrá tres columnas apuntando a las sendas clases concretas de Bank.

El approach anterior cuenta con la desventaja del desperdicio de espacio en la base de datos y del oscurecimiento del modelo. Una alternativa muy significativa para evitar esto consiste en realizar una pequeña modificación en el Lens y del tipo de información que se guarda en

la columna oid de cada objeto persistente. Para esto, en vez de almacenar el oid como un simple número, se le anteponen tres dígitos alfanuméricos. Esos tres dígitos son una referencia **unívoca** a la clase a la cual referencia. Por tanto. es necesario que el structure type que representa el mapeo deje asentado cuales son los tres dígitos que representa a la clase persistente. Y de esta manea se esta consiguiendo que una columna de una tabla pueda ser polimórfica. Es decir, referenciar tanto un objeto de la clase A como de la clase B, contando con el apoyo de los tres dígitos mencionados a la hora de determinar de que tabla es la instancia referenciada.
En el ejemplo anterior, definamos a InternationalBank con un identificador unívoco *IBA*, por ejemplo. De manera similar con NationalBank y StateBank asignándoles *NBA* y *SBA* respectivamente. Ahora supongamos que la tabla Account tiene una columna que indica la referencia al objeto al cual pertenece denominada owner (ver mapeo uno a muchos). Una instancia cualquiera de Account podría tener en la columna mencionada el valor "NBA0000023". Esto indica que el owner será una instancia de NationalBank., en particular, la instancia numerada 0000023 de dicha clase.
Consecuencias en el modelo de objetos :

Bajo el primer approach se vislumbran variables de instancias superfluas. Esto aún con consecuencia directa en la base de datos donde se mantienen tres columnas de datos donde solo una por instancia es utilizada.

Una clase singleton debe ser persistente

Ejemplo :

Se utiliza el NullObject pattern (Woolf[96]) para representar un objeto persistente, por tanto, las tablas pueden contener tanto objetos concretos como null objects.
Solución con Objetos :

Se redefine el new de la clase singleton y se declara una variable de clase para contener la única instancia de la clase. De esta manera, al recibir el new, la clase singleton testea si la variable de clase ha sido asignada previamente. Si esto ocurre retorna dicho valor. Caso contrario, ejecute un super new para crear la única instancia, la asigna a la variable de clase mencionada y retorna la misma.
Solución con relacional :

Bastante similar a la solución tradicional, con el agregado de que la variable de clase funciona como un buffer de bases de datos, ya que evita el query en caso de que el singleton esté en memoria. La única diferencia entonces apunta a la forma de crear la instancia. Por un lado el cache puede dejar de ser válido si la sesión actual difiere de la sesión en que fue cacheado (suponiendo un acceso concurrente). Por otro lado, a la hora de recuperar la instancia se presentan dos casos : que la tabla este vacía o que la tabla tenga un elemento. Si ocurre lo primero, indefectiblemente el mensaje *new* deberá disparare sobre el singleton e inmediatamente grabar la nueva instancia en la base (*session add : newInstance*). Si ocurre el segundo caso, un query a la base recuperando todas las instancias de la tabla en una colección debe ser ejecutado y posteriormente asignar como la cached instance al primer (y la único) objeto de esta colección.
Consecuencias en el modelo de objetos :

Sin consecuencias que ensucien el modelo de objetos.

Domain Specific Patterns

Versioning Pattern

Bussines Information Context

Context

Bussines contexts donde se interactúa con el mercado a través del ofrecimiento de conceptos. Un concepto no es más que lo que una empresa ofrece al propio mercado, es decir, los productos o servicios que vende.

Problem

Una organización debe poder evolucionar con el tiempo aprendiendo de estados actuales de la misma. Considerando que la venta de conceptos al mercado es el backbone de la existencia de la compañía, esta última debe prestar especial atención a como coloca los conceptos en el mercado.
Es decir, la compañía debe resolver:

- ¿Cómo administrar toda información relacionada a sus ventas de manera eficiente y efectiva?
- ¿Cómo ver su flujo de ventas desde "arriba" y no focalizado en cada concepto particular?
- ¿Cómo identificar certeramente la evolución de sus conceptos el mercado?

Forces

La experiencia y el conocimiento de la empresa por si misma puede influenciar considerablemente sobre la acción de venta. Un conocimiento profundo de como se han manejado los conceptos ofrecidos al medio puede volcar la balanza financiera eficientemente del lado de la compañía. Conocer **que es** lo que realmente se está vendiendo y conocer **que otros** conceptos se pueden vender por la misma línea de venta del primero puede derivar en un crecimiento exponencial del área de ventas.

Solution

Hay dos puntas claras para la solución del problema y son las siguientes:

1. Como resulta vital manejar información sobre como evoluciona un concepto a través de la dinámica del market, se debe mantener información de **que y como** un concepto ha sido evolucionado o versionado a través del tiempo. Esto no es más que, desde un concepto inicial sin ancestros, mantener una línea de versionamiento correspondiente a todos las modificaciones, radicales o no, que se han hecho sobre el concepto para un relanzamiento o reaparición en el mercado. Sin embargo, esto no necesariamente implica que los conceptos versionados dejan de formar parte de las ofertas de la empresa. Dos versiones del mismo concepto pueden convivir tranquilamente en el mercado.
   De esta manera resulta muy sencillo conocer la línea de evolución de un concepto, así también como los ancestros del mismo.
2. Además, como un concepto exitoso puede llevar consigo ventas adicionales o complementarias, resulta adecuado mantener por cada versión de un concepto el conjunto de conceptos complementarios que se enganchan a su venta. Estos conceptos complementarios no son más que versiones de otros conceptos existentes en el dominio de ventas de la compañía.
   Por tanto, es posible por cada versión de un concepto conocer fácilmente que otras versiones de

conceptos conforman sus complementos, así también como a quienes complementa cada versión de un concepto.

Algunas reglas deben aplicarse a la hora de versionar un concepto. Los managers pueden luego determinar los cambios necesarios, según se gestión sobre, los resultados de las reglas:

- Si el concepto a versionar tiene complementos, los lleva consigo a la nueva versión a crear.
- Si el concepto a versionar complementa a otros, nuevas versiones de los conceptos a quienes complementa deben ser creados. Esto se repite recursivamente.
- Un concepto versionado solo tiene un ancestro. Esto es, una versión de un concepto no se realiza desde varias versiones. En el caso que varios conceptos se fusionen en uno solo, un nuevo concepto original deberá ser creado.
- Una versión puede tener varios conceptos que lo referencien como su ancestro.

Examples

La yerba *Taragüí* puede verse como la versión inicial que el establecimiento lanza al mercado. Un complemento de la misma es el *Mate Listo Taragüí* ya que no es más que una venta enganchada a la compra de la yerba. Las muñecas *Babies* son otro ejemplo similar a la yerba. El producto viene con un set de ropa para muñecas que puede ampliarse con futuras compras de nuevos sets adicionales. Estos sets adicionales no son más que complementos del producto.

Ahora, regresando al ejemplo de la yerba, la primer versión puede versionarse a un nuevo producto que, por ejemplo, se llame *Taragüí Seleccionada* conformada por las más selectas hojas de yerba mate. Su ancestro es la yerba original. Cuando se versiona esta última, la nueva versión tiene consigo el mismo complemento, es decir, el *Mate Listo Taragüí* es también complemento de *Taragüí Seleccionada*. Sin embargo. los dos tipos de yerba conviven en el mercado sin problema alguno. La compañía yerbera puede decidir modificar nuevamente la versión original generando un producto *Taragüí Malteada* la cual también deriva de la original y genera otra línea de evolución.

Por otro lado, el *Mate Listo Taragüí* ha decido mejorarse lanzando al mercado *Mate Listo Rendidor Taragüí* que en vez de proveer un mate de plástico provee un mate de calabaza. Esta es una nueva versión del concepto que genera que los conceptos complementados (*Taragüí* y *Taragüí Seleccionada*) sean reversionados para contener como complemento al *Mate Listo Rendidor Taragüí*.

Aplicability

El pattern puede aplicarse en dominios como:

- Gestión y control de marketing directo
- Marketing de productos
- Auditorias
- Altas Finanzas
- Proyecciones de Ventas

Versioning Pattern **Object Model Context**

Name

Versioning Structural Pattern

Intent

Proveer manejo eficiente de objetos que se versionan y complementan a través del tiempo.

Motivation

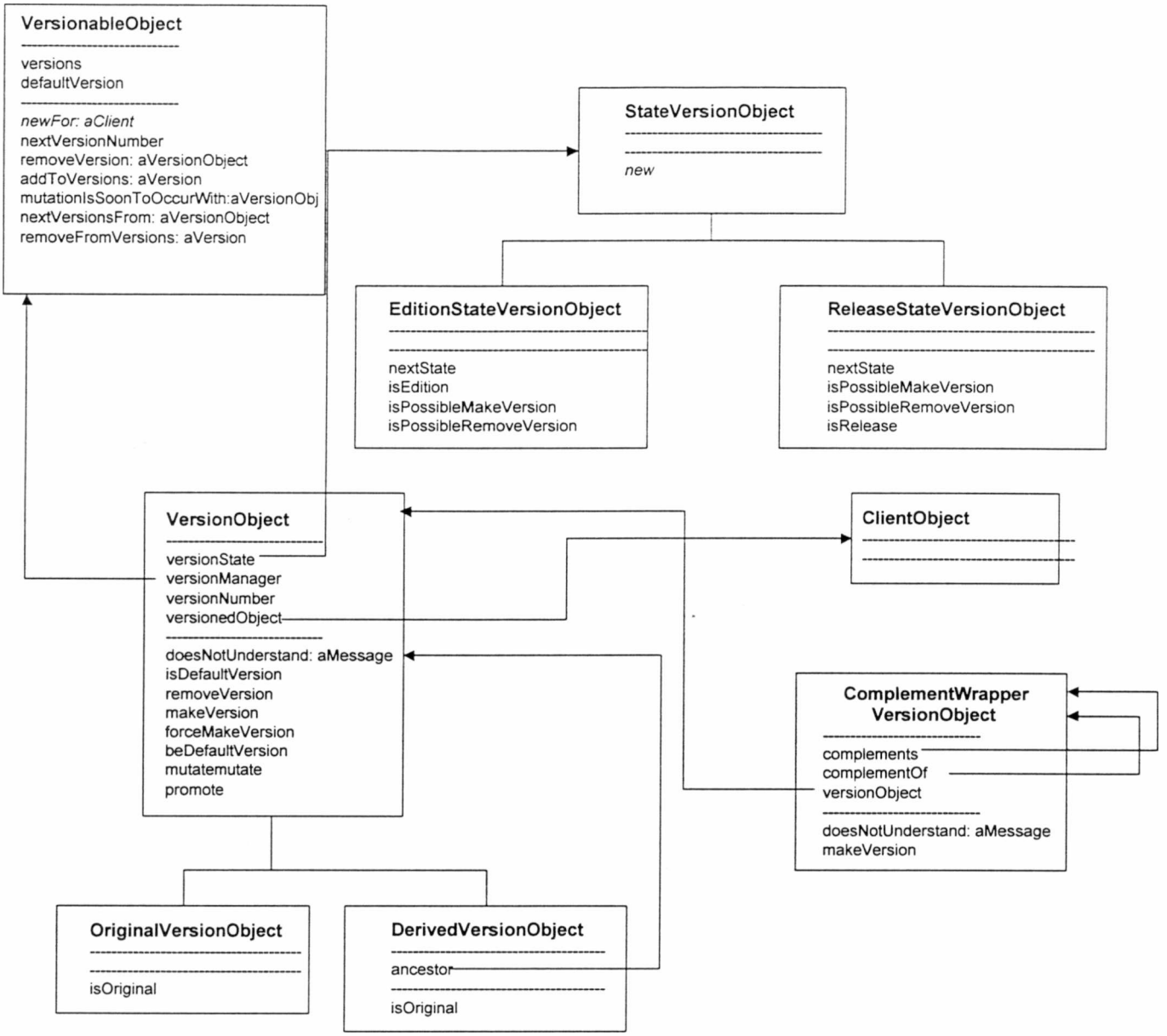
En el dominio de las bussines applications resulta vital el manejo eficiente de los productos o servicios que una empresa ofrece al mercado. Su profundo conocimiento deriva en un aprovechamiento mayor de la fuerza de ventas de los mismos. Una compañía genera, regenera, adapta, readapta, complementa y recomplementa productos o servicios en su arte de interactuar con el mercado. Esta información debe ser ágilmente manejada por su sistema de información de manera tal de tener un rápido acceso a las evoluciones/involuciones de los conceptos que se ofrecen al medio. De la misma manera, cuando un concepto decide modificarse o adaptarse a nuevas necesidades del mercado, es decir, se ha decidido versionar el concepto, debe considerarse que otro conjunto de conceptos (en realidad conjunto de versiones de conceptos) pueden verse afectados por este acto. En particular, si se versiona un concepto que complementa a varias versiones de otros conceptos, estas últimas versiones deben evolucionar a nuevas versiones donde se haga referencia a la nueva versión del concepto inicialmente mencionado. Este proceso se repite recursivamente a medida que conceptos que complementan a otros conceptos van siendo versionados.

Applicability

Este pattern es aplicable cuando:

- Un objeto evoluciona (se versiona) y debe mantenerse un lazo de evolución entre tales versiones.
- Un objeto es complementado por otros y cambios en los complementos afectan al objetos original.
- Se necesita un rápido y eficiente manejo de las versiones, a través del tiempo, de un objeto en particular.
- Se necesita un rápido y eficiente manejo de los complementos de una versión de un objeto o de todas las versiones de un objeto.
- Se necesita identificar los objetos a quienes complementa otro objeto dado.

Structure



Participants

VersionableObject: Representa la clase abstracta de todas aquellas subclases que deseen proveer manejo de versiones para sus instancias. Todo objeto versionable conoce el conjunto de versiones de sus instancias y cual es la versión activa en el dominio. Esto es, la versión default.
Provee los servicios de interacción a través de las versiones existentes, el próximo número de versión disponible. También puede determinar si la versión activa puede versionarse o puede eliminarse (ver colaborations).

ClientObject: Representa una instancia particular de una clase a la cual se desea incorporar la capacidad de versionamiento. Si se desea que, ante la mutación de su estado interno, las

ediciones sean creadas automaticamente, debería conocer al VersionObject que lo contiene y delegarle a este un update por cada cambio que realice internamente.

StateVersionObject: Representa la clase abstracta para manejar el estado de las versiones. Exige a sus subclases la respuesta por si o por no de la posibilidad de versionar una versión o de removerla de la lista de versiones de su manager.

EditionStateVersionObject: Es el estado donde las versiones pueden ser modificadas sin ningún problema, es decir, sus variables de instancias pueden ser cambiadas sin exigencia alguna. No es posible hacer una versión desde una versión en este estado. Sin embargo, es posible removerla del conjunto de versiones de un objeto. Por otro lado, tiene la responsabilidad de cambiarse de estado (promote). Cuando esto ocurre para al estado Release.

WorkingStateVersionObject: Es el estado posterior al proceso de edición de las versiones. No es posible remover versiones en este estado ni es posible hacer nuevas versiones desde el mismo. El cambio de estado (promote) hace que el estado de la versión pase a ser release.
En working no es posible cambiar el estado interno de la versión.
Este estado cobra sentido cuando se maneja visibilidad de versiones en un entorno multiusuarios de versiones. Esto es, los dueños de un objeto ven absolutamente todas las veriones, pero los clientes o usuarios solo ven las que estan en estado release. El estado working indica un estado de consolidación, pero en proceso de testing, por ejemplo.

ReleaseStateVersionObject: Es el estado final de una versión. No es posible remover versiones en este estado. Mucho menos cambiar el estado interno. Esto es, la versión permanece inmodificable. Es posible, sin embargo, crear nuevas versiones (promote) desde este estado. La nueva versión será internamente similar a la released pero estará en estado de edition para que pueda ser modificada libremente.

VersionObject: Representa un wrapping de una versión de un objeto versionable. Una versión debe tener un estado (StateVersionObject), un manager (VersionableObject), Debe poder afirmar si es la versión activa o default y su número de versión. Muchas de estas responsabilidades requieren del manager como colaborador principal o directamente delegándole la tarea. De esta manera, el cliente no debe preocuparse por el manager, y por ende pedirle colaboración directamente a quien envuelve el objeto cliente, es decir, a VersionObject. Por ende, VersionObject debe conocer es la instancia que el encapsula, esto es, cual es la instancia del ClientObject. De igual manera, sabe responde cuales son sus versiones sucesoras.

OriginalVersionObject: Representa el objeto inicialmente creado sobre el ClientObject. Es la versión original que no deriva de nadie. Si se realiza un versión nueva a partir de esta, el objeto muta a un versionable que conoce su ancestro (DerivedVersionObject) y se setea como el ancestor de este último y el original es informado acerca del nuevo sucesor.

DerivedVersionObject: Representa una versión que ha sido versionada desde otras. La cadena de derivaciones es de un ancestro por vez.

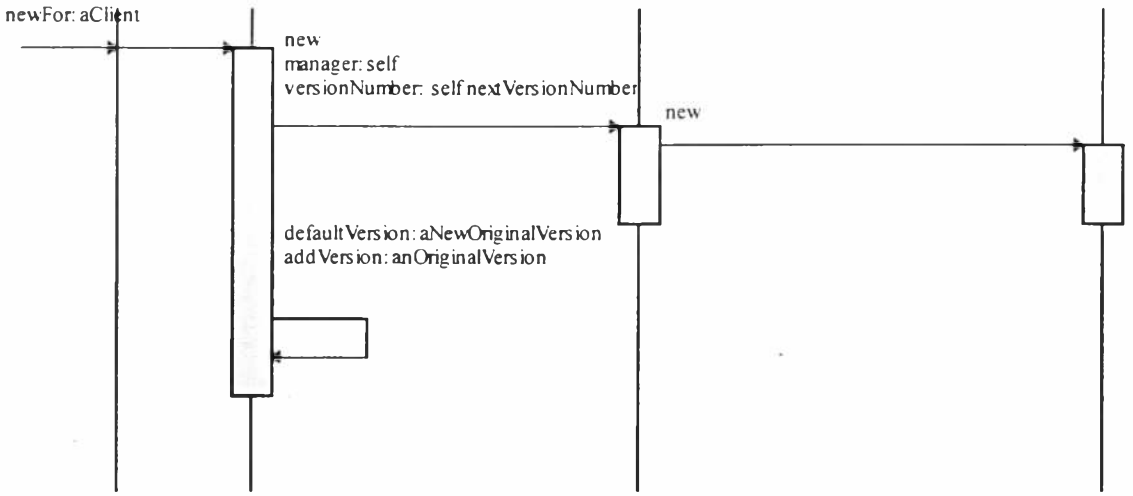
ComplementWrapperVersionObject: Como una manera de evitar acoplar el manejo de complementos al VersionObject, se implementa ComplementWrapperVersionObject que wrappea al

VersionObject y agrega comportamiento para el manejo de complementos de versiones de objetos. Obviamente, su principal colaborador-delegador resulta ser el propio VersionObject

Colaborations and Implementation

- *Inicializar un objeto versionable*

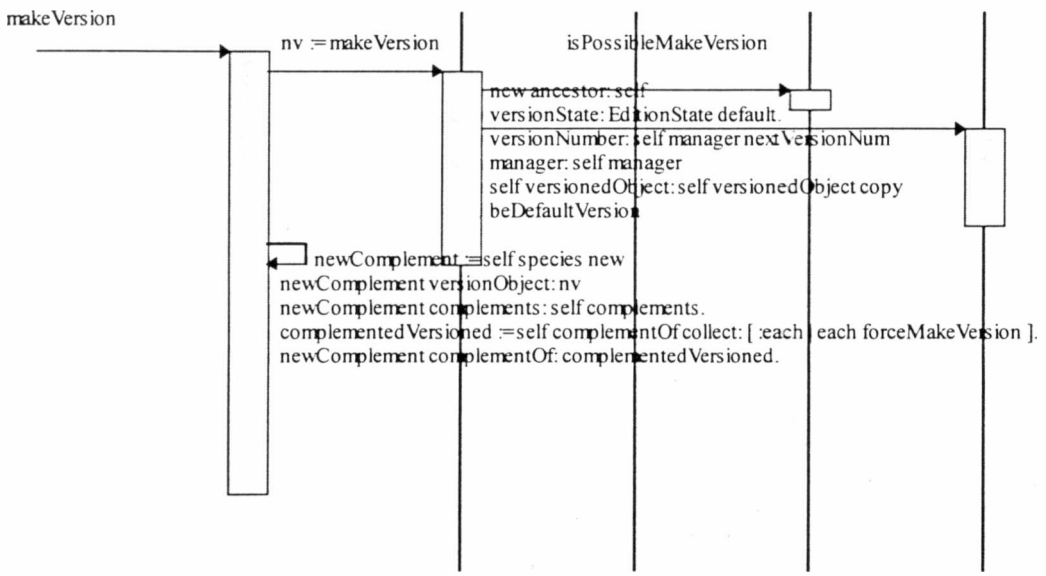
Client Object	Versionable Object	Version Object	Original Version Object	Derived Version Object	State	Edition State	Working State	Relea State



Se crea un nuevo objeto versionable que será una versión original. Esta misma se asigna como la versión número uno y como la default. Además, su estado inicial es en edición.

- *Crear una nueva versión desde otra versión existente*

Client Object	Complement Wrapper VersionObject	Version Object	Original Version Object	State	Derived Version Object	Edition State	Working State	Relea State



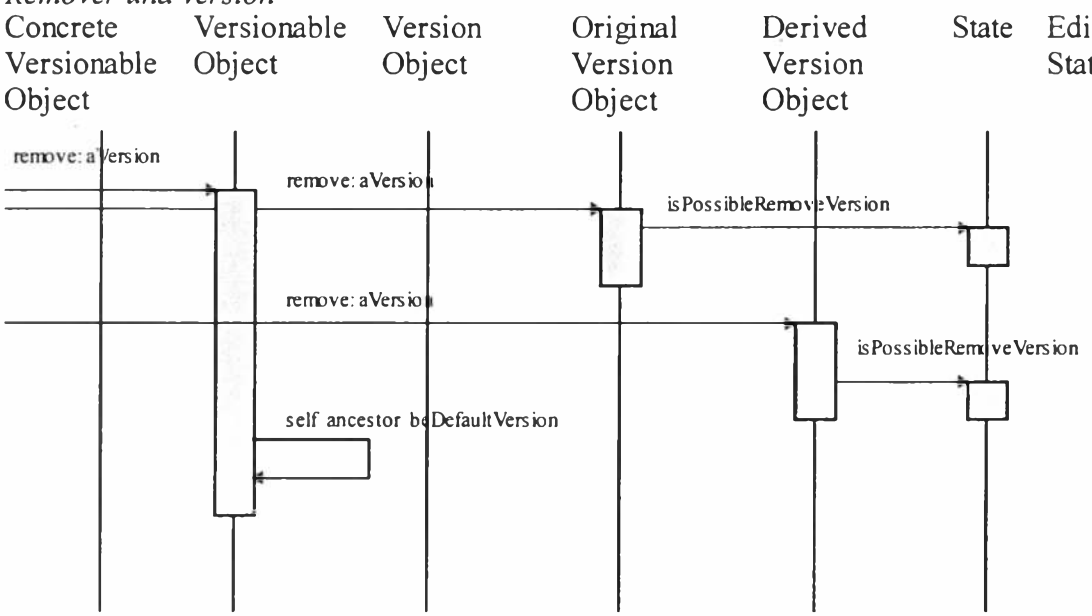
El wrapper que representa un complemento delega al VersioObject. La versión chequea si esta habilitada para versionar delegando a su estado. Independientemente si es original o derivada, crea una

nueva versión derivada y se asigna como ancestro de esta última. Toda la información que posee se la pasa a la nueva instancia junto con una copia del objeto que versiona. Además asigna la nueva versión como otra de sus sucesoras.
La nueva versión se crea con estado de edition y el manager es el encargado de proveer el número de versión de la nueva versión. La nueva versión se asigna como la default.
El Complement crea, con la nueva versión, una nueva instancia de si mismo. Los complementos de la nueva instancia son los mismos en tanto que a quienes complementa deben ser forzosamente versionados. Esto implica que si estaban en edición, pasan a release.

- *Cambiar el estado de una versión a working/release*
  La versión receptora delega el promote a su actual state

- *El Cliente envía un mensaje al ClientObject.*
  ComplementWrapperVersionObject no entiende el mensaje y, como reimplementa doesNotUnderstand:, lo deleja al VersionObject quien, como reimplementa también doesNotUnderstand:, se lo delega al ClientObject.

- *Remover una versión*



  Se borra la versión solo si está en edición (no tiene sucesores) y se asigna a su predecesor o ancestro como el default.

- *Agregar o borrar un complemento a una versión*
  La versión default o activa recibe otra versión de un objeto que formará parte de sus complementos. Esto ocurrirá solo si la versión sobre la cual se está agregando esta en edición. Caso contrario se realiza una nueva versión sobre la cual se agregará el complemento (ver *Modificar el estado interno de un ...*)

Consequences

- Hay una rápida recuperación de líneas de versionamiento desde una versión particular
- Lo anterior se repite para los complementos

- Los estados de las versiones pueden ser utilizados también para niveles de acceso al ambiente de versiones y para asignar usuarios con privilegios en la creación de versiones.
- Desde un punto, es decir, desde una versión, se puede observar con bajo costo de acceso las líneas de evoluciones de la misma.
- Desventaja. No se maneja en forma automatica los update del ClientObject.

Sample Code

Class:	VersionableObject
Superclass:	Object
Category:	DIMBO - Versioning
Instance variables:	versions defaultVersion

private
removeFromVersions: aValue

version handling
addToVersions: aValue
mutationIsSoonToOccurWith: aVersionObject
    mutation is soon to occur so, aVersionObject, that is a release, must be saved.
    the current version stays in edition
nextVersionsFrom: aVersionObject
    the receiver answers the versions after aVersionObject
removeVersion: aVersionObject
    aVersionObject is removed from the versions of the receiver. The ancester of this object
    is now the default version
versions

accessing
defaultVersion
defaultVersion: aValue
nextVersionNumber
original
originalVersionClass

initialize
initializeFor: aClient

MetaClass: **VersionableObject class**
instance creation
newFor: aClient
    A new versionable object manager is created with the first
    version of aClient. Clients must use this protocol to create a
    new versionable object . Note that is different to create a new version
    of a versionable object

Class:	VersionObject
Superclass:	Object
Category:	DIMBO - Versioning
Instance variables:	versionState versionManager versionNumber versionedObject

This class acts like a wrapper on the object choosen to be versionable. In order to create a versionable object the client must send the message: VersionableObject newFor: aClientObject, where aClientObject is the object to be versionable. It answer an instance of VersionObject that wrappers whole aClientObject protocol. Thus, the client don't pay attention about the new structure of the versionable object.
VERY IMPORTANT: The only requeriment for the clients is that he must sends change: #value to himself because a dependency transformer created by each VersionObject is listening for changes in order to change the state of the version when the client wants mutate the object. This message must be sended BEFORE the change takes action. If the clients is complex may need implement the postCopy message

accessing
manager
manager: aManager
versionedObject
versionedObject: aClient
    the receiver sets as depend on the client
versionManager
versionManager: aValue
versionNumber
versionNumber: aValue
versionState
versionState: aValue

copying
postCopy

state handling
mutate
    the receiver must be versioned if his state is release because a change
    is soon to occur
promote
version handling
allVersions
beDefaultVersion
    the receiver sets his manager with him as default
forceMakeVersion
    The receiver tries to make a new version from him.
    The new version will be in edition state.
    If it is not possible the method returns nil. Otherwise the new version.
makeVersion
    The receiver tries to make a new version from him.
    The new version will be in edition state.

If it is not possible the method returns nil. Otherwise the new version.

nextVersions

the receiver answers the versions after him in the time

removeVersion

The receiver will be removed only if it is in edition state. Anyway, the manager is responsable to do that. If is not possible to remove it, nill is answered. (the manager answer the new default version)

testing

isDefaultVersion

private

concreteMakeVersion

a new version is made

derivedClass

wrapper handling

doesNotUnderstand: aMessage

the doesNotUnderstand must be redefined in order to the receiver acts like a wrapper on versionable object. If the versioned object does not understand aMessage default behavior is executed

---

Class:	OriginalVersionObject
Superclass:	VersionObject
Category:	DIMBO - Versioning

retrieving

original

testing

isOriginal

Class:	DerivedVersionObject
Superclass:	VersionObject
Category:	DIMBO - Versioning
Instance variables:	ancestor

retrieving

original

accessing

ancestor

ancestor: aValue

testing

isOriginal

Class: StateVersionObject
Superclass: Object
Category: DIMBO - Versioning

testing
isEdition
isPossibleMakeVersion
isPossibleRemoveVersion
isRelease

promoting
promote
the receiver changes to next state

MetaClass: **StateVersionObject class**
Instance variables: **SingleInstance**

instance creation
new
the receiver is a singleton

Class: EditionStateVersionObject
Superclass: StateVersionObject
Category: DIMBO - Versioning

accessing
name

testing
isEdition
isPossibleMakeVersion
isPossibleRemoveVersion

promoting
nextState

Class: ReleaseStateVersionObject
Superclass: StateVersionObject
Category: DIMBO - Versioning

promoting
nextState

testing
isPossibleMakeVersion
isPossibleRemoveVersion
isRelease

Class:	ComplementWrapperVersionObject
Superclass:	Object
Category:	DIMBO - Versioning
Instance variables:	complements complementOf versionObject

This class is a wrapper on VersionObject. He adds complement and complemented handling. These complements are version objects.

wrapper handling
doesNotUnderstand: aMessage
    the doesNotUnderstand must be redefined in order to the receiver
    acts like a wrapper on version object. If the version object does not understand
    aMessage default behavior is executed
complements handling
addToComplementOf: aValue
addToComplements: aValue
complementOf
complementOf: aCollection
complements
complements: aCollection
removeFromComplementOf: aValue
removeFromComplements: aValue

accessing
versionObject
versionObject: aValue

initialize
initialize

version handling
makeVersion
    This is a wrapped protocol from the version object. A new version is created. The method answers it

Known Uses

DIMBO (DIrect Marketing with Bussines Objects - Lucio Dinoto tesis degree - UNLP - Argentina).

Related Patterns

Versioning Pattern (Bussines Information Context)
Composite Pattern[1] para la modelización de las versiones originales y derivadas.
Factory Method[1] para el creado de versiones desde los states.
Iterator[1] para el VersionableObject a la hora de proveer interacción con sus versiones.
State[1] para el manejo de los estados de las versiones.

Bibliografía

[1] Design Patterns - Elements of Reusable Object-Oriented Software - Gamma, Helm, Johnson, Vlissides
[2] Object Oriented Databases -

Apéndice

Modelo de Use Cases

BUSSINES APPROACH - MODELO DE USE CASES

Entendiendo el Negocio...

"Identifique características comunes de sus clientes y podrá ver al próximo de ellos caminando por la calle"

(* Los corchetes [] significan las posibles opciones iniciales que existen. Si se referencian con doble corchete [[]] indica múltiple choice. Las mismas pueden incrementarse a lo largo del ciclo de vida del negocio.)

Actor Gestor de Marketing Directo

1. Establecimiento de Canales/Medios de Distribución

El gestor instancia un nuevo medio para operar con sus servicios. El medio puede surgir del canal de venta directa (como si fuera un supermercado) o bien puede ser un medio electrónico (tv, radio, fax, email) o impreso (diario, revista, correo).

El instanciar un medio resulta productivo para la venta de uno o varios conceptos que se realizarán a través de uno o varios de estos medios (por ejemplo, publicitar en un diario para vender directamente en un local físico). Cada tipo de medio lleva consigo características particulares que deben ser explicitadas, como ser:

Medio Directo: nombre, lugar, descripción, dirección, tipo de lugar [comercio, exposición, conferencia], característica del medio [estable, periódica, ocasional por fechas, ocasional por temporada].

Medio Electrónico: nombre

Televisión: guión, duración, secuencia de imágenes y texto, musicalización, objetivo [venta directa, publicidad y promoción, indicador de otro medio]

Radio: slogan, texto, teléfonos indicados, musical, segmento horario

Email: dirección electrónica, subject, contenido, signature

Fax: teléfono, título, texto

Telemarketing: introducción, enumeración de beneficios, esquema de preguntas conducidas por respuestas, cierre, objetivo [venta, recepción]

Medio Impreso: nombre, titular, texto, descripción fotográfica

Periódico: ubicación [página fija, sin posición concreta, en suplementos, encarte]

Revista: afinidad revista-concepto [si,no], publica también la competencia [si,no], espacio en porcentaje sobre la hoja, posición [central, principio, final], temporada [verano, otoño, invierno, primavera]

Carta: tipo [correo, insertado en compras, acompañando notas de factura, confiado a terceros, correo en conjunto con otras empresas], descripción del folleto, carta empresaria, premios por rapidez, descripción hoja-cupón de respuesta, descripción sobre de respuesta, estilo de la carta [misterioso sin revelar contenido, seductor, expresivo],

secuencia lógica: párrafo con el beneficio, texto descriptivo del beneficio principal, texto descriptivo de otros beneficios y de las consecuencias de los mismos, texto descriptivo de testimonios y pruebas, texto descriptivo de lo que el lector pierde si no aprovecha la oferta, texto de reiteración con énfasis de los beneficios, texto final incitador a la acción de compra.

De similar manera el gestor puede decidir modificar un medio existente. Esto provoca que se cree un nuevo medio con las características anteriores que conforma una nueva versión del medio editado. Esto es de esta manera porque seguramente existen planes de marketing directo y ventas de conceptos que se han realizado sobre dicho medio de distribución

2. Creación de un Nuevo Concepto (Producto/Servicio)
El gestor crea un nuevo concepto que puede ser tanto un Producto como un Servicio. En el caso de producto, el concepto no necesariamente debe corresponderse con un producto real o físico. Puede ser sobre un conjunto de ellos o tan solo sobre una compra determinada o aún sobre una compra común y corriente. Por ejemplo, en un supermercado el concepto (producto) a vender puede ser denominado "Góndolas", ya que el plan de marketing puede apuntar a maximizar el vaciamiento del producto "Góndolas".
Las características que tienen los Conceptos son: nombre, descripción, posicionamiento, fecha de creación, personas responsables en la creación del mismo.
En tanto que los Productos se caracterizan por su marca y los Servicios por el conjunto de personas/entidades que lo ejecutan
Todo concepto puede ser original o bien tener un predecesor. Por otro lado, todo concepto puede ser el último de su generación o bien tener un sucesor. Además, todo concepto puede tener también conceptos complementarios. Un concepto complementario tiene la particularidad de extender el concepto inicial y determinar una línea venta enganchada.

3. Creación de un Nuevo Concepto Mejorado o Complementario de otro existente
El gestor decide evolucionar un concepto. Debe decidir si es complementario de uno existente o bien es una mejora (versionamiento) de uno existente. Las condiciones y requisitos son similares a la creación de un producto con la diferencia que debe establecerse a quien complementa o a quien mejora.

4. Establecimiento de Segmentos de Mercado
Los segmentos en conjunto son útiles para ser intersecados sobre otro conjunto de clientes, obteniendo como resultado el conjunto de clientes que cumplen ciertos requisitos. El gestor mantiene instanciados segmentos de mercado o bien crea nuevos segmentos para conseguir particiones de clientes para fines específicos.
Los segmentos se dividen en cuatro grupos clásicos a su vez particionado: Segmento De Compra , Segmento GeoDemográfico , Segmento Psicológico , Segmento Empresarial.
A su vez, cada uno de estos se divide en subsegmentos
Segmento De Compra :

Recencia (período de última compra)

Origen (tipo[lista externa, lista interna, compra voluntaria, cadena de amistad, anuncio, promoción])

Modalidad de Pago (tipo[contado, cuenta corriente, plan, tarjeta])

Forma de Compra (tipo[correo, directa, teléfono, Fax, E-mail, TV]

Frecuencia (número de compras en un período). Existen frecuencias por semana, mes, año y la acumulada historica

Valor Monetario de la Compra (promedio de compra en una frecuencia establecida el en punto anterior)

Edades (tipo [primera edad (hasta los 25), segunda edad (hasta 65) y tercera edad]

Tipo de Concepto (nombre del concepto)

Segmento GeoDemográfico :

Geográfico (país, provincia, ciudad, barrio, código postal)

Demográfico (estado civil, religión, sexo, edad, nacionalidad, lugar de residencia, componentes del grupo familia (padre, madre, madrastra, padrastro, hijos, hijas, abuelas, abuelos, edades, nombres, fechas de nacimiento y lugar (de todos)).

Socio-Económico (renta per cápita, ocupación, nivel de educación, clase social [alta, media alta, media, media baja, baja], propiedades, grupos de referencia, clubs en los que actúa, deportes que practica, donde transcurre sus vacaciones

Segmento Psicológico :

Beneficios Esperados (conjunto de tipos [precio, calidad, servicio, fidelidad, apariencia, vanidad, status, confort, seguridad])

Estilo de Vida (personalidad [introvertido, extravertido, dirigido, integrado], trabajo, hobbies, asociación, deportes, lugar de descanso, familia, profesión, comunidad, moda, éxito, actividad política, comidas, opinión económica [si, no], opinión política [si, no], opinión cultural [si, no], opinión social [si, no]).

Segmento Empresarial :

Demográfico (división de clientes, sectores, emplazamiento geográfico)

Compra (criterios, relación cliente comprador, estructura del poder, política de compras)

Situación (urgencia, aplicación de los conceptos adquiridos)

Características (personales del comprador)

Variables de comportamiento (Beneficios esperados (conjunto de tipos [precio, calidad, servicio, fidelidad, apariencia, vanidad, status, confort, seguridad]) , posición jerárquica, tasa de utilización o de uso del producto, grado de lealtad[alto,medio,bajo], sensibilidad a factores de marketing[alta,media,baja], usuarios generales del producto).

5. Establecimiento de Parámetros y Fórmulas de Valor Empresario sobre el cliente y sobre conceptos

El gestor puede instanciar nuevos parámetros para luego ser aplicados sobre los clientes para obtener sus calificaciones empresariales. De similar manera estos parámetros pueden ser aplicados sobre conceptos.

Un valor o calificación empresaria es un valor numérico fruto de una función matemática n-aria, donde la aridad puede ser determinada por el propio gestor. Existen parámetros

preestablecidos que pueden combinarse en función de otros parámetros también preexistentes o agregados por el gestor.
Todos los parámetros existentes son en función de información obtenida de la propia base de conocimiento de clientes como desde las transacciones realizadas con ellos, esto es, el conocimiento de ventas realizadas. Los parámetros se instancian numéricamente de dos formas posibles: por cada cliente existente o por cada concepto vendido. Dichos parámetros se combinan incondicionalmente entre si en formas de productos, cocientes, sumas o restas lo cual determina una función. Esta función es la que se aplica sobre los clientes para poder clasificarlos bajo un orden decreciente o creciente según se establezca posteriormente a la hora de generar los listados.
Parámetros clásicos preexistentes y respectivas fuentes son :

(A) Ventas Totales (Concepto)
(B) Promedio de Gastos (Cliente)
(C) Venta Promedio (Concepto)
(D) Rotación (Concepto)
(E) Clientes Promedio (Cliente)
(F) Nro. Ventas Mínimas (Cliente)
(G) Tráfico (Cliente)
(H) Promedio de Gasto por Visita (Cliente)
(I) Clientes Existentes (Cliente)
(J) Clientes Compradores (Cliente)
(K) Clientes Privilegiados (Cliente)
(L) Nro. Promedio de Conceptos Comprados (Concepto)
(M) Precio Promedio de Conceptos Comprados (Concepto)
(N) 365 días
(O) 30 días
(P) 7 días

Cuando el gestor instancia una nueva fórmula, el mismo indica las fuentes desde sus clientes o desde las ventas combinándolas para obtener funciones que luego se aplicarán sobre los mismos orígenes.

6. Generación de Oferta sobre Conceptos particulares

Una oferta se compone de lo que se dice y cómo se lo dice. El "qué" se relaciona a la estrategia y el "cómo" a las tácticas de presentación. Una oferta encapsula un concepto. Diferentes ofertas para el mismo concepto pueden generar brutos de venta muy diferentes.
Los componentes básicos que conforman una oferta y que el gestor debe asignar son :

- Límite de cantidad del concepto ofrecido
- Límite de tiempo en que se ofrece
- Fecha de inicio de la oferta
- Descuentos ofrecidos
- Sorteos ofrecidos
- Regalos ofrecidos
- Ventajas ofrecidas
- Costo del concepto

- Valor percibido por el cliente
- Valor comparativo con la competencia
- Precio (Notar que Precio - Costo = Margen)

Conforme un concepto tiene su propio posicionamiento, una oferta puede también considerar un propio posicionamiento en el mercado actual.
Por otro lado, una oferta se desarrolla vía un determinado conjunto de medios de distribución.
Cada oferta debe llevar consigo un **reductor de riesgo de rechazo a la misma,** del potencial comprador. Esto es, se debe hacer los máximos esfuerzos al menor costo para que el potencial consumidor adquiera el concepto sin sentir riesgo al respecto. Por tal causa se debe dejar establecido un plazo de garantía de devolución, un período de prueba gratuito (que puede ser nulo) y la propia garantía conjuntamente con las condiciones para que estos plazos sean válidos.
La oferta, en un principio, puede ser con pago es al contado, con tarjeta, contra reembolso o en crédito a cuotas. De cualquier manera las formas de pago pueden variar con el tiempo.
Además las ofertas incluyen un compromiso que establece el tiempo y vigencia en la relación empresa-cliente para con la oferta. Tal compromiso puede ser una subscripción semanal, mensual o anual, un plan continuo, un círculo, un club o directamente ocasional. Este último se produce cuando la relación empresa-cliente antes mencionada perdura solo un instante de tiempo que es cuando el cliente adquiere el concepto una y solo una vez.

7. Identificación de Puntos de Venta
Un punto de venta es el lugar donde se ejecuta un plan de marketing. El punto de venta está sujeto al medio por el cual se está distribuyendo el concepto. Existen casos donde el punto de venta y el medio coinciden, por ejemplo, un supermercado tiene un medio para la venta que es su propio local y su punto de venta también es dicho local. Otro caso puede ser el en el que el medio es la radio y el punto de venta la zona costera para promocionar un nuevo helado.
Un punto de venta puede calificarse como fijo o temporal. Estos últimos se corresponden a aquellos lugares donde la venta solo tiene significado en temporadas determinadas, en fechas especiales o bajo premisas preestablecidas.

Actor Gestor Front-End de Marketing Directo

(*[1])
8. Use Case Abstracto Creado de Plan
Un plan de marketing directo no es más que una estrategia que gestiona un manager para lanzar al mercado una o más ofertas. Puede considerarse como una secuencia de actividades dinámicas preestablecidas sujetas a periódicas y hasta profundas revisiones. Dicho plan cubre tres cuestiones básicas:

1. ¿Dónde se está ahora? Respuesta desde la finalidad del negocio y el análisis del mercado.
2. ¿Dónde se desea estar? Respuesta desde el establecimiento de metas y objetivos.
3. ¿Cómo se llega allí? Respuesta desde las decisiones sobre las metas y objetivos.

Una empresa puede tener solo un plan de marketing directo sobre el cual hacer la gestión y el control toda su vida o bien puede ir instanciando varios de ellos a través del tiempo. Además un plan debe dejar establecido cuales son los medios por los cuales se promociona, se publicita y se vende el concepto entre otros. También indicar quienes conforman la competencia global contra su empresa. Esto es, realizar una comparación sincera con el lugar que ambos ocupan en el mercado.
Por otro lado debe tener en cuenta cuales son las influencias externas (gobierno, inflación, competencia, orientaciones tecnológicas etc) que lo van afectando positiva o negativamente.
Un plan también puede ir apuntando cada problema (generalmente por influencias externas) que ha ido y va surgiendo con su clara identificación y descripción y eventualmente las soluciones u oportunidades que se le encontraron.
Las metas sirven a efectos de cumplir el objetivo a largo plazo. No son más que acciones que se enmarcan en plazos normales de tiempo que van desde una fecha inicial a un tope. El objetivo es el marco final que la empresa quiere alcanzar como resultado de la ejecución de su plan de marketing. Cada plan establece acciones que deben en un intervalo de tiempo. El actor gestor front-end debería ser informado por el actor gestor controlador sobre las tareas (acciones) pendientes al día actual. De esto se desprende que las acciones tienen tres estados posibles : encolada, ejecutandose y pendiente.

Con todos estos datos, en cualquier momento del tiempo el o los auditores/consultores correspondientes tomarán la información existente en cada plan para luego obrar a conveniencia.

En este contexto, puede manejarse un solo plan de marketing directo con un firme objetivo que es la "reclutamiento, manutención y evolución comercial de clientes y con metas predefinidas que consisten en una batería de acciones de control disparadas para obtener conclusiones sobre lo que nos rinden los clientes. Es decir, puede existir un plan default de marketing drecto.

El gestor identifica los medios que se van a utilizar y su respectiva funcionalidad en este marco, así como las ofertas que van a dispararse y los segmentos sobre los cuales accionar. De esta manera el gestor obtiene los clientes sobre los cuales aplicarlos en base a la intersección clientes-segmentos.
Define el objetivo del plan y la secuencia inicial de acciones o metas para su logro. Las mismas en forma estándar consisten de acciones de control que realiza el propio gestor de marketing directo y que se mencionan más abajo. Estas acciones indican **qué es** lo que se debe evaluar y **cómo** se debe evaluar. Una acción entonces puede estar completa o incompleta dependiendo de su ejecución.

Debe quedar en claro, que en el contexto del marketing directo una meta o acción es tan solo una orden de evaluación o control en una plazo determinado desde el inicio del plan o desde la última acción o por causas externas. Cada acción generalmente dispara tests los cuales provocan una evaluación y conclusión final que debe ser asentada.

Es importante aclarar que una oferta puede estar inmersa en varios planes. Por otro lado, como un plan engloba ofertas (que llevan consigo el/los medio/s) que encierran conceptos, la compra de estos últimos se asocia a una oferta en particular en el contexto de todos los planes que la incluyen. Así, a la hora de calcular la rentabilidad sobre cada plan, la venta de un concepto en una oferta particular afecta a todos aquellos planes que la contienen. De cualquier manera, el hecho de que una oferta pertenezca a más de un plan es muy poco común.

9. Creado de Plan(*[1]) de Marketing Directo para Adquisición de Nuevos Clientes
El gestor aplica lo referenciado en (*[1]) para todos los clientes dentro de su dominio de información que son considerados clientes posibles y/o potenciales.

Actor Gestor Back-End de Marketing Directo

9. Creado de Plan(*[1]) de Marketing Directo para Fidelización de Clientes Actuales
El gestor aplica lo referenciado en (*[1]) para todos los clientes dentro de su dominio de información que son considerados clientes compradores, existentes, propagandistas y privilegiados.

11. Creado de Plan(*[1]) de Marketing Directo para Reactivación de Clientes Antiguos
El gestor aplica lo referenciado en (*[1]) para todos los clientes dentro de su dominio de información que son considerados antiguos.

12. Creado de Plan(*[1]) de Marketing Directo para Clientes Selectivos o Privilegiados
El gestor aplica lo referenciado en (*[1]) para todos los clientes dentro de su dominio de información que son considerados privilegiados.

Actor Controlador de Marketing Directo

13. Identificación de Clientes por Segmentos
El controlador selecciona una serie de segmentos que pueden surgir de:

Este conjunto de segmentos es aplicado sobre el conjunto todo de clientes que conforman la base de conocimiento obteniendo otro conjunto de clientes (menor o igual) que resulta de filtrar sus características conocidas con la de los segmentos instanciados. Este conjunto de clientes tiene dos particiones:
La primera consiste de los clientes que han hecho matching con las características del segmento. Esta primer partición se enlista ordenada por número de matchings en forma descendiente indicando también el número de características sin matching. Tales características también son mostradas discriminadamente.
La segunda consiste en los clientes en los cuales algunas características no pueden ser matcheadas por falta de información (que es diferente a que no haya matching!). Esta lista es ordenada por número de matchings en forma descendiente indicando el número de características desconocidas y las características sin matching. Igualmente, los tres tipos de características son mostradas en forma discriminada.

14. Análisis de Rentabilidad por Concepto
El gestor debe calcular la rotación de cada concepto (producto o servicio), para calcular la rentabilidad (margen x rotación) de los mismos. El informe se discrimina por cada versionamiento y componente de los conceptos para luego obtener los subtotales por concepto, seguido de la rentabilidad de cada plan existente en el dominio del negocio y finalmente la rentabilidad total de la organización.

15. Visión de Clientes Potenciales y Calificación Parametrizada
Se realiza un informe de los clientes de la empresa denominados potenciales jerarquizados por la o las fórmulas de calificación que se aplican sobre cada uno por elección del actor Controlador.
Un cliente potencial es aquel que ha sido fuertemente identificado (a través de contactos o a través de listas externas) pero que aún no ha tenido compra alguna sobre el negocio.

16. Visión de Clientes Compradores y Calificación Parametrizada
Se realiza un informe de los clientes de la empresa denominados compradores jerarquizados por la o las fórmulas de calificación que se aplican sobre cada uno por elección del actor Controlador.
Un cliente comprador es aquel que la empresa ya ha identificado y tiene una relación comercial latente pero aún no solidificada. Esto es, el cliente es un comprador ocasional para la empresa y es probable que también tenga contacto con la competencia.

17. Visión de Clientes Posibles y Calificación Parametrizada
Se realiza un informe de los clientes de la empresa denominados posibles jerarquizados por la o las fórmulas de calificación que se aplican sobre cada uno por elección del actor Controlador.

Un cliente posible es aquel del que la empresa sabe no demasiada información. La fuente es el contacto con el mismo o posibles listas externas. No ha habido interacción comercial con el mismo y no se sabe de él lo suficiente como para considerarlo un cliente potencial.
Es el estado previo de un cliente antes de pasar a ser potencial o comprador.

18. Visión de Clientes Existentes y Calificación Parametrizada
Se realiza un informe de los clientes de la empresa denominados existentes jerarquizados por la o las fórmulas de calificación que se aplican sobre cada uno por elección del actor Controlador.
Un cliente existente es aquel que tiene con la empresa una considerable relación comercial. De el se conoce muchísimo (eventualmente todo). De cualquier manera puede darse que tenga algún mínimo contacto con la competencia.

19. Visión de Clientes Propagandistas y Calificación Parametrizada
Se realiza un informe de los clientes de la empresa denominados propagandistas jerarquizados por la o las fórmulas de calificación que se aplican sobre cada uno por elección del actor Controlador.
Un cliente propagandista es aquel que ha cumplido y cumple el rol de encadenar clientes. Esto es, ha sugerido, recomendado, promocionado los conceptos de la empresa a amigos, familiares, allegados, su clientela, etc. Este tipo de clientes debe ser siempre correspondido en sus exigencias para que siga acercando nuevos clientes. Estos clientes sienten a la empresa como parte suya, la defienden y promocionan fanatizándose en su crecimiento ya que en general se verán también favorecidos. Algunas veces este tipo de clientes se generan otorgando regalos o descuentos por atracción de clientes.

20. Visión de Clientes Antiguos y Calificación Parametrizada
Se realiza un informe de los clientes de la empresa denominados antiguos jerarquizados por la o las fórmulas de calificación que se aplican sobre cada uno por elección del actor Controlador.
Un cliente antiguo es aquel que ha sido un cliente estable (existente, propagandista, comprador) pero que lleva un tiempo considerable sin interactuar con la empresa.

21. Visión de Clientes Privilegiados o Seleccionados
Se realiza un informe de los clientes de la empresa denominados privilegiados jerarquizados por la o las fórmulas de calificación que se aplican sobre cada uno por elección del actor Controlador.
Un cliente privilegiado es aquel que además de ser un cliente existente, es decir que tiene una muy considerable acción de compras con la empresa, realiza compras selectivas o de alto valor económico. Sobre este tipo de clientes debe prestarse una atención especial ya que conforman un caudal de ingresos muy desequilibrante en la economía de la empresa.

22. Rastreo de Devoluciones de Productos
El gestor controlador solicita un informe de aquellos productos (solo productos, no servicios) que han ocasionado devolución por parte de los clientes.

23. Rastreo de Insuficiencia de Stock
El gestor controlador solicita un informe de aquellos conceptos (productos o servicios) que no han podido venderse ante la oportunidad correspondiente por falta de stock. Esto es natural en un producto no stockeado, mientras que en un servicio se da por la falta de recursos humanos o tecnológicos para su ejecución.

24. Análisis de Medios Utilizados
El gestor controlador solicita un análisis de medios utilizados en los diferentes planes de marketing. El mismo se realiza discriminado por medios y a su vez, por cada medio, los productos o servicios vendidos con su cantidad y su monto. Además se presenta el total facturado por cada medio y la cantidad de unidades vendidas en total por cada medio.

25. Rastreo de Atenciones a Problemas Post-Venta y Consultas Pre-venta
Se realiza un informe de todas las consultas y respuestas a problemas, quejas o cuestiones que plantean los clientes acerca de un producto o servicio adquirido u observado. Esto puede marcar insuficiencia de información en el packaging de venta o en la acción motivadora o publicitaria realizada. De esta manera se rastrean conceptos con problemas, conceptos con falta de información, conceptos con información escasa en la promoción motivadora o bien clientes con interés en un particular concepto.

26. Testeo de la Evolución de Conceptos y la Respuesta en el Mercado
Se realiza un informe de conceptos que han sido evolucionados (nuevas versiones) presentando desde el producto original hasta el último versionamiento. Este informe consiste de los medios utilizados por cada versión con la cantidad y monto en cada uno de ellos y los totales por versión. Finalmente se reporta el total de unidades y el monto facturado a lo largo de todos los versionamientos.

27. Testeo de la Complementos de Conceptos y la Respuesta en el Mercado
Se realiza un informe de aquellos conceptos que han sido complementados presentando el original y cada uno de los complementos y recursivamente los complementos de los complementos. El informe consta de los medios utilizados por cada concepto y el monto y total vendido en cada uno de estos últimos. También se subtotaliza el monto y total de cada conceptos en todos los medios y por ultimo se totaliza el total de unidades y el monto final desde el original hasta todos sus complementos recursivos.

Actor Ejecutor de Acciones de Marketing Directo

28. Venta Directa de Conceptos
El ejecutor de las acciones de marketing directo realiza la venta de un concepto (producto o servicio) y deja asentada la información del mismo que contiene:

Actor Operario sobre la Base de Conocimiento de Marketing Directo

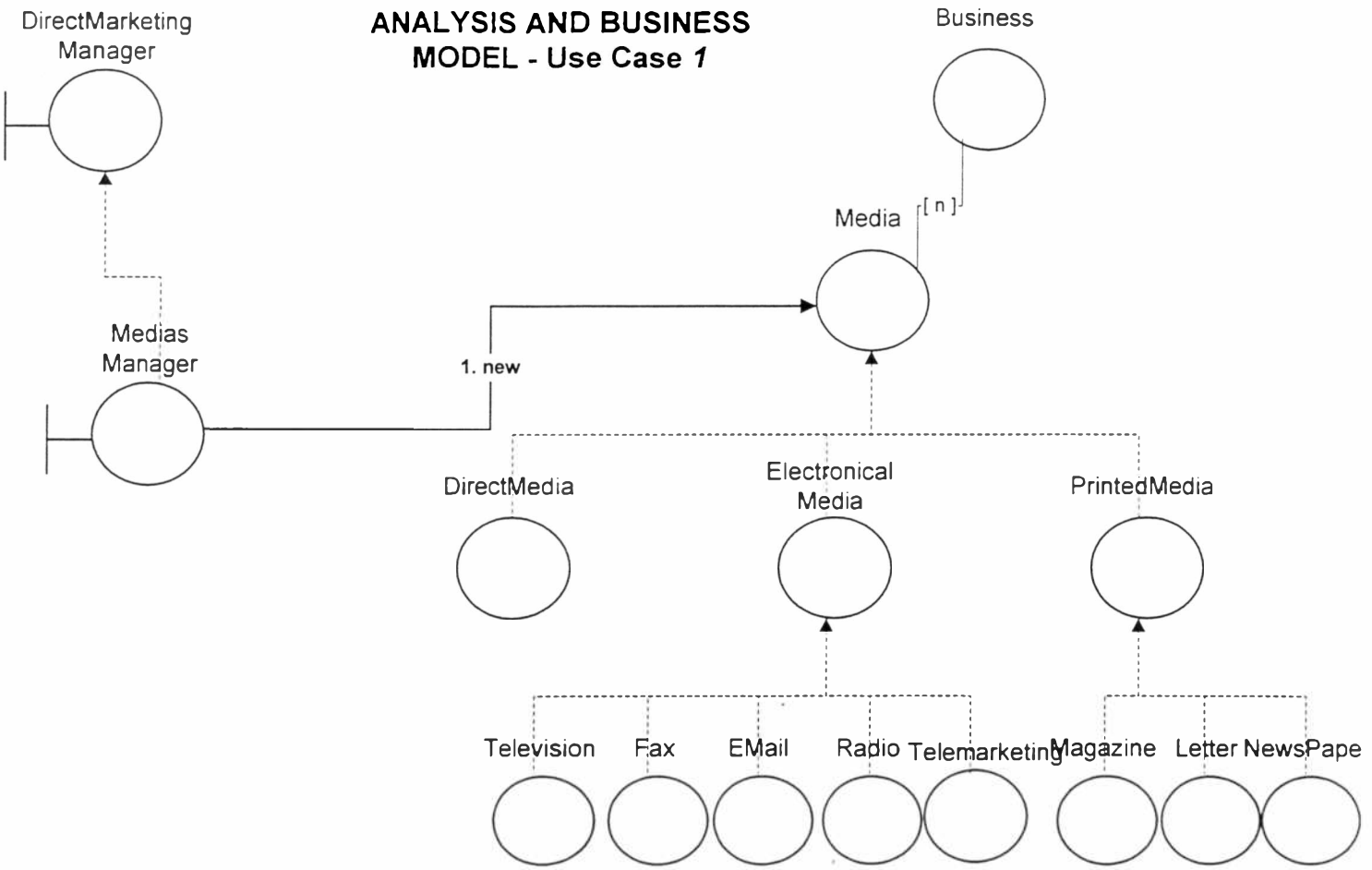
29. Estandarización y Actualización de Direcciones de Clientes
El operador debe identificar aquellos clientes que por redundancia en la base de datos tienen dos direcciones. Esto puede darse por ejemplo, si se cuenta con información antigua del cliente que posteriormente se ha mudado a una nueva dirección y ha sido asentado en la base de datos con esta nueva dirección. El operador debe considerar que eventualmente puede darse que un cliente "Juan Pérez" pueda estar asentado con "J.Pérez" o "Pérez Juan" *(o Juancito ¿ No ?)*. Esto puede implicar que se trate de la misma persona. El operador debe estandarizar la base en una sola dirección ya que las acciones de marketing directo que por ejemplo, hagan uso del correo para informar de algo al cliente incurrirán en un inútil gasto doble por envío y provocarán por otra parte una sensación de empresa desorganizada.
Esta estandarización y actualización se refleja también en los teléfonos, el fax y el email.

30. Purificación de Redundancias
El operador debe considerar que eventualmente puede darse que un cliente "Juan Pérez" pueda estar asentado con "J.Pérez" o "Pérez Juan". Esto puede implicar que se trate de la misma persona y debe ser unificada la información de cada entrada en una sola.

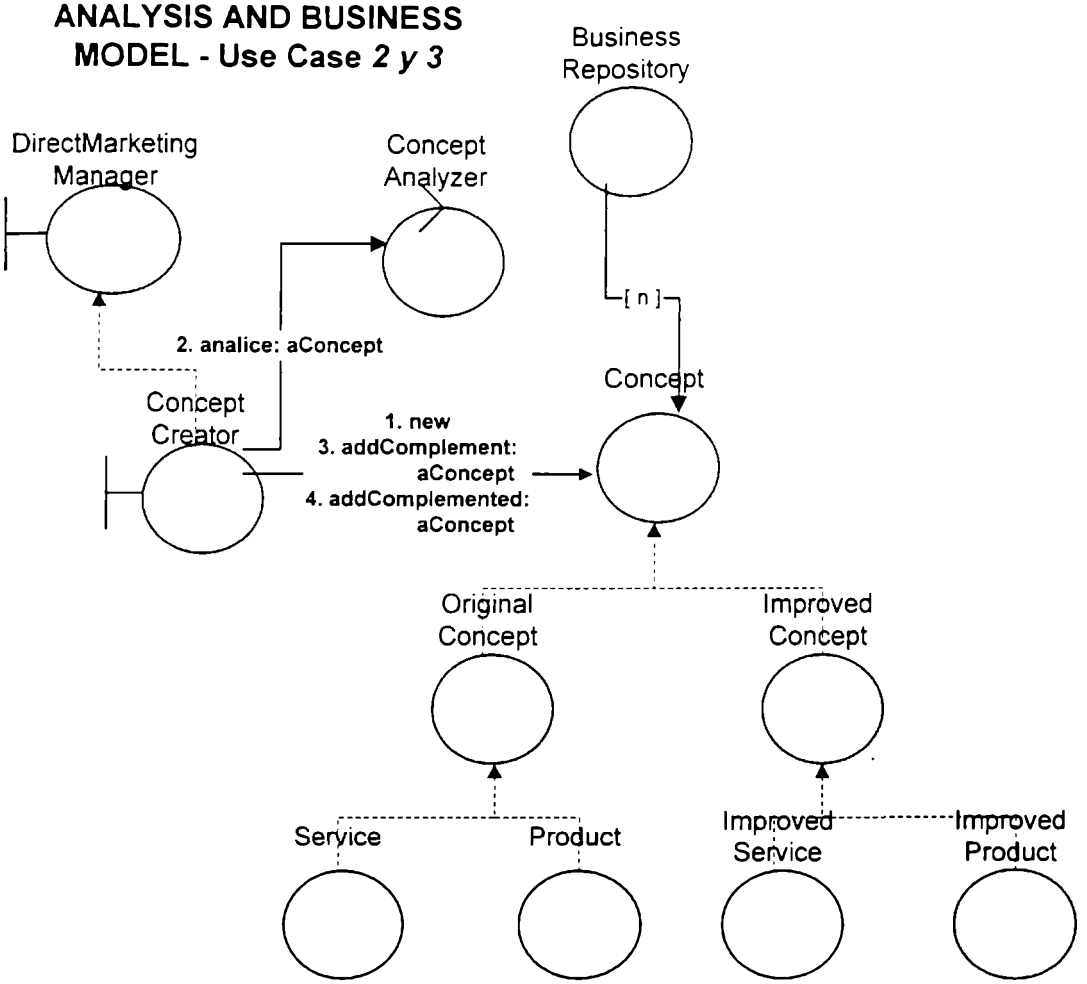
31. Incorporación de Nuevos Conocimientos a la BD
El operario consigue información adicional de clientes y asienta tales datos en la base de conocimiento instanciando los mismos como clientes posibles.
Tiene sentido incorporar tal información si cuenta al menos con la siguiente información:

Modelos de Análisis

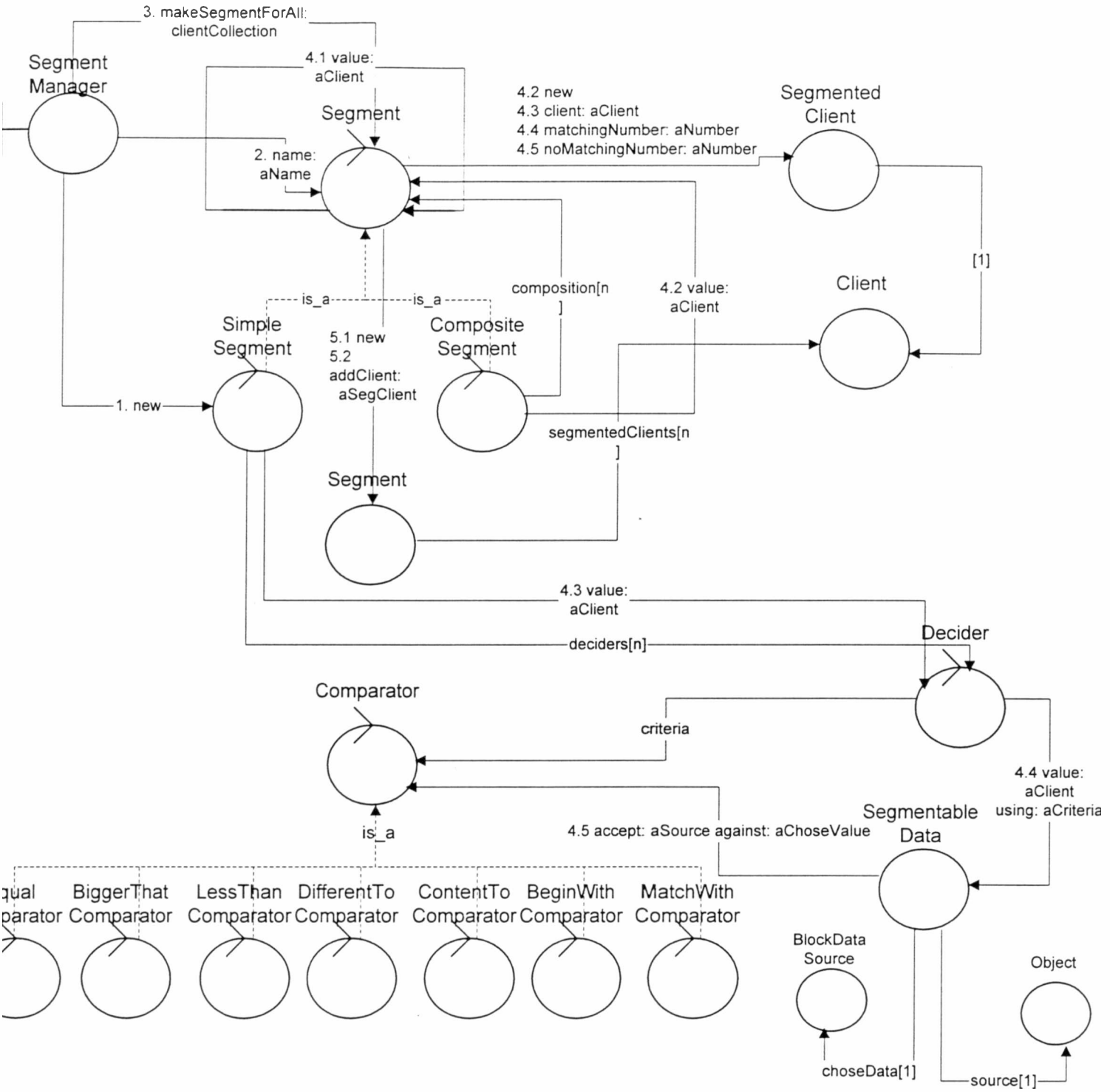
ANALYSIS AND BUSINESS MODEL - Use Case *1*



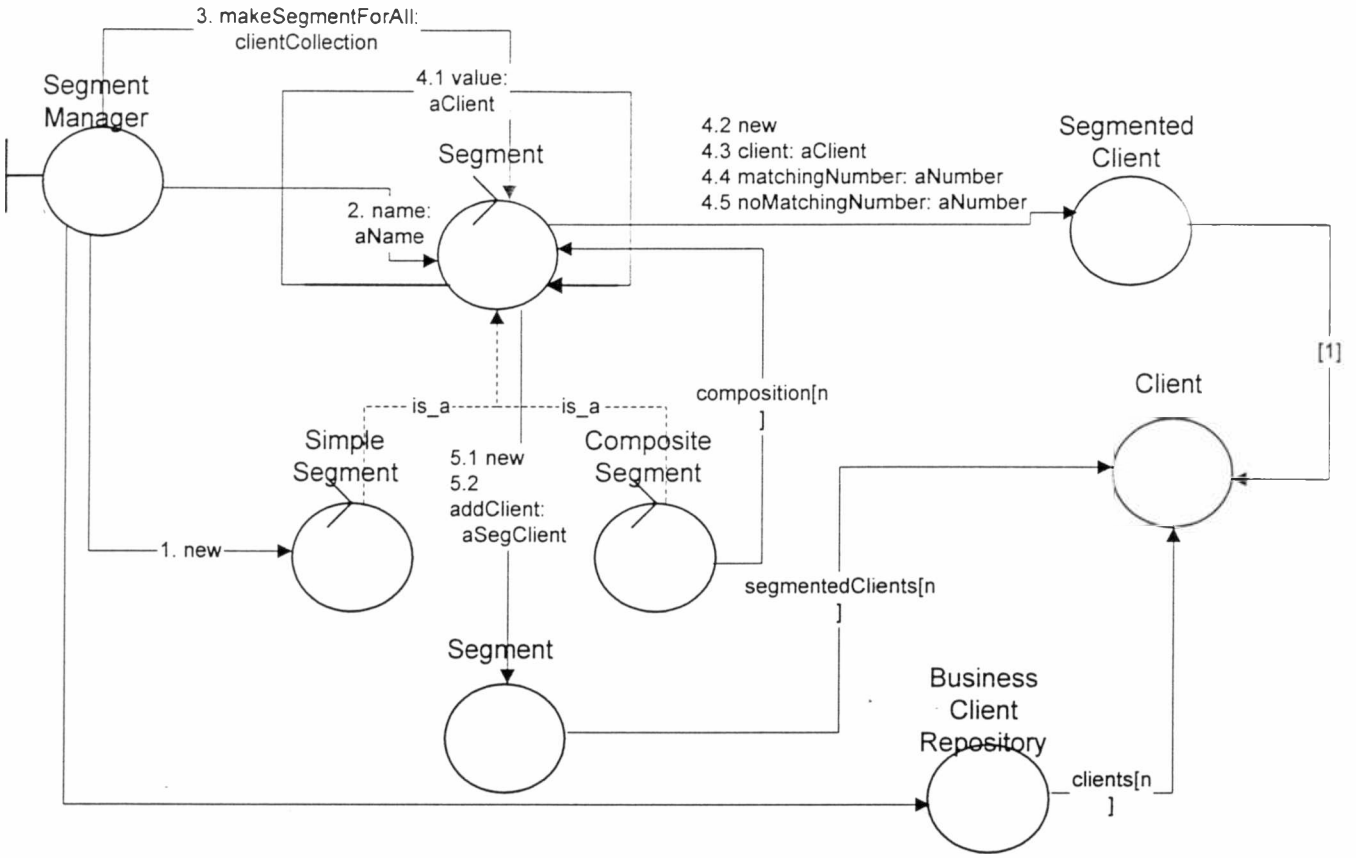
ANALYSIS AND BUSINESS MODEL - Use Case *2 y 3*



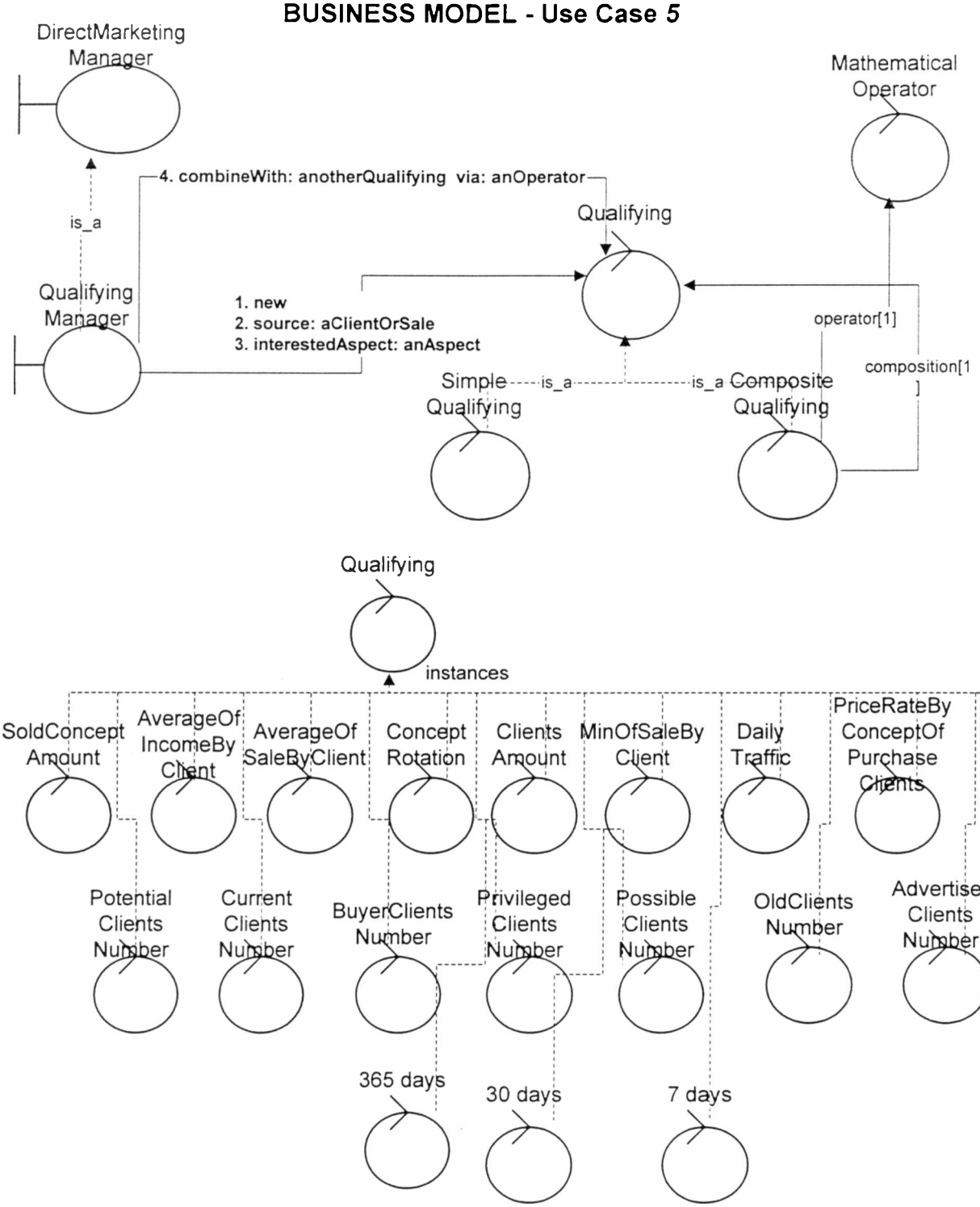
ANALYSIS MODEL - Use Case 4



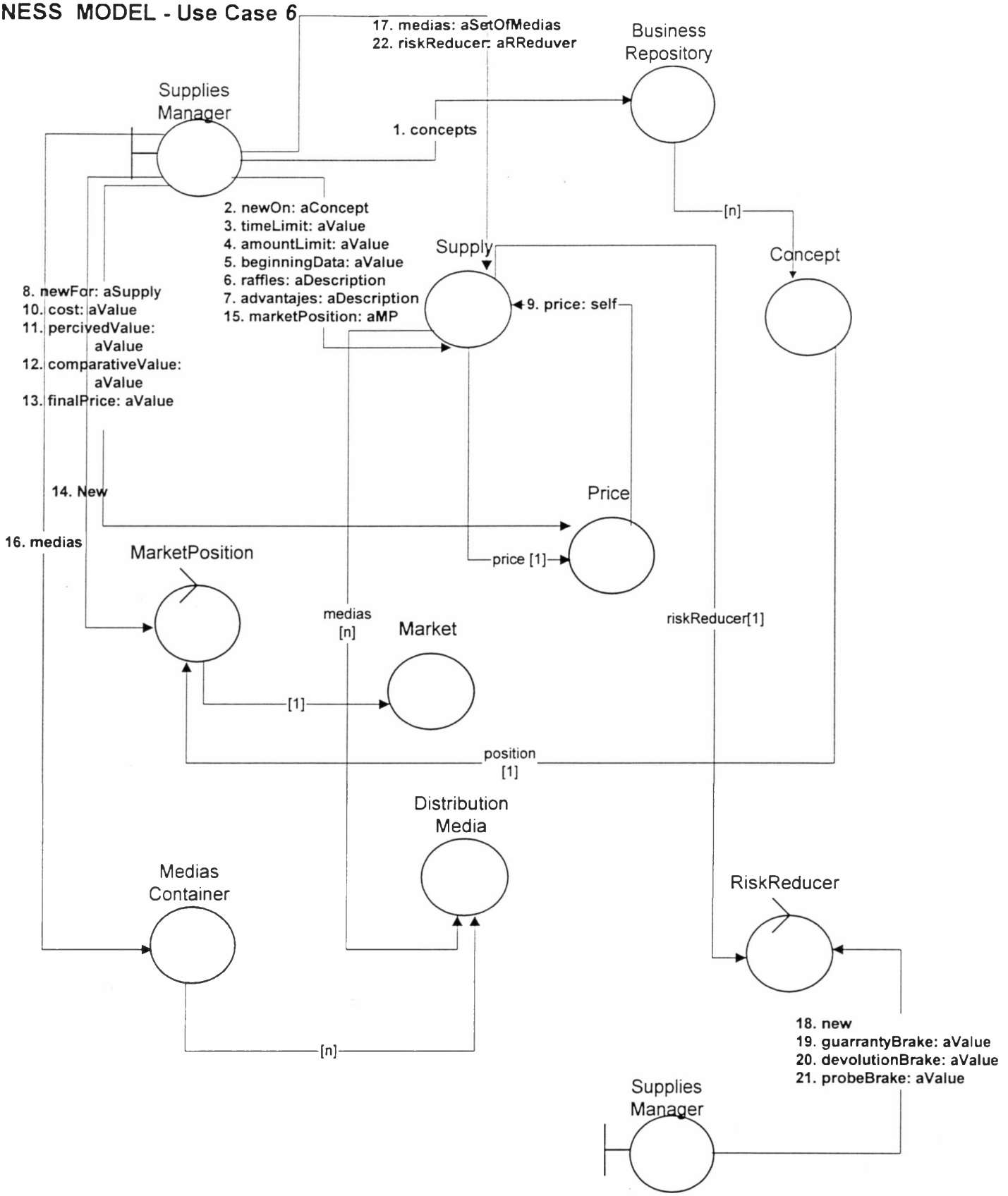
BUSINESS MODEL - Use Case *4*



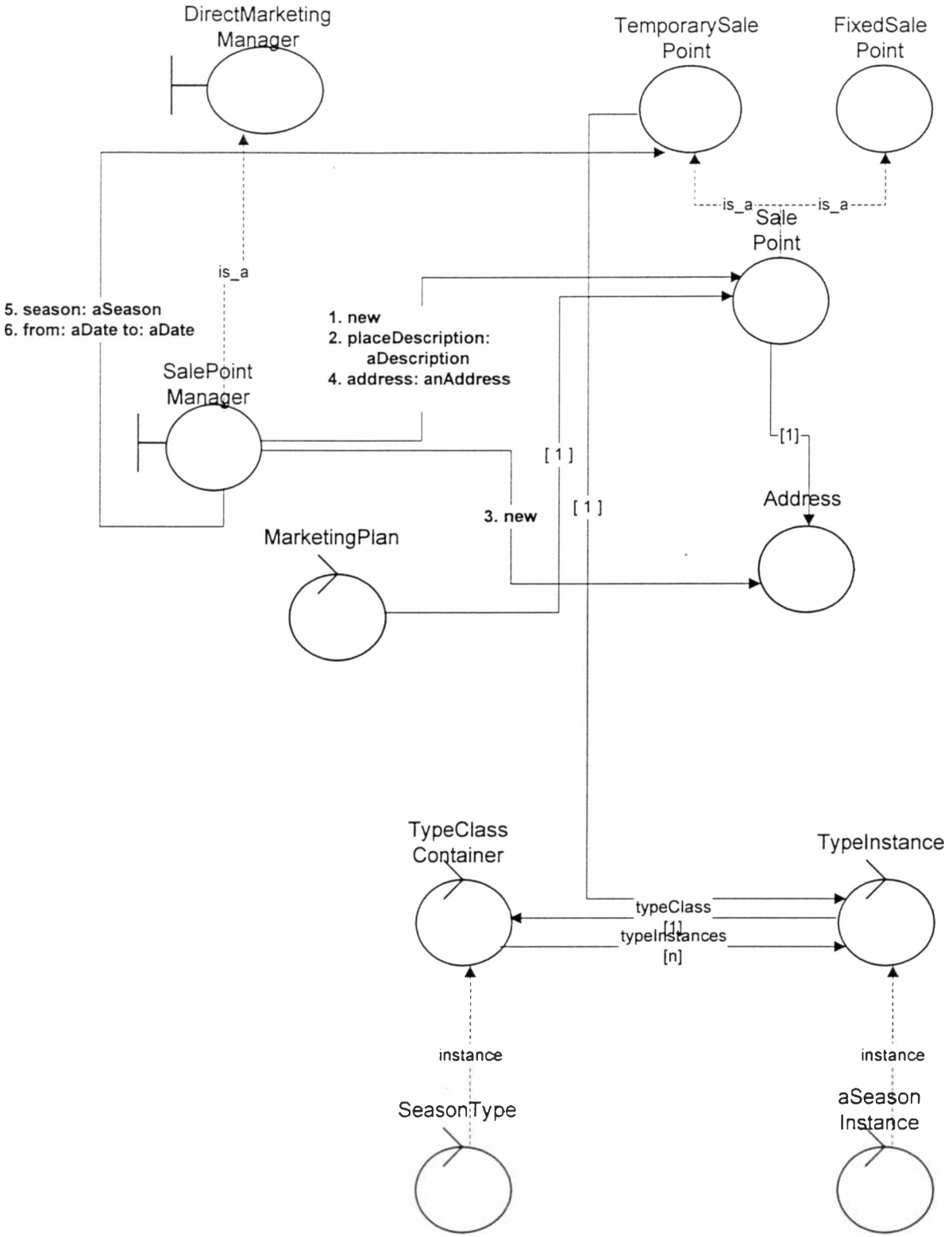
BUSINESS MODEL - Use Case *5*



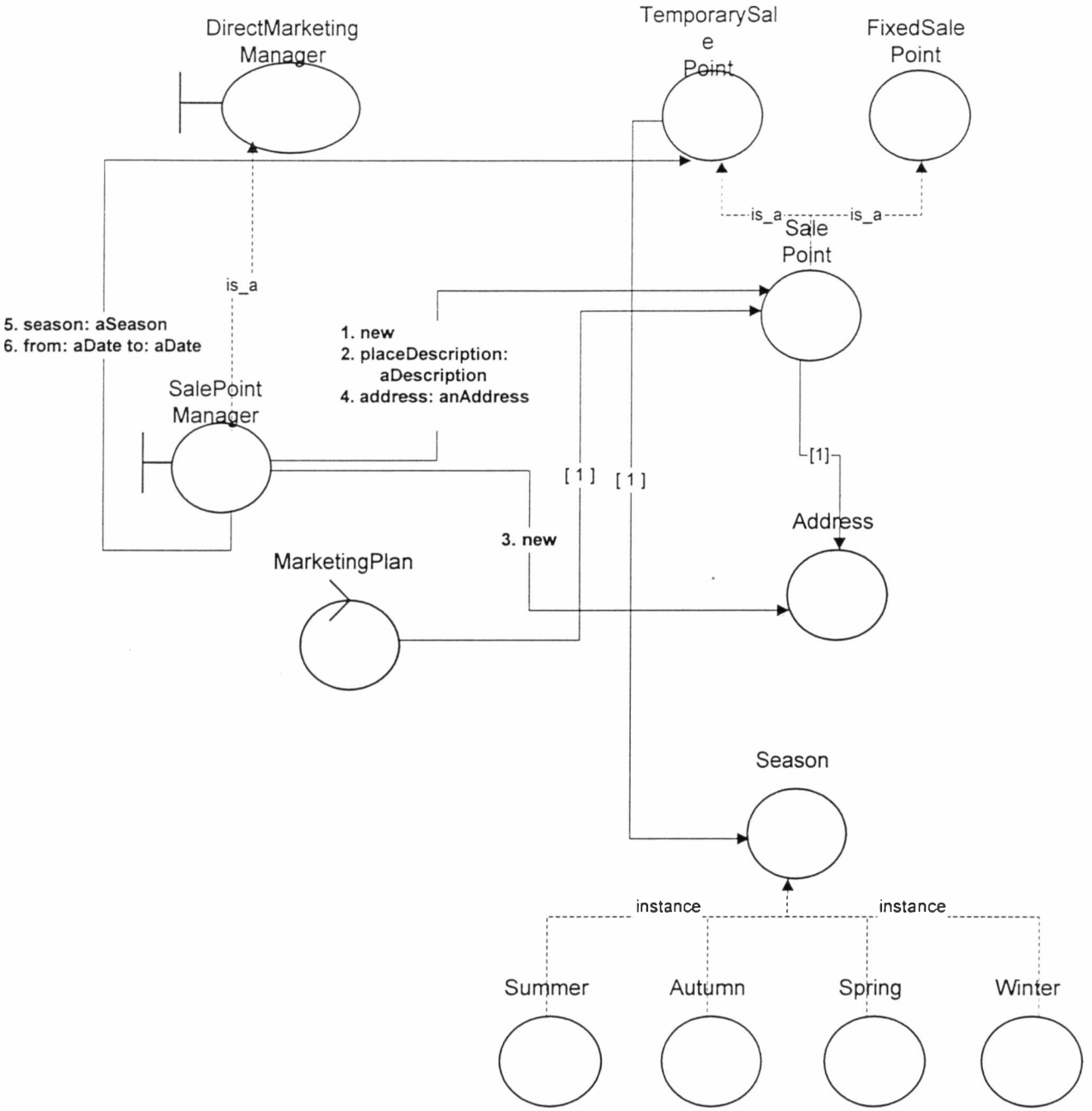
BUSINESS MODEL - Use Case 6



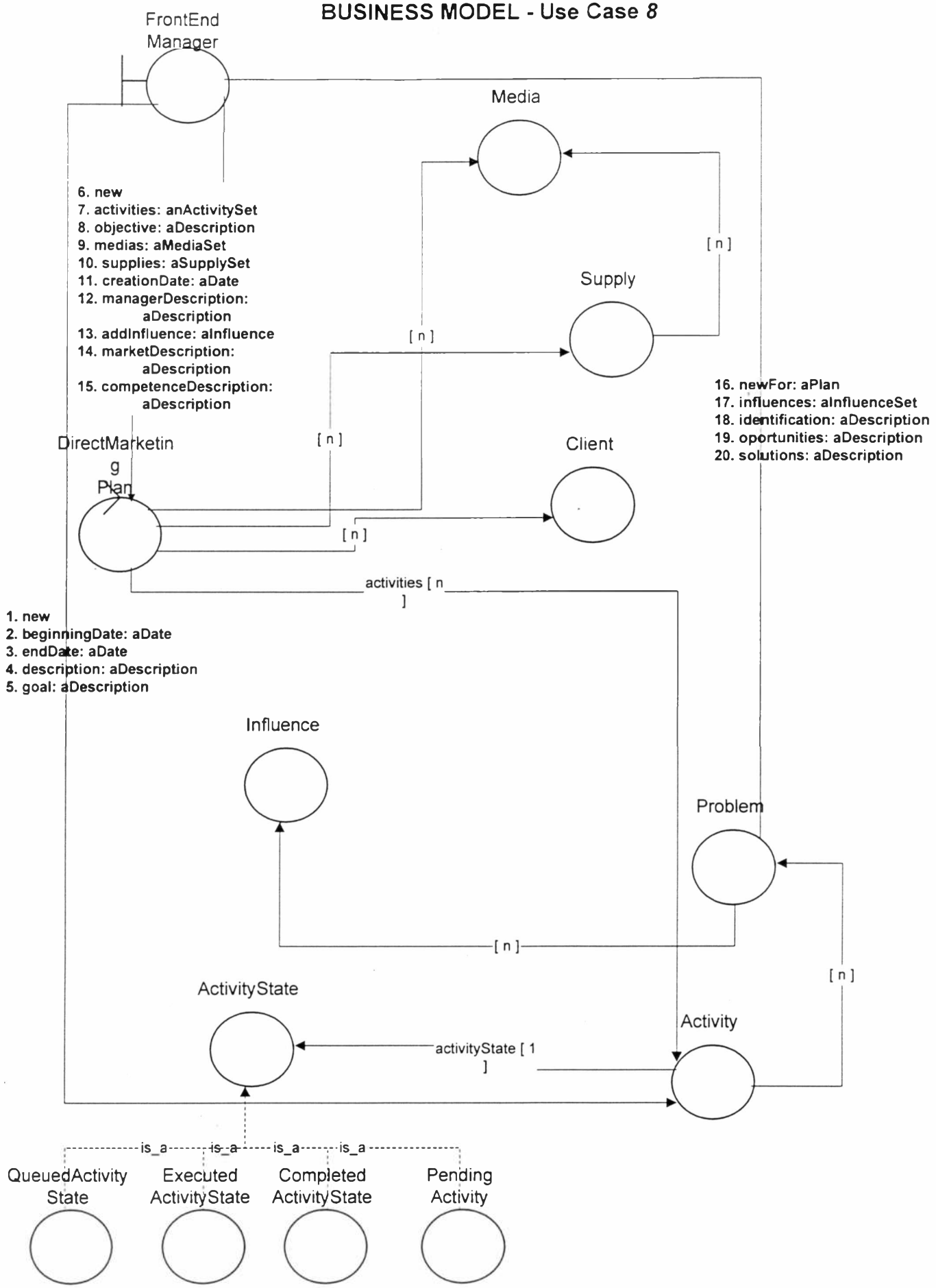
ANALYSIS MODEL - Use Case 7



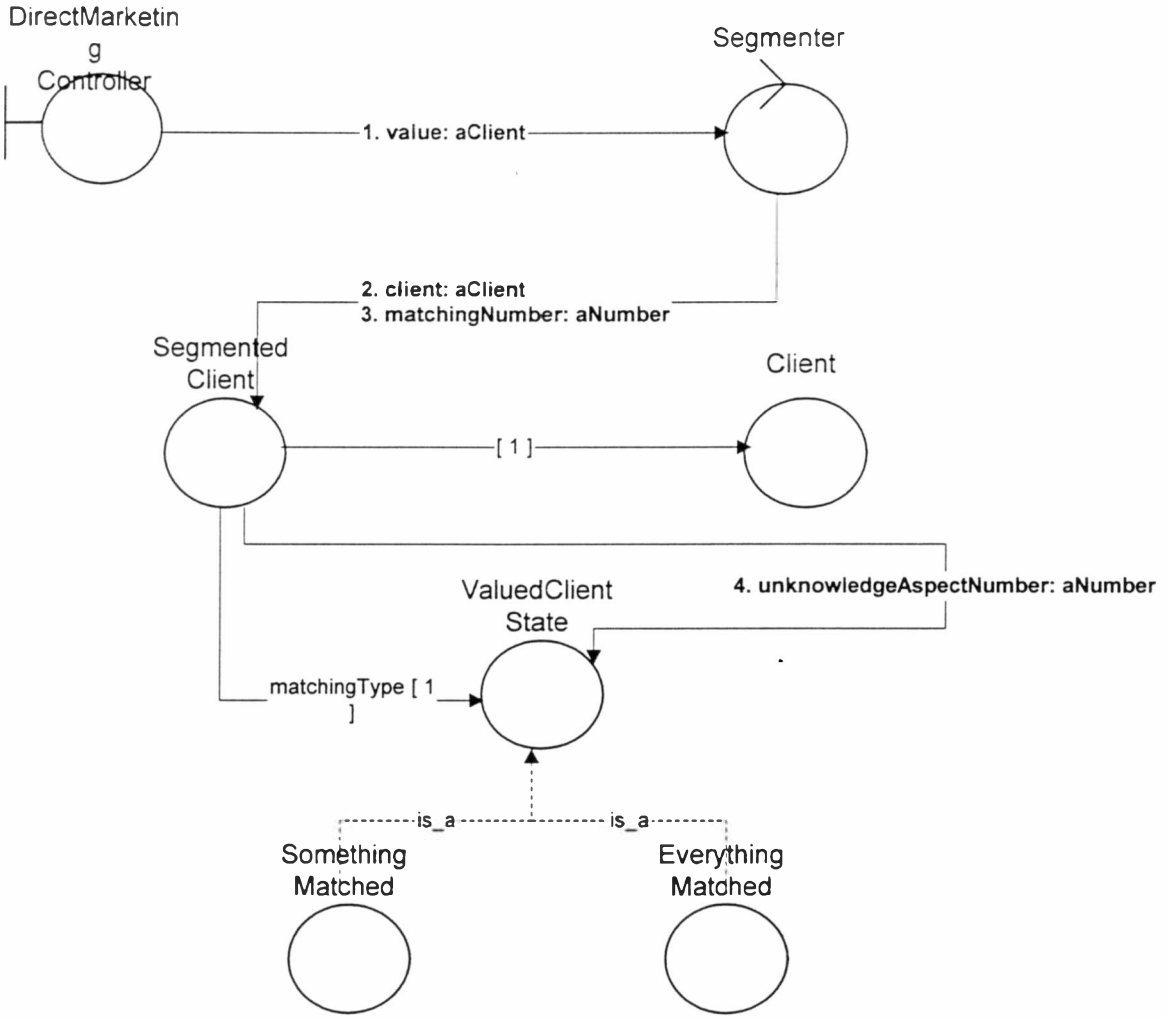
BUSINESS MODEL - Use Case 7



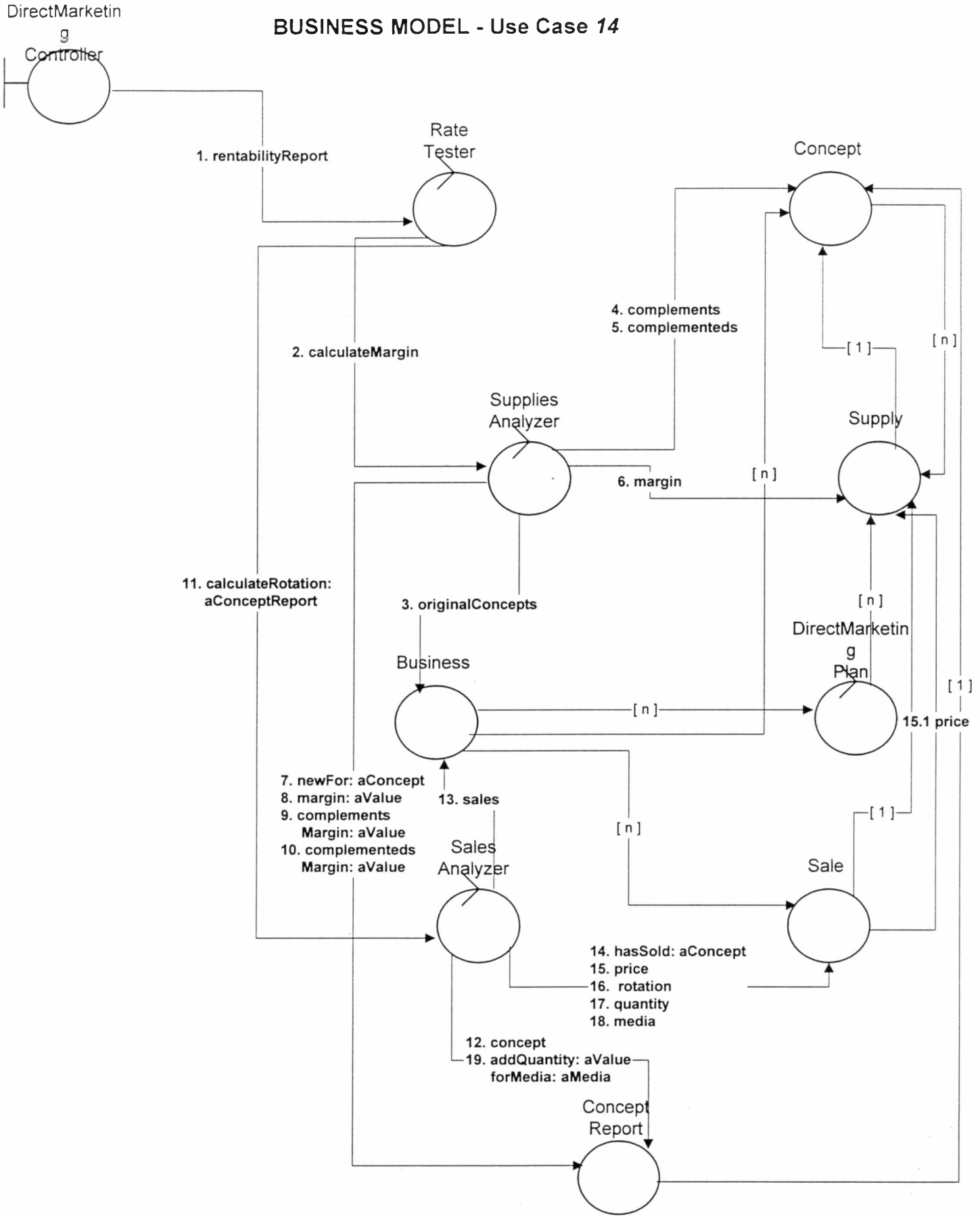
BUSINESS MODEL - Use Case *8*

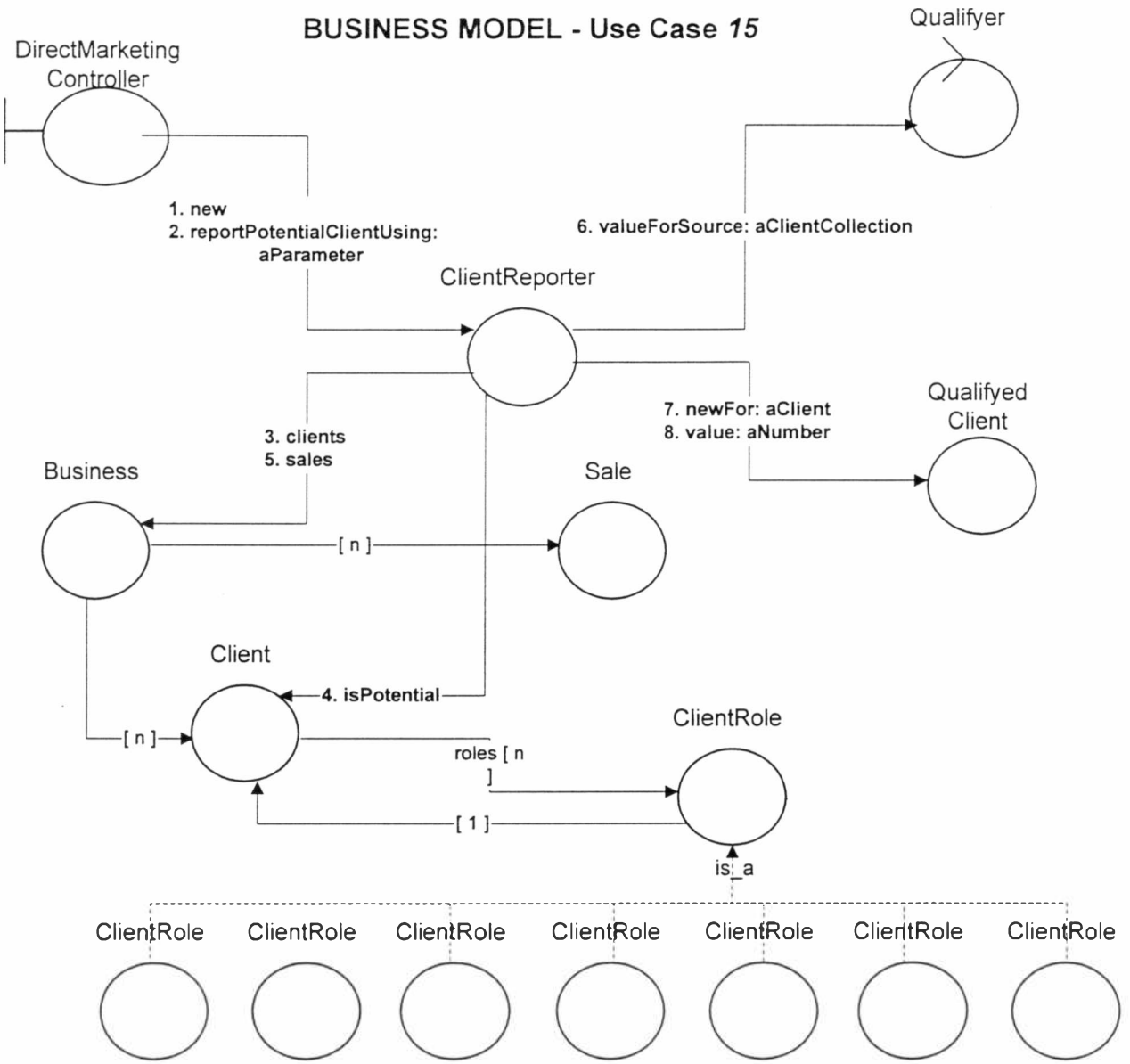


BUSINESS MODEL - Use Case 13

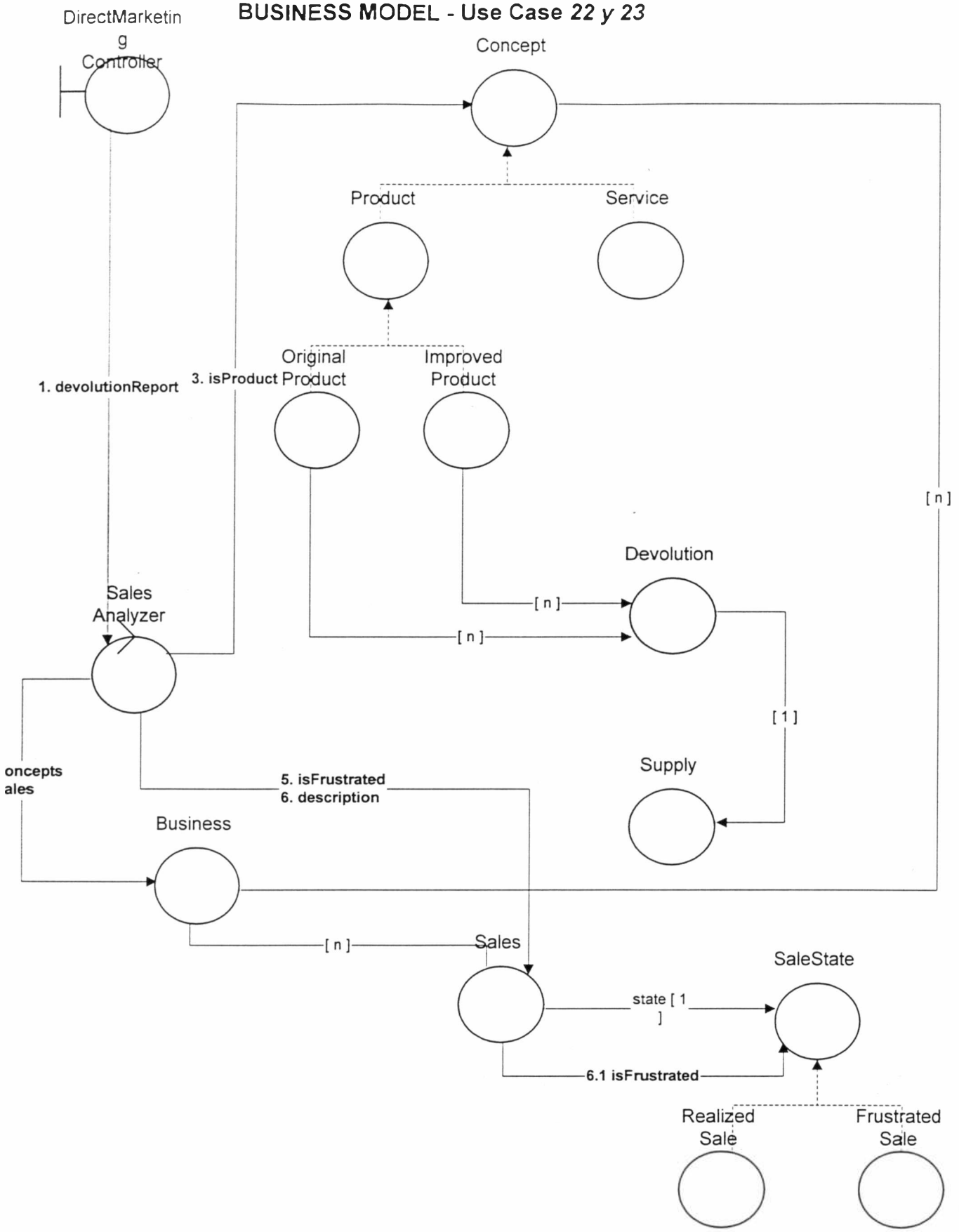


BUSINESS MODEL - Use Case *14*

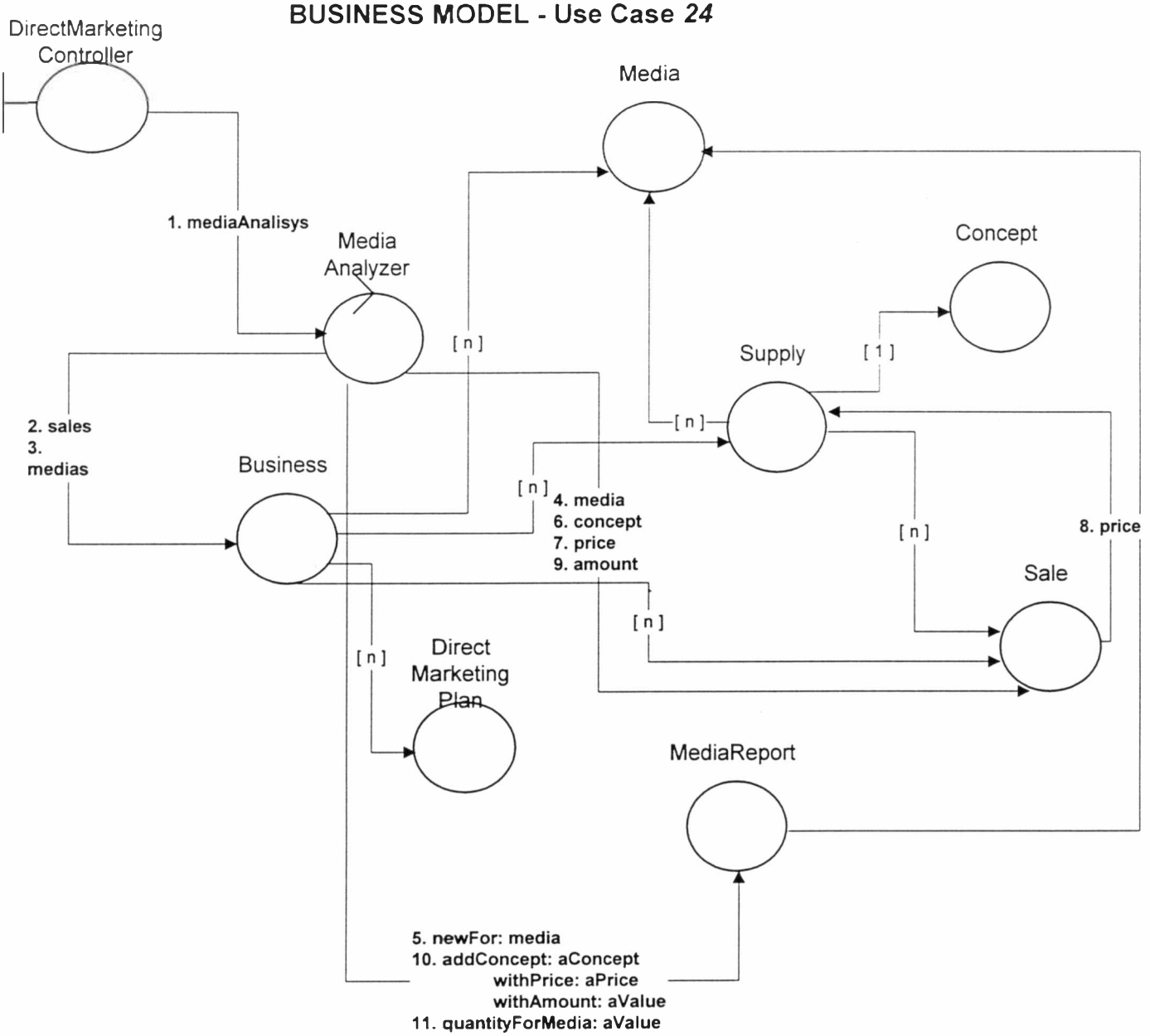



BUSINESS MODEL - Use Case 15
DirectMarketing Controller
Qualifyer
1. new
2. reportPotentialClientUsing: aParameter
6. valueForSource: aClientCollection
ClientReporter
3. clients
5. sales
7. newFor: aClient
8. value: aNumber
Qualifyed Client
Business
Sale
[n]
Client
4. isPotential
[n]
ClientRole
roles [n]
[1]
is_a
ClientRole
ClientRole
ClientRole
ClientRole
ClientRole
ClientRole
ClientRole

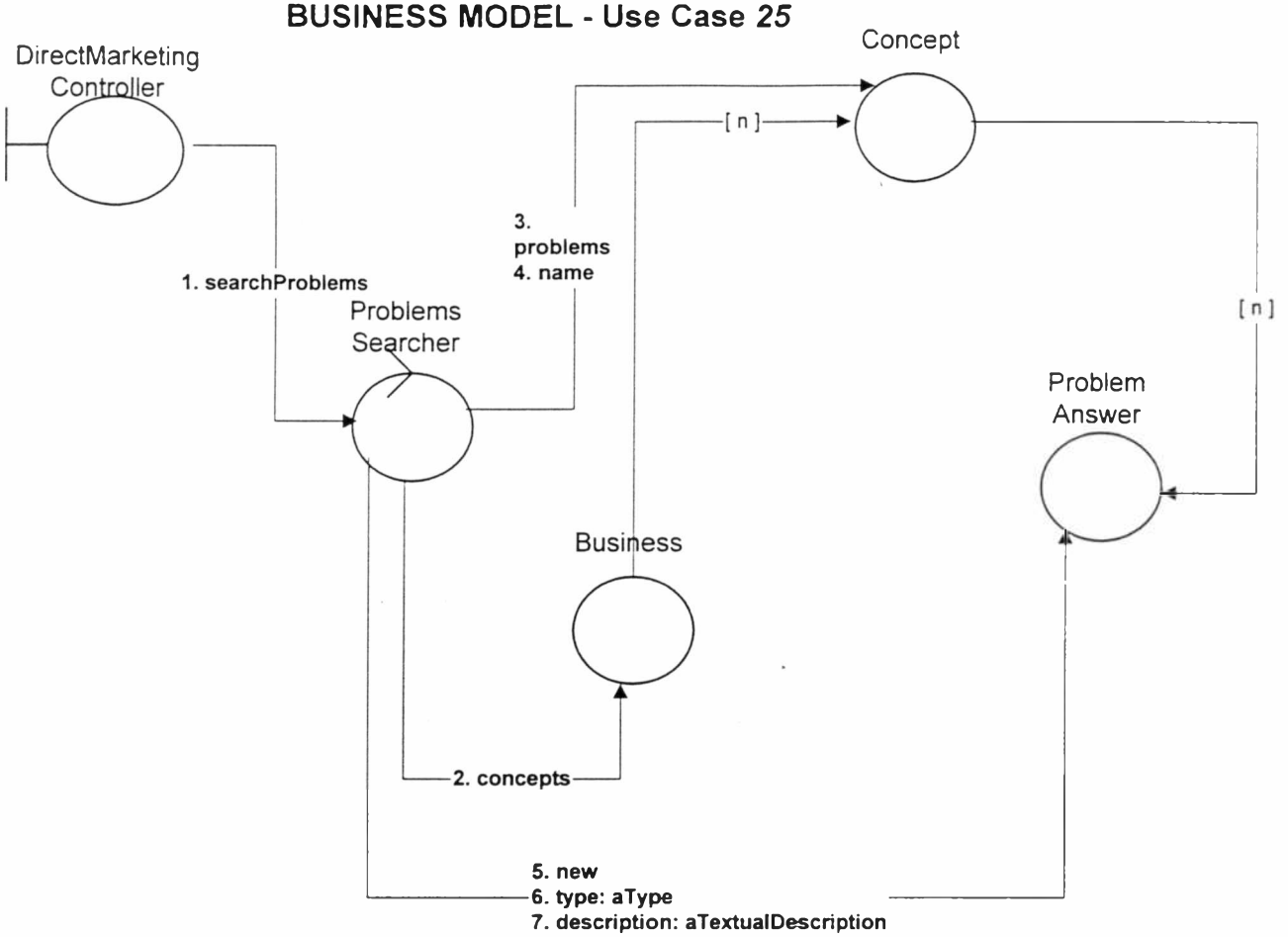
BUSINESS MODEL - Use Case *22 y 23*



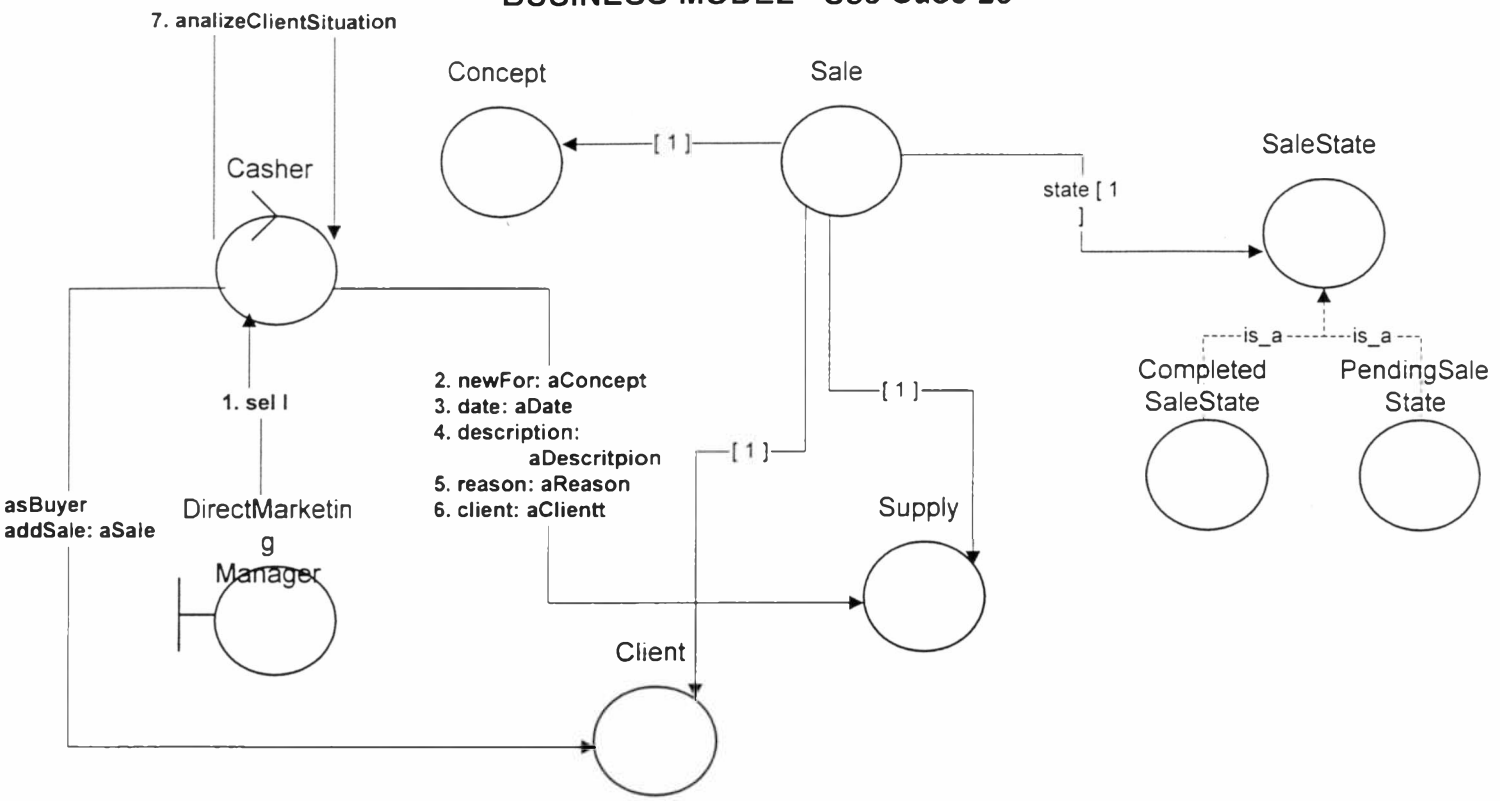
BUSINESS MODEL - Use Case *24*



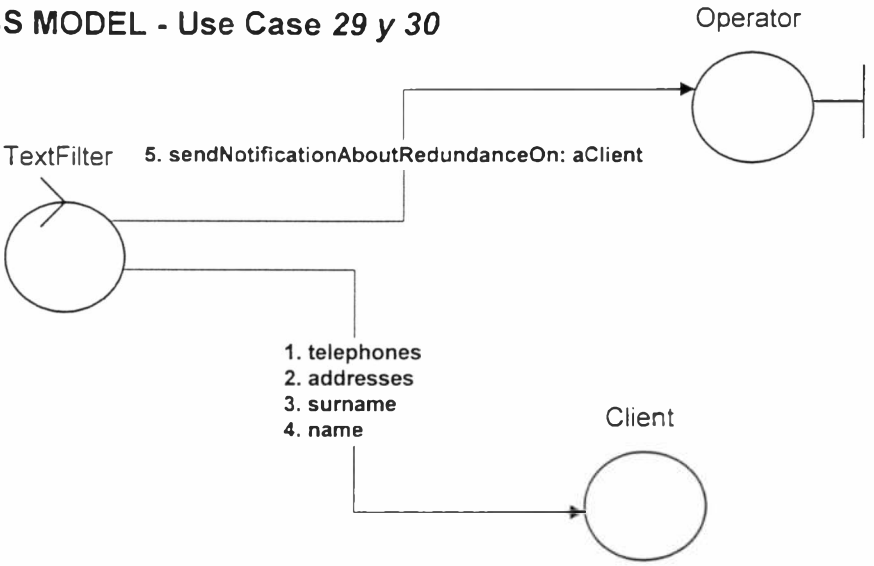
BUSINESS MODEL - Use Case *25*



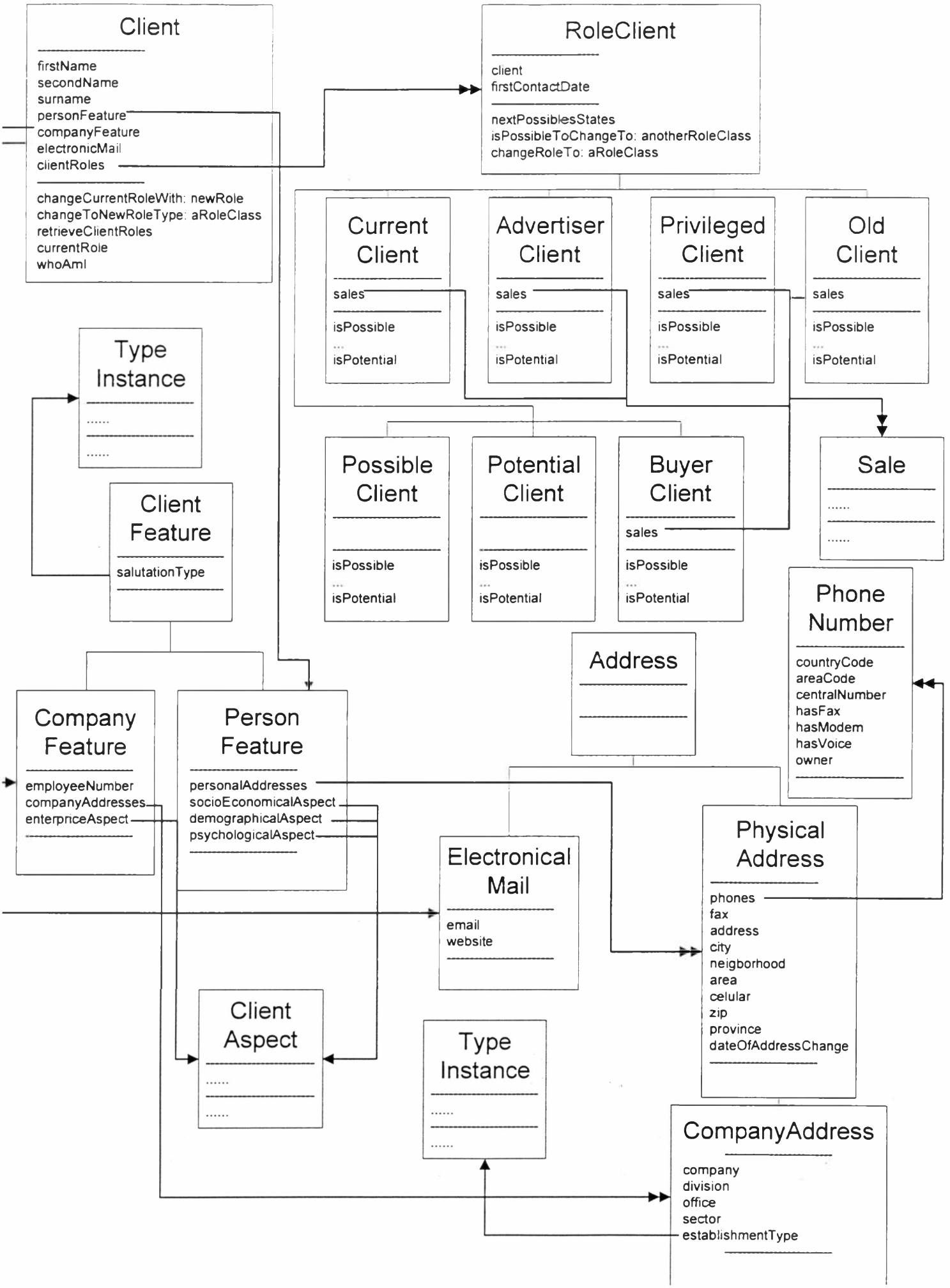
BUSINESS MODEL - Use Case *28*

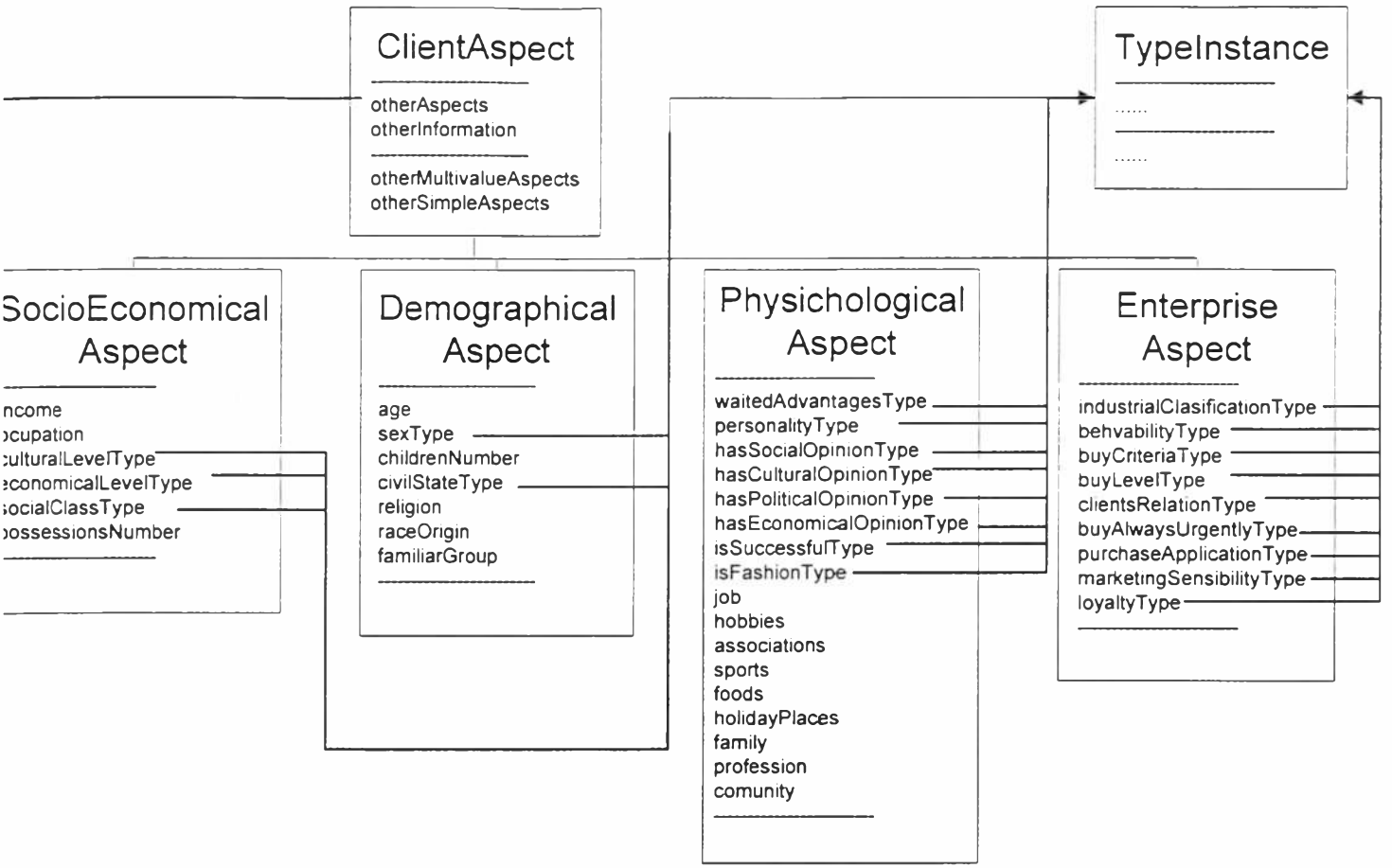


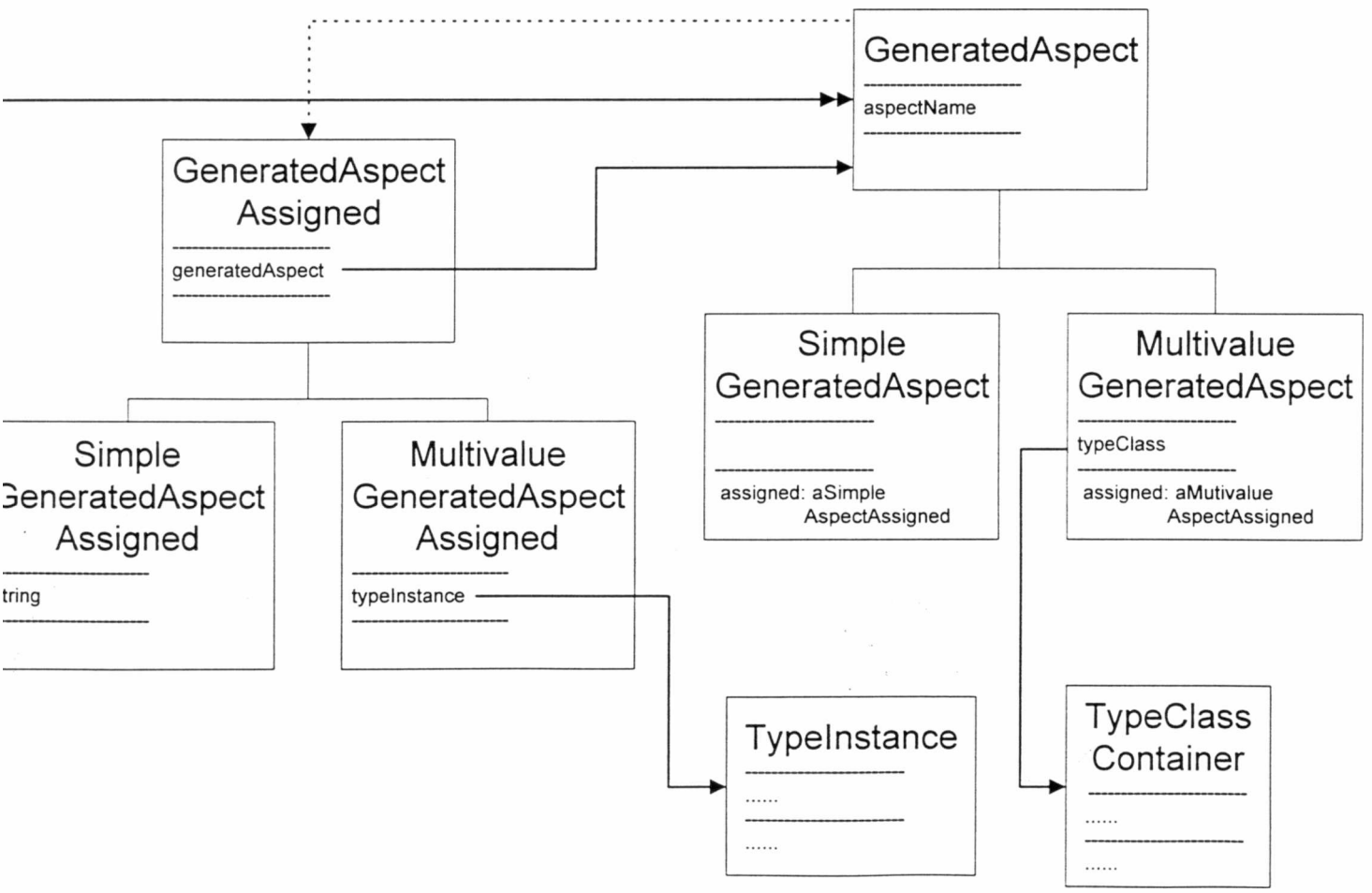
BUSINESS MODEL - Use Case *29 y 30*

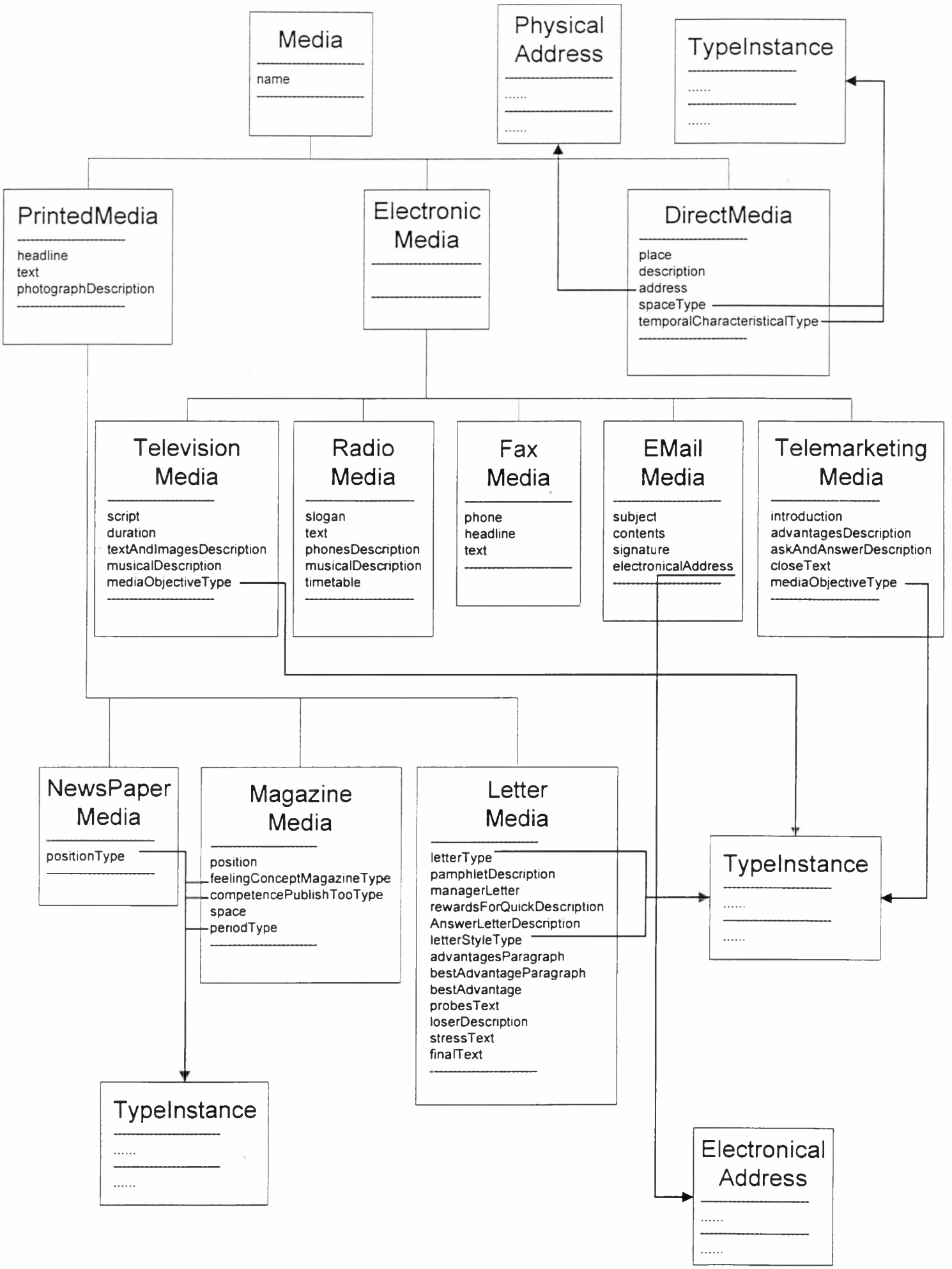


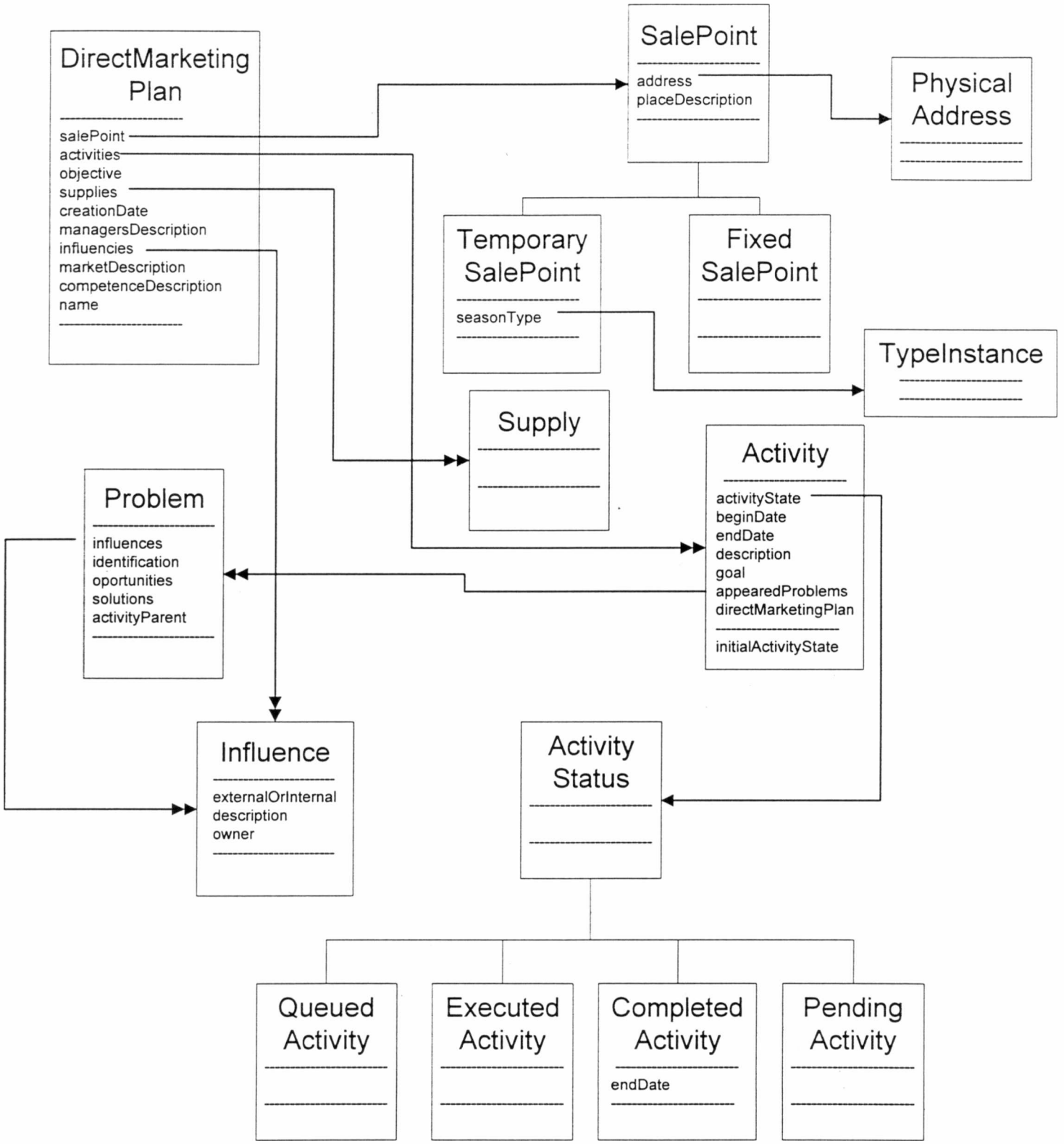
Modelos de Diseño e Implementación

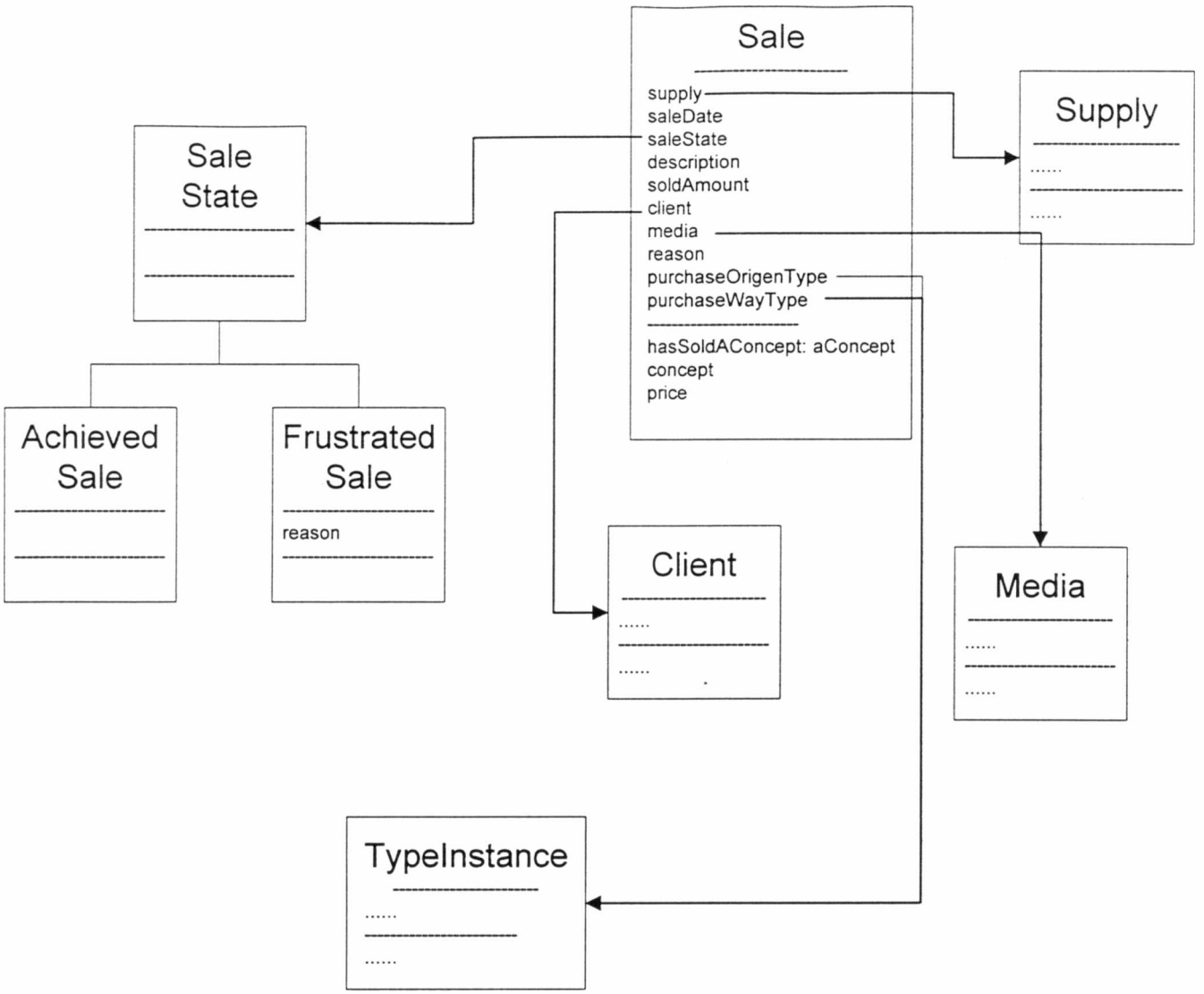

Client
firstName
secondName
surname
personFeature
companyFeature
electronicMail
clientRoles
changeCurrentRoleWith: newRole
changeToNewRoleType: aRoleClass
retrieveClientRoles
currentRole
whoAmI
RoleClient
client
firstContactDate
nextPossiblesStates
isPossibleToChangeTo: anotherRoleClass
changeRoleTo: aRoleClass
Current Client
sales
isPossible
isPotential
Advertiser Client
sales
isPossible
isPotential
Privileged Client
sales
isPossible
isPotential
Old Client
sales
isPossible
isPotential
Type Instance
Client Feature
salutationType
Possible Client
isPossible
isPotential
Potential Client
isPossible
isPotential
Buyer Client
sales
isPossible
isPotential
Sale
Phone Number
countryCode
areaCode
centralNumber
hasFax
hasModem
hasVoice
owner
Address
Company Feature
employeeNumber
companyAddresses
enterpriceAspect
Person Feature
personalAddresses
socioEconomicalAspect
demographicalAspect
psychologicalAspect
Electronical Mail
email
website
Physical Address
phones
fax
address
city
neigborhood
area
celular
zip
province
dateOfAddressChange
Client Aspect
Type Instance
CompanyAddress
company
division
office
sector
establishmentType

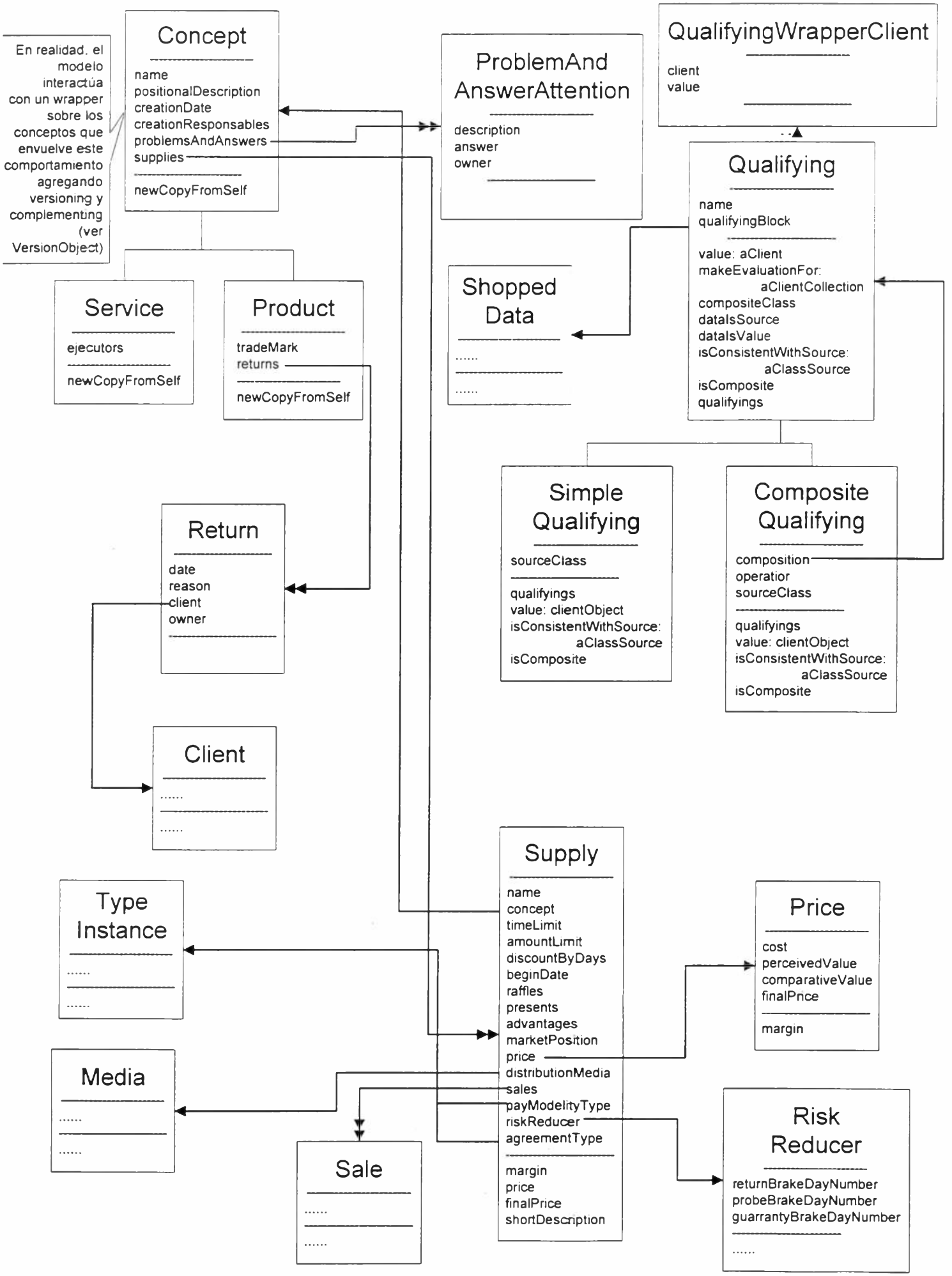

ClientAspect
otherAspects
otherInformation
otherMultivalueAspects
otherSimpleAspects
TypeInstance
.....
.....
SocioEconomical Aspect
ncome
ocupation
culturalLevelType
economicalLevelType
ocialClassType
ossessionsNumber
Demographical Aspect
age
sexType
childrenNumber
civilStateType
religion
raceOrigin
familiarGroup
Physichological Aspect
waitedAdvantagesType
personalityType
hasSocialOpinionType
hasCulturalOpinionType
hasPoliticalOpinionType
hasEconomicalOpinionType
isSuccessfulType
isFashionType
job
hobbies
associations
sports
foods
holidayPlaces
family
profession
comunity
Enterprise Aspect
industrialClasificationType
behvabilityType
buyCriteriaType
buyLevelType
clientsRelationType
buyAlwaysUrgentlyType
purchaseApplicationType
marketingSensibilityType
loyaltyType

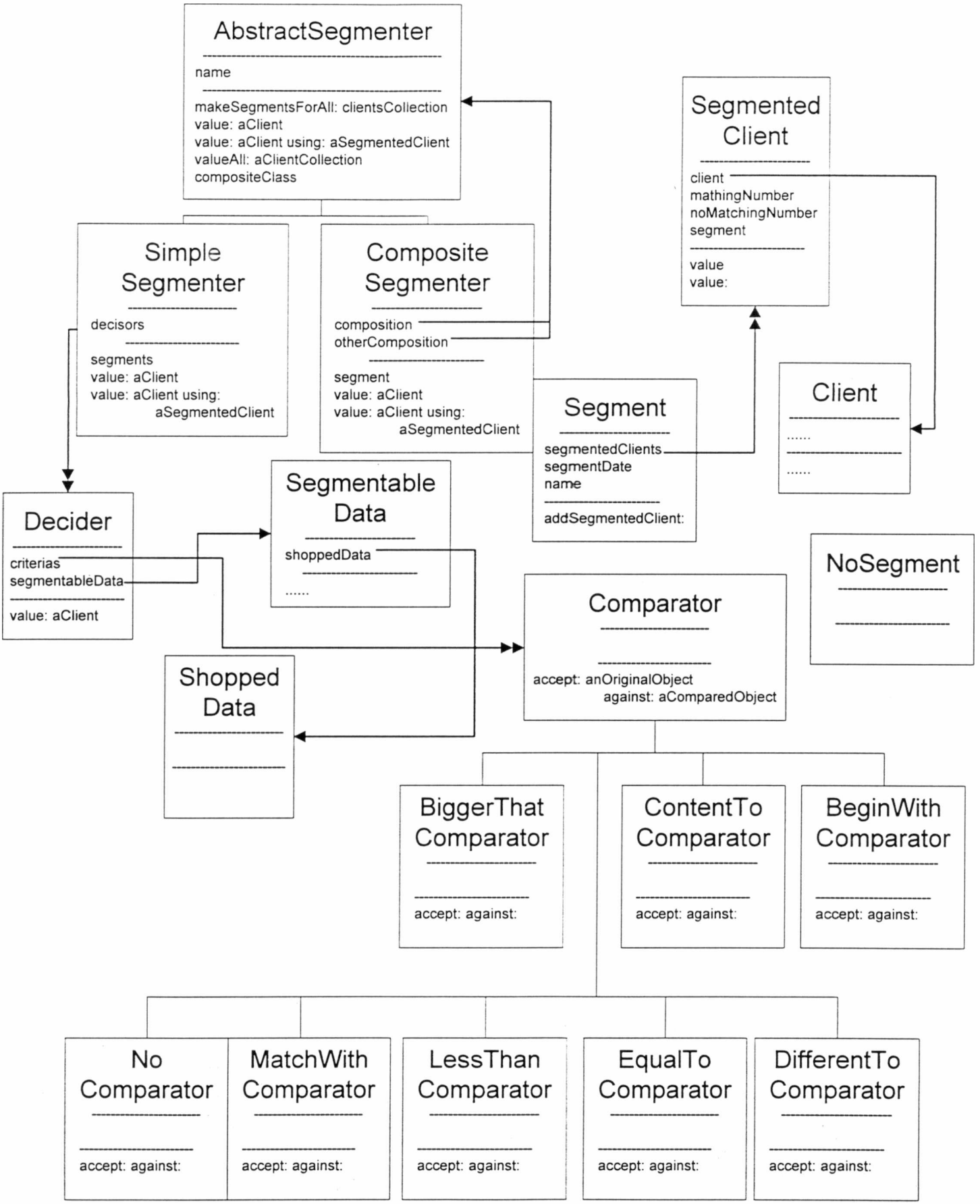

GeneratedAspect
aspectName
GeneratedAspect Assigned
generatedAspect
Simple GeneratedAspect
assigned: aSimple AspectAssigned
Multivalue GeneratedAspect
typeClass
assigned: aMutivalue AspectAssigned
Simple GeneratedAspect Assigned
tring
Multivalue GeneratedAspect Assigned
typeInstance
TypeInstance
.....
.....
TypeClass Container
.....
.....

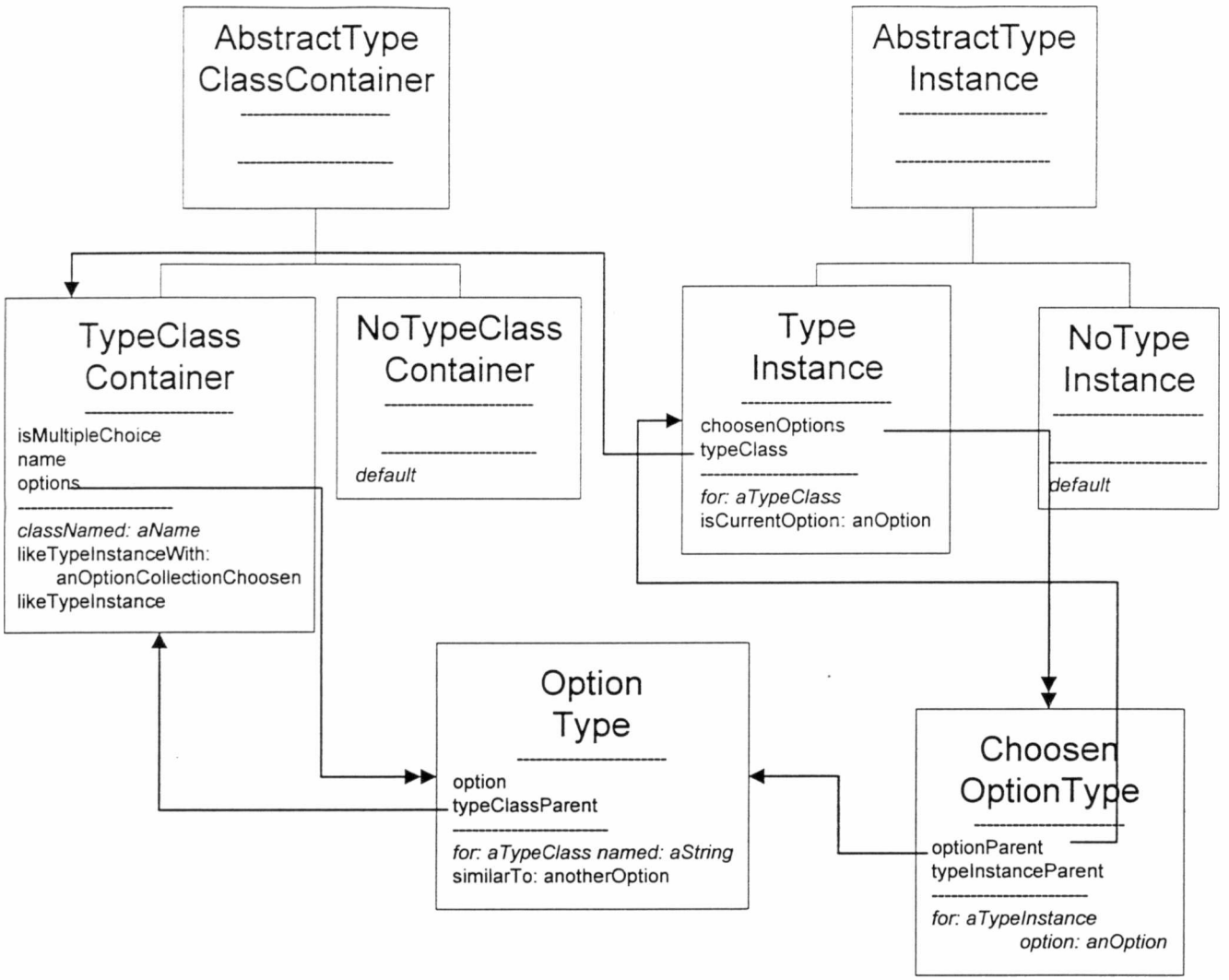

Media
name
Physical Address
.....
.....
TypeInstance
.....
.....
PrintedMedia
headline
text
photographDescription
Electronic Media
DirectMedia
place
description
address
spaceType
temporalCharacteristicalType
Television Media
script
duration
textAndImagesDescription
musicalDescription
mediaObjectiveType
Radio Media
slogan
text
phonesDescription
musicalDescription
timetable
Fax Media
phone
headline
text
EMail Media
subject
contents
signature
electronicalAddress
Telemarketing Media
introduction
advantagesDescription
askAndAnswerDescription
closeText
mediaObjectiveType
NewsPaper Media
positionType
Magazine Media
position
feelingConceptMagazineType
competencePublishTooType
space
periodType
Letter Media
letterType
pamphletDescription
managerLetter
rewardsForQuickDescription
AnswerLetterDescription
letterStyleType
advantagesParagraph
bestAdvantageParagraph
bestAdvantage
probesText
loserDescription
stressText
finalText
TypeInstance
.....
.....
TypeInstance
.....
.....
Electronical Address
.....
.....

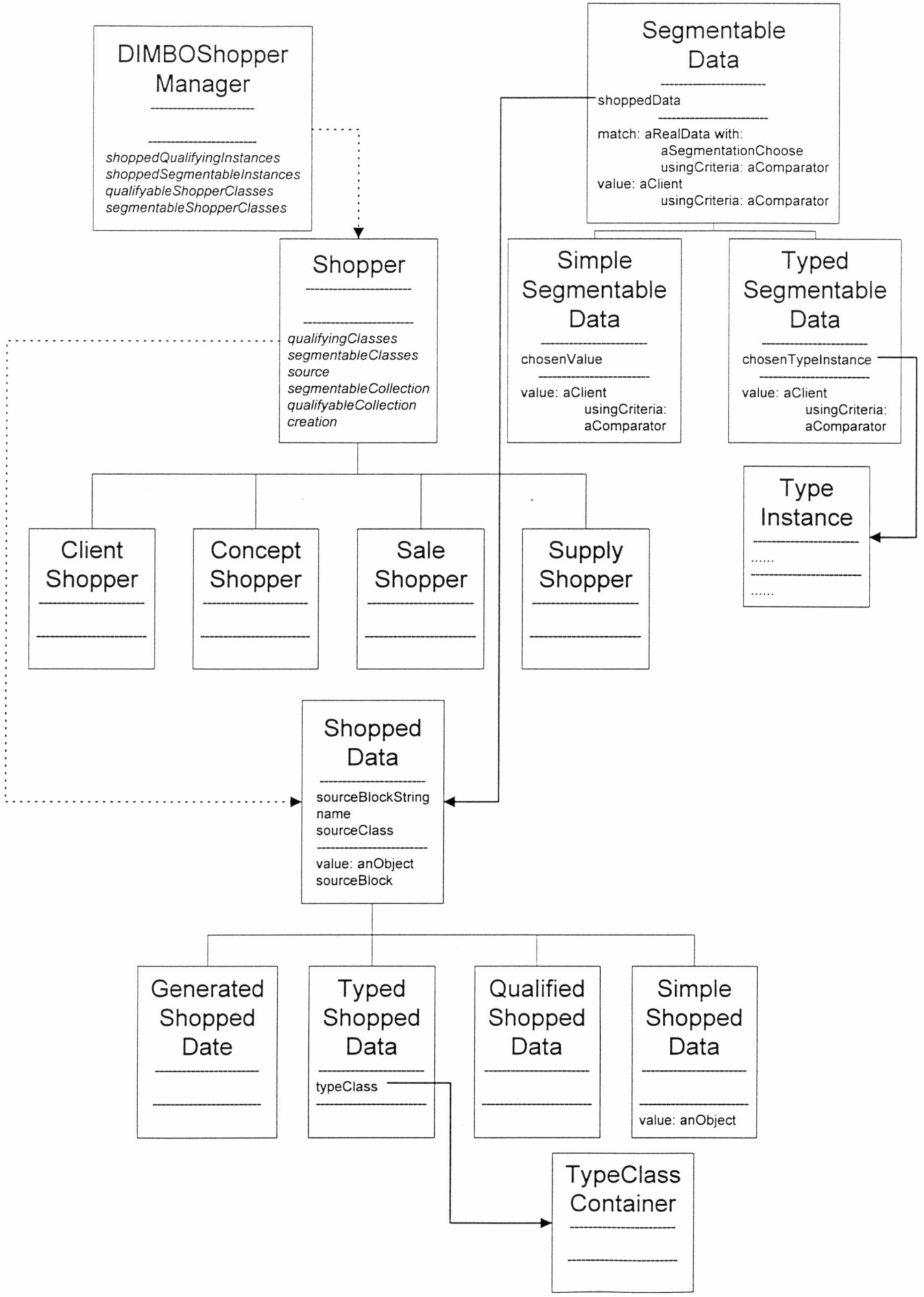

DirectMarketing Plan
salePoint
activities
objective
supplies
creationDate
managersDescription
influencies
marketDescription
competenceDescription
name
SalePoint
address
placeDescription
Physical Address
Temporary SalePoint
seasonType
Fixed SalePoint
TypeInstance
Supply
Activity
activityState
beginDate
endDate
description
goal
appearedProblems
directMarketingPlan
initialActivityState
Problem
influences
identification
oportunities
solutions
activityParent
Influence
externalOrInternal
description
owner
Activity Status
Queued Activity
Executed Activity
Completed Activity
endDate
Pending Activity

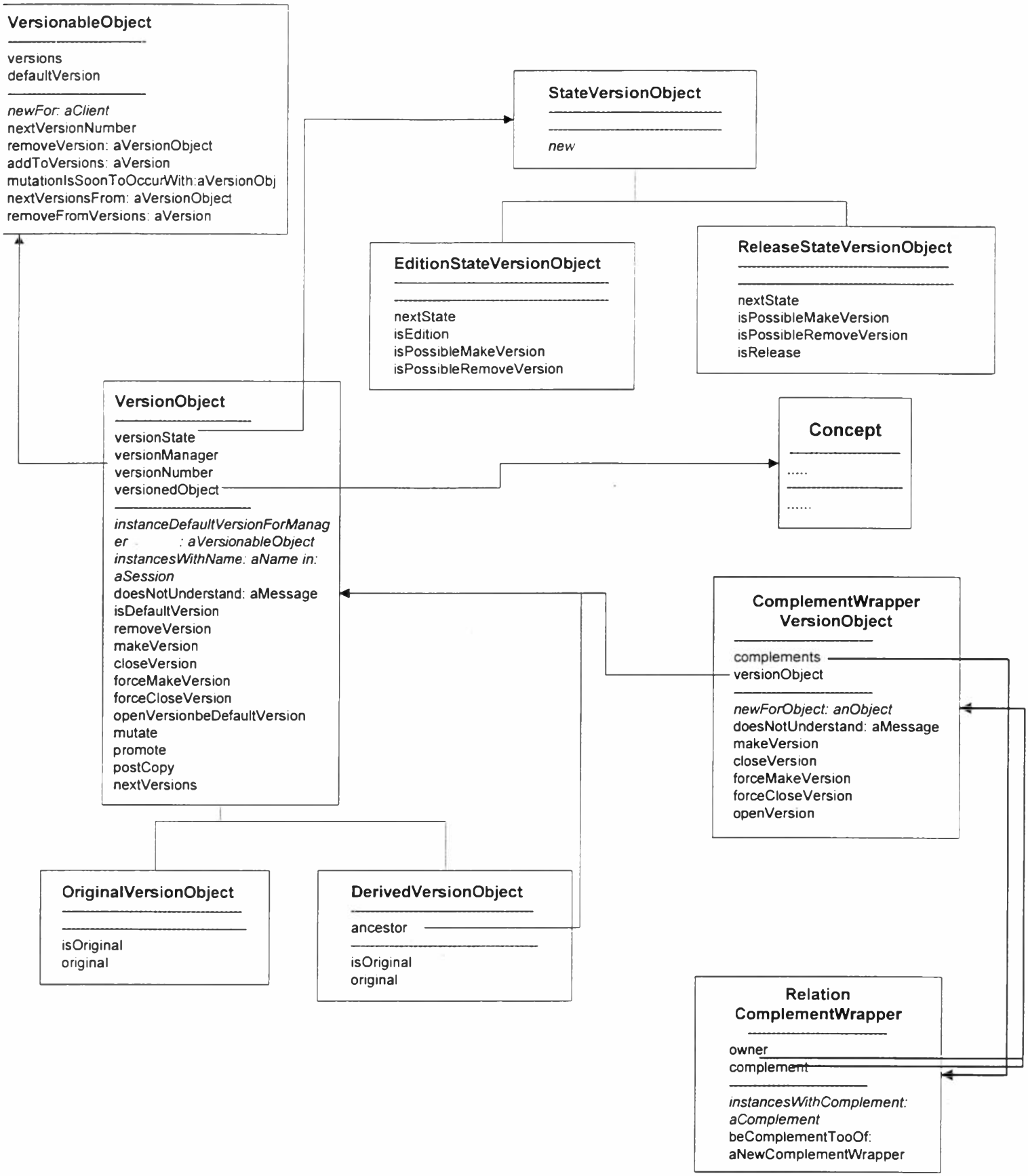

Sale
supply
saleDate
saleState
description
soldAmount
client
media
reason
purchaseOrigenType
purchaseWayType
hasSoldAConcept: aConcept
concept
price
Supply
.....
.....
Sale State
Achieved Sale
Frustrated Sale
reason
Client
.....
.....
Media
.....
.....
TypeInstance
.....
.....


En realidad. el modelo interactúa con un wrapper sobre los conceptos que envuelve este comportamiento agregando versioning y complementing (ver VersionObject)
Concept
name
positionalDescription
creationDate
creationResponsables
problemsAndAnswers
supplies
newCopyFromSelf
ProblemAnd AnswerAttention
description
answer
owner
QualifyingWrapperClient
client
value
Qualifying
name
qualifyingBlock
value: aClient
makeEvaluationFor: aClientCollection
compositeClass
dataIsSource
dataIsValue
isConsistentWithSource: aClassSource
isComposite
qualifyings
Service
ejecutors
newCopyFromSelf
Product
tradeMark
returns
newCopyFromSelf
Shopped Data
.....
.....
Simple Qualifying
sourceClass
qualifyings
value: clientObject
isConsistentWithSource: aClassSource
isComposite
Composite Qualifying
composition
operatior
sourceClass
qualifyings
value: clientObject
isConsistentWithSource: aClassSource
isComposite
Return
date
reason
client
owner
Client
.....
.....
Supply
name
concept
timeLimit
amountLimit
discountByDays
beginDate
raffles
presents
advantages
marketPosition
price
distributionMedia
sales
payModelityType
riskReducer
agreementType
margin
price
finalPrice
shortDescription
Price
cost
perceivedValue
comparativeValue
finalPnce
margin
Type Instance
.....
.....
Media
.....
.....
Sale
.....
.....
Risk Reducer
returnBrakeDayNumber
probeBrakeDayNumber
guarrantyBrakeDayNumber
.....


AbstractSegmenter
name
makeSegmentsForAll: clientsCollection
value: aClient
value: aClient using: aSegmentedClient
valueAll: aClientCollection
compositeClass
Segmented Client
client
mathingNumber
noMatchingNumber
segment
value
value:
Simple Segmenter
decisors
segments
value: aClient
value: aClient using:
aSegmentedClient
Composite Segmenter
composition
otherComposition
segment
value: aClient
value: aClient using:
aSegmentedClient
Segment
segmentedClients
segmentDate
name
addSegmentedClient:
Client
.....
.....
Decider
criterias
segmentableData
value: aClient
Segmentable Data
shoppedData
.....
NoSegment
Comparator
accept: anOriginalObject
against: aComparedObject
Shopped Data
BiggerThat Comparator
accept: against:
ContentTo Comparator
accept: against:
BeginWith Comparator
accept: against:
No Comparator
accept: against:
MatchWith Comparator
accept: against:
LessThan Comparator
accept: against:
EqualTo Comparator
accept: against:
DifferentTo Comparator
accept: against:


AbstractType
ClassContainer
AbstractType
Instance
TypeClass
Container
isMultipleChoice
name
options
classNamed: aName
likeTypeInstanceWith:
anOptionCollectionChoosen
likeTypeInstance
NoTypeClass
Container
default
Type
Instance
choosenOptions
typeClass
for: aTypeClass
isCurrentOption: anOption
NoType
Instance
default
Option
Type
option
typeClassParent
for: aTypeClass named: aString
similarTo: anotherOption
Choosen
OptionType
optionParent
typeInstanceParent
for: aTypeInstance
option: anOption


DIMBOShopper Manager
shoppedQualifyingInstances
shoppedSegmentableInstances
qualifyableShopperClasses
segmentableShopperClasses
Segmentable Data
shoppedData
match: aRealData with:
aSegmentationChoose
usingCriteria: aComparator
value: aClient
usingCriteria: aComparator
Shopper
qualifyingClasses
segmentableClasses
source
segmentableCollection
qualifyableCollection
creation
Simple Segmentable Data
chosenValue
value: aClient
usingCriteria:
aComparator
Typed Segmentable Data
chosenTypeInstance
value: aClient
usingCriteria:
aComparator
Type Instance
.....
.....
Client Shopper
Concept Shopper
Sale Shopper
Supply Shopper
Shopped Data
sourceBlockString
name
sourceClass
value: anObject
sourceBlock
Generated Shopped Date
Typed Shopped Data
typeClass
Qualified Shopped Data
Simple Shopped Data
value: anObject
TypeClass Container


VersionableObject
versions
defaultVersion
newFor: aClient
nextVersionNumber
removeVersion: aVersionObject
addToVersions: aVersion
mutationIsSoonToOccurWith:aVersionObj
nextVersionsFrom: aVersionObject
removeFromVersions: aVersion
StateVersionObject
new
EditionStateVersionObject
nextState
isEdition
isPossibleMakeVersion
isPossibleRemoveVersion
ReleaseStateVersionObject
nextState
isPossibleMakeVersion
isPossibleRemoveVersion
isRelease
VersionObject
versionState
versionManager
versionNumber
versionedObject
instanceDefaultVersionForManager : aVersionableObject
instancesWithName: aName in: aSession
doesNotUnderstand: aMessage
isDefaultVersion
removeVersion
makeVersion
closeVersion
forceMakeVersion
forceCloseVersion
openVersionbeDefaultVersion
mutate
promote
postCopy
nextVersions
Concept
.....
.....
ComplementWrapper VersionObject
complements
versionObject
newForObject: anObject
doesNotUnderstand: aMessage
makeVersion
closeVersion
forceMakeVersion
forceCloseVersion
openVersion
OriginalVersionObject
isOriginal
original
DerivedVersionObject
ancestor
isOriginal
original
Relation ComplementWrapper
owner
complement
instancesWithComplement: aComplement
beComplementTooOf: aNewComplementWrapper

An approach and an experience working with use cases and business objects

Lucio Dinoto
LIFIA - Universidad Nacional de La Plata - Argentina luciod@sol.info.unlp.edu.ar
JPMorgan - Argentina dinoto_lucio@jpmorgan.com

Motivation

This position paper has been written as part of an undergraduate final work whose objective was to develop a generic application in Visual Works in order to support the whole activities from direct marketing. The analysis pattern has been described using Fowler structures [Fowler96]. Use cases [Jacobson92] were used to capture the whole functionality of the domain. Thereby, management people can be able to fix the domain characters helping and improving the system people job. Naturally, system people can take part in this process, too. The system people develop an object model from the context model. And this one is specified according to definitions from managers and system people interaction. Thus, the gap between both areas tend to be shorter.

In the analysis and design phase around my undergraduate final thesis, I worked as the analyst and as the designer of the whole application together two thesis's advisers. One of them, a Ph.D. in marketing, drove me in the functional requirements of the application. Since the context was unlimited we needed to speak in the same terms and on the same concepts to be able to fix the domain needs and the right functionality of the application.
In order to do that, we began using Jacobson's approach. Actors and use cases models were used to fix the limits of the application. Approximately forty use cases appeared in this analysis. This phase requires **the bigger effort** in the interaction process between managers and system people. In our experience, generally between one and two page of text were enough to describe each of the found use cases.
After that, we worked on each use case developing an analysis model [Jacobson92] in order to support **each** use case functionality, but only thinking in the **real business** without pay attention yet in the system information model. In fact, we were developing a business analysis model. In this interaction with marketing people, some rules -the first ones- began to appear.
The main problem here was that lots of diagrams (analysis models with entity, control and interface objects and sometimes interaction diagrams) appeared doing the documentation very confused and uncontrollable. Therefore, the interaction with the marketing adviser began to be unproductive because I lost a lot of time explaining what each Jacobson's concept means in each phase of the analysis and we have lots of different diagrams living together us !.
We got to be agree with the functionality that the application must provide and I got a clear view of the domain, but we reached a new problem : we are not able to detect the best business solution to each problem. Nor to improve each found solution.

In view of this fact I began to write the general problems and our first solutions in a pattern-like way. Bearing this in mind, we could think about improving the business process for each problem and solution presented. Use cases descriptions gave us which were the main problems the company and the application should solve. We did a hard analysis from the use cases and we were able to detect which were the things that we must stand out from the analysis giving the "best" solution for the situation. Thus, we worked around business patterns for our analysis model. Business patterns was the platform for next object models (information system). In this implementation process I

drove to myself with the use cases approach, but designing the objects with the defined business patterns in mind!. Only when these business patterns were filtered I began to think in the final object design models.
A consequence coming from this approach is that a clear, simplified and standard documentation of the domain and the common solutions can be reached from the identified business patterns.
Finally, the main advantage of this process is to get a deep context analysis around direct marketing. This area is covered by analysis patterns where the rules of direct marketing appear.

Now, what does mean each "pattern" ? : The found "patterns" from the use cases and the object models are the rules of the business. A "pattern" can appear from a use cases or several of them. In our point of view and on the examples I gave in the paper, we found the rules. Nobody tell us about them at the beginning of developing because we work in a new and unknown context and we need to learn how is the real business. I used the word "patterns" although perhaps, the right name for them is "rules". The notation style is like patterns, but they need a lot of iterations in order to be real patterns because I have only been working with them around marketing and direct marketing.
We think business rules are very useful to understand new domains where we must begin to work. In my view, business rules are not simple constrains. They are very good and recommended solutions for the business. When we work in a mixed environment with business people or system people, use cases and their object models help us to find, to discover and to define business rules that can be, for example, common and simple rules for marketing people but not for system people. Marketing people say that an ordinary mistake of sales people is the one that they try to sell services or product differently. Marketing people affirm salesman must sell concepts, as a wrapper of the service or of the product they want to deliver on the market. This is a rule for our business! (as the concept pattern). Use cases are a very successful tool to define the limits, the characters and the fluxes of the business. Business models appear to cover the sequential interaction of the process the use cases involve. With this in hands we can think about "good solutions" or "business rules" or "patterns-like" to improve the business. I am sure that with a simple and easy engineering work similar to the guideline I describe with my paper, new people involved in the business could be able to understand better the context of each solution without losing too much time.

Approach

The following phases were used in the developing life-cycle :

Actors and Use Cases model

Activity : Actors and use cases must be identified from the domain (Jacobson like).
Participants : Management and system people.
Guidelines : A best approach here is to get few and large use cases instead lots of short use cases. Short use cases make a functional subdivision of the activity and it is not useful when analysis process wants to capture the whole workflow from each use case in order to get the right objects for his models. Every Jacobson concepts about use cases should be used here : extension, abstractions and alternative and normal curses. Be careful with the sentences and the vocabulary used here because use case descriptions are the base for future activities and these type of mistakes

have big consequences. Also, using an important amount of time here is an investment instead a waste. At this point, managers and system people interaction gives us some guidelines and constrains (see example above) useful for next business rules identification process.

Business Analysis Models

Activity : Here each use case must be modeled. Entity, control and interface objects appear. **But we must think in the business, not in the information system.** It means that at this phase, we must think about the real functionality of the business without pay attention about the system information solutions for us.

Participants : Management and system people.

Guidelines : For very complex use cases is necessary to make interaction diagrams in order to be clear with the found solution. But in general analysis models tend to be very abstract and people without a general idea about the solutions and the notations (like the managers) are very confused about what the information models want to tell them. And this happens because Jacobson's approach for analysis models uses arrows to mean collaborations and knowledge from the object but does not pay attention (at this point) about which and when these facts must happen. In next models (design model) interaction diagrams appear to cover this problem, but sometimes a clearer and just-in-time view of the model and its solution makes simpler the understanding by non-developer people (like managers).

In view of this fact, some extensions have been used in these object diagrams :

- Entity, interface and control object are used as in the original approach
- Arrows for knowledge and "part_of" relations are used as the original approach, but declaring on them an arity in the first case and an arity with a label in the second one.
- Each simple arrow in the original approach is replaced by an arrow with a label pointing out a collaboration and an index. This index, for example i, tell us the collaboration timing in the use case execution. In case that a collaboration needs others objects to ask for them for others collaborations to get its objective, a sub-index is used, for example, i.1 for the first collaboration, i.2 for the second one and so on. This extension has been created because we used to be confused looking at object diagrams without time specification and we needed looking for use case text description in order to be able to understand what the object model did.

Information System Analysis Models

Activity : **Some** business analysis models must be refined and adapted to information system. For them, a new analysis model is useful to define hide aspects from the business phase that developers must consider for the implementation. However, some business models do not need a refined information system model because the functionality to be implemented is evident from the first one.

Participants : System people.

Guidelines : Similar to Business Analysis Models guidelines.

Business Patterns

Activity : Thinking around defined use cases several business rules appear in the interaction process between system and management people. Some of them will be the patterns of our work (the rules that we think they are reusable for the context where we work are going to be documented in a pattern-like way). People involved in this phase must be able to improve and to document each of patterns found here. Despite this, system and management people must identify patterns to be standard in the business from the whole use cases. The main condition to declare something like a business pattern is that the problem should be recurrent in the company. In other words, managers and/or system people must agree that a discovered solution for a kind of problem

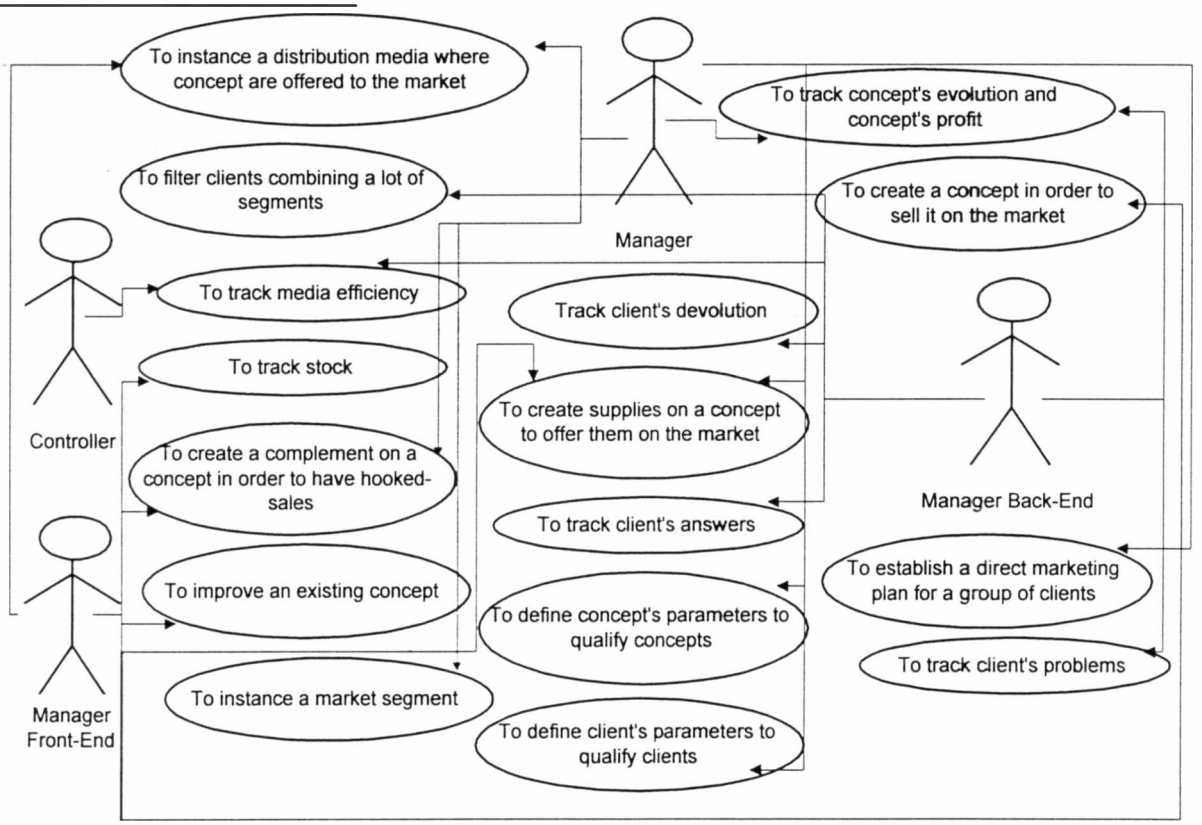
should be used each time the situation occurs in the company (or in the information system of the company). When it occurs they are defining a rule and a solution for this rule in the business context. Generally, *problem* and *forces* description provide us the description about the constrains we got in the Actor and Use Cases Models for the rule.
Because some patterns may be recurrent from others applications, a high reuse level should be considered here. **Traceability** between phases described above plays a very important role for designers because they are going to come back several times to analysis models to adjust some aspect considered before this deep analysis.

Participants : Management and system people.

Guidelines : Each pattern must be a standard in the business and each pattern must be available to improve it in the future. They should be useful to review the domain in the future by people in the same project or for new developers or new managers involving in the same project or in new project with similar domain.

An example took out from a direct marketing domain

Reduced Use Case Model

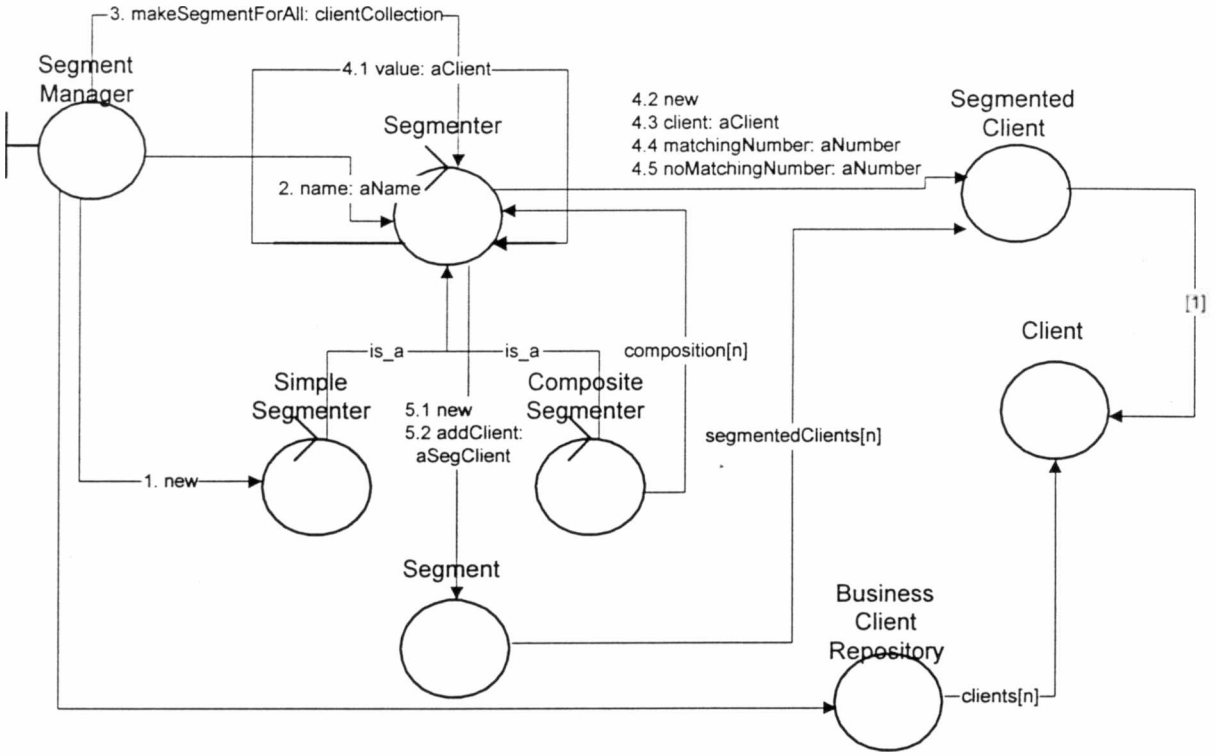


Business Constrains for the use case "To instance a segment market" (they are the ones we get in the interaction with management people at the first time. Next rules are based in this constrains):

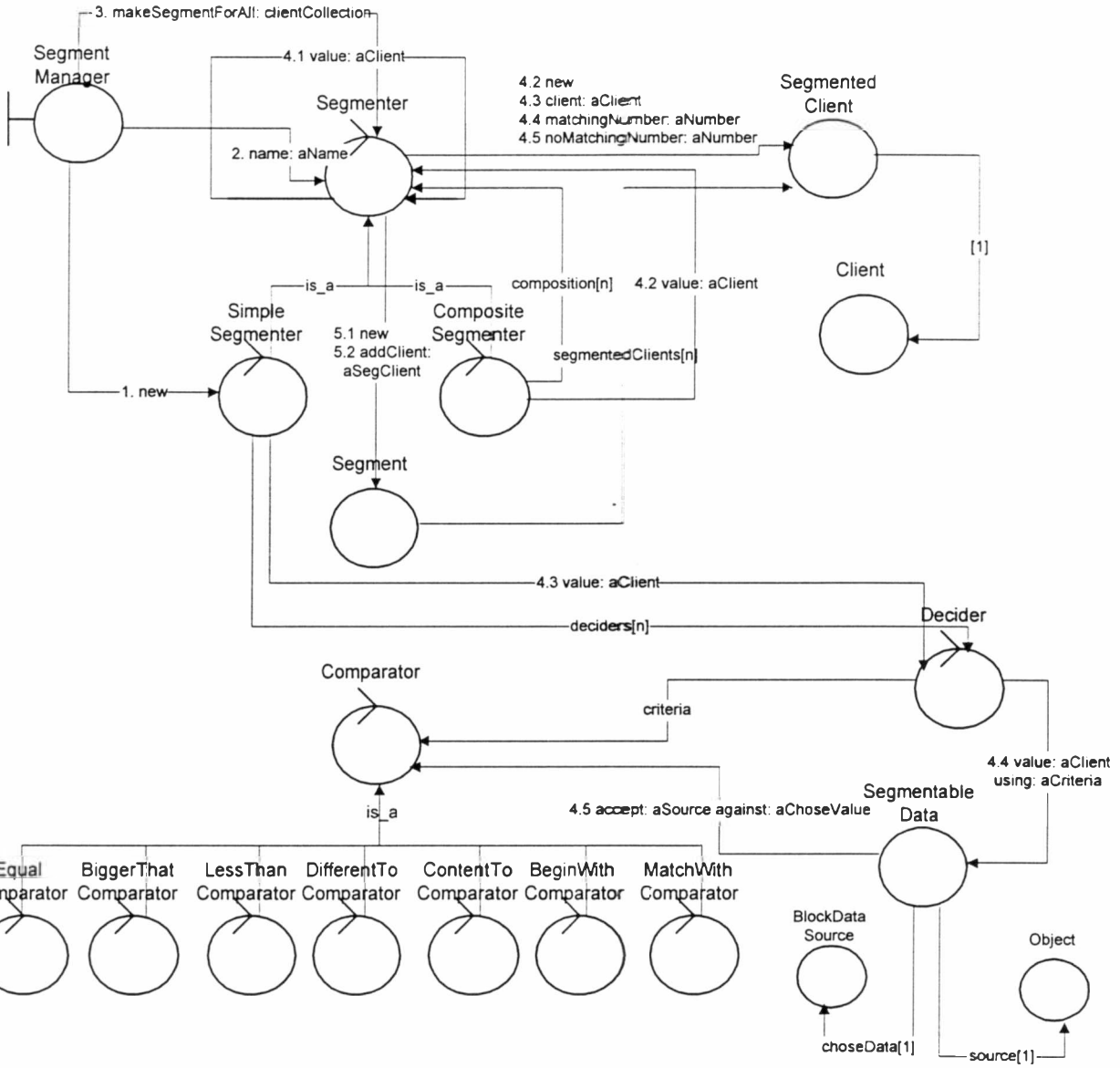
- Usually the company combines known and successful segments in order to get special group of clients where new businesses have a better rate of success.
- The company values a client in a particular segment according to the number of successes he gets in the segmentation process where chosen selections and real values are compared.

- The company must keep track of success segments and saved them in order to combine them in the future. Thus, they get new segment with a high-probability of successes.

BUSINESS MODEL - Use Case *To Instance a Market Segment*



ANALYSIS MODEL - Use Case *To Instance a Market Segment*



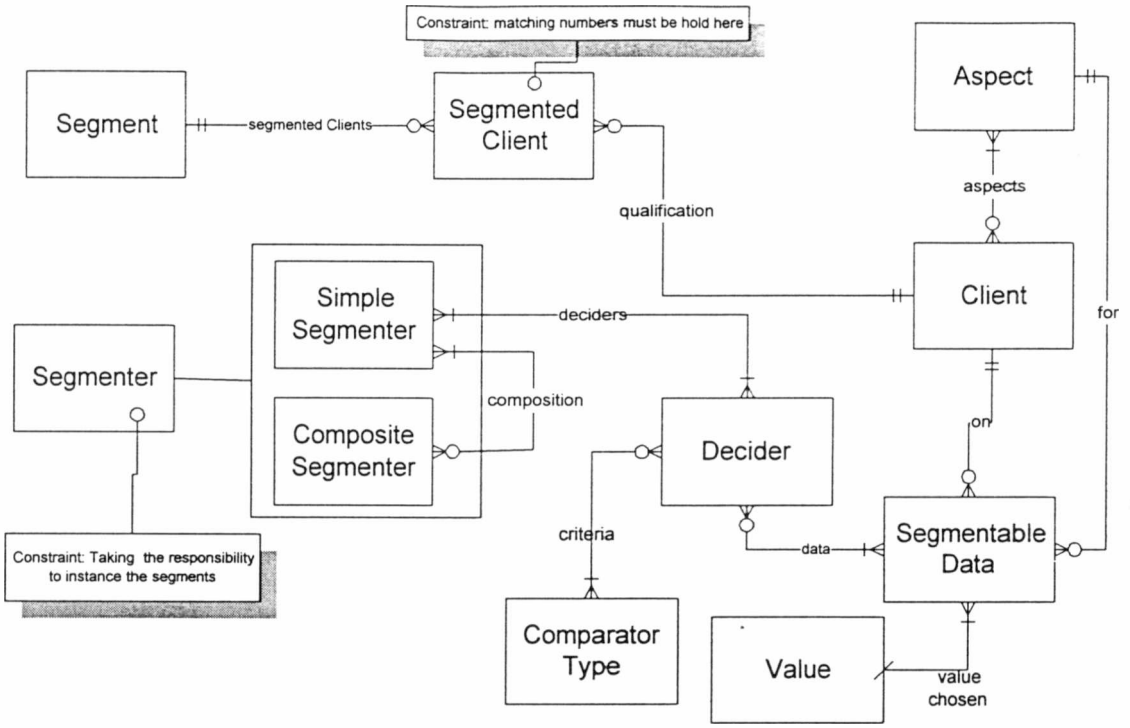
Segment pattern

Context : The management actions need to treat client grouped dynamically by some aspects.

Problem : How be able to group client by known aspects ? How do you not lose accuracy ?

Forces : Every known client in the company needs to be filtered and grouped by aspect in relation with the information associated to him in order to take tradeable actions with them. Such group tends to have common characters from the clients and they are useful for management decisions such as driving a supply to this group of people. Several segments can be joined by generating another segment. Similarly, segments can be intersected to each other. Thus, a segment or a set of defined segments in the company are useful to do client intersections.

Solution : To create and to maintain client segments in such a way as :



In our example other patterns appeared from the use cases. The *concept pattern* has been found from the use case "To create a concept in order to sell it on the market". In general terms it says something like : "*...the companies must not sell simply things : they must sell concepts... A concept is a wrapper on a product or a service...*".
Also we can obtain another "pattern" from the use cases "To improve an existing concept" and "To create a complement on a concept in order to have hooked-sales". It is the one named *evolution pattern* and it says "*...To know what, when, to whom and how a concept has been sold in the past is very useful in order to take the right action in future business decisions. So, the solution will be to keep the evolution line of the concepts to help to the company in order to know a lots about itself....On this basis, the company can be able to detect and to assign which concept can be sold together with others. It means some concepts can be self-hooked to others. For example a "Barby Doll" is a concept. Another concept may be the clothes to the doll because it is a product sold separately of the original doll. However, when a client buys a doll, he could be tempted to buy the clothes for the doll too !. In that case the doll's clothes are considered as a complement to the original concept...*"
These pattern are considered for us the rules from our business.

Conclusions

The main advantage of this approach is to have everybody speaking the same language without taking them out from their context. The result of some approaches like Jacobson's business objects [Jacobson95] is that people try to solve the whole problems in object language. Lots of problems common to system people are not interesting for management people. Thus, people working with the same structures (and notations) but thinking in a different way, may result in a big confusion. I mentioned that one of my tutors was a Ph.D. in marketing. He did not know anything about object or design patterns. When I began to study the domain with him I only worked with use cases and

analysis models from [Jacobson92]. It was very difficult for him to understand the rules from these diagrams. And I felt he was not giving to me the tips to discover the rules. As a result of this, lots of divergencies appeared continuously. I did not know anything about context rules. In other words I did not know anything about marketing rules. The marketing adviser had to give me these rules!. And patterns way bring us a better manner to work in these details. The approach of this position paper try to drive us in a right, clear and arranged way to work around rules from the context.

Another problem are notations and approaches. There are several of them and with very different styles. Thus, system people can not think that managers are going to learn about each analysis and design approach in order to work together in each phase of the developing!.

The result of the whole process described here develop a business object model for marketing and direct marketing. A scheme opened to improve is available in the management people point of view and a reusable scheme in the system people point of view. Use cases, at the beginning of the whole process, are a common departure point for managers and system people. And business patterns were the meeting point where managers and system people mix their knowledge and their conclusions. The object model is a business object model because it tries to be useful for any direct marketing system.

References

[Fowler96] : M. Analysis Patterns : Reusable Object Models, Addison-Wesley.

[Martin95] : Martin and J.Odell. Object Oriented Methods : A foundation. Englewood Cliffs, NJ :Prentice-Hall, 1995

[Jacobson92] : Object Oriented Software Engineering : A Use-Case Driven Approach. Addision-Wesley, New York 1992

Referencias

Bibliografía Utilizada

[1.] Marketing Directo Integrado - Josep Alet
[2.] Object Oriented Software Engineering - A Use-Case Driven Approach. Ivar Jacobson. Addision-Wesley 1992.
[3.] The Object Advantages: Bussines Process Re-engineering with Object Technology- Ivar Jacobson, Maria Ericsson, Agneta Jacobson.
[4.] Design Patterns - Elements of Reusable Object-Oriented Software - Gamma, Helm, Johnson, Vlissides.
[5.] Smalltalk Best Practice Patterns - Kent Beck
[6.] The Smalltalk Developer's Guide to Visual Works - Tim Howard
[7.] Smalltalk-ParcPlace Visual Works Release 2.0 and 2.5.1 Guide, Tutorial, Database Tutorial and Cookbook (http://www.parcplace.com)
[8.] Smalltalk with Style - Klimas, Skublics, Thomas.
[9.] Paper: Understanding and Using ValueModels - Bobby Woolf (bwoolf@ksccary.com)
[10.] Paper> Suggestions for a successful user interface - Amy Gause
[11.] Domain Specific Patterns: Conversions, Persons and Roles, and Document and Roles. Ari Schoenfeld (ari@servidor.unam.mx) - PLoP 96
[12.] Pattern Integration - Variations of State - Odrowski and Sogaard (odrowsk@ibm.net). PLoP 96
[13.] The Type Object Pattern - Bobby Woolf (bwoolf@ksccary.com). PLoP 96
[14.] Patterns Languages of Program Design - Adilson Wesley
[15.] Paper: Use Cases> the Pros and Cons - Donald Firesmith (dfiresmith@ksccary.com)
[16.] Designing Object Oriented Software - Rebecca Wirfs-Brock. 1990
[17.] Crossing Chasms: A Pattern Language for Object-RDBMS Integration - Kyle Brown (kbrown@ksccary.com) and Bruce Whiteack
[18.] Analysis Pattern. Reusable Object Model. - Martin Powler - Adilson-Wesley 1997
[19.] Libro de BDOO - Silvia o Casco
[20.] The Null Object Pattern - Boody Woolf (bwoolf@ksccary.com). PLoP 96
[21.] Smalltalk for Bussines - Object Magazine Septiembre 1996
[22.] Testing Object-Oriented Components - John D. McGregor - Clemson University - OOPSLA97
[23.] An approach and an experience working with use cases and business objects - Lucio Dinoto - Workshop #32 OOPSLA 97 - Atlanta - USA