REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 10, número 2
Págs. 7 - 26

Modelo de Datos Reorientando a Objetos (MODRO): Regresando al Paradigma Orientado a Sistemas

Jaime Octavio Albarracín Ferreira* Fernando Antonio Rojas Morales*
Fecha de Recibido: 29/01/2009 Fecha de Aprobación: 15/05/2009

Resumen

El modelado semántico relacional y el modelado de objetos de una empresa produce modelos fragmentados y amorfos, síntoma de organizaciones con funcionalidad estructuralmente fragmentada. De hecho, la reingeniería afirma que la funcionalidad de la organización tradicional está fragmentada, y que su estructura no corresponde a la lógica de sus procesos, sino a jerarquías de mando y de poder. A su vez, la teoría de objetos diagnostica que las funcionalidades están sueltas si no están contenidas dentro de su propia estructura. La reingeniería señala cómo desfragmentar la funcionalidad organizacional, mientras que la teoría de objetos indica cómo encapsularla. Pero la teoría de objetos obstaculiza la desfragmentación de la funcionalidad organizacional debido a su exigua granularidad. Por fortuna, la teoría de sistemas puede corregir esto, al concebir conjuntos de estructuras de funcionalidad global, en vez de estructuras individuales con funcionalidades individuales auto contenidas, como lo hace hoy la teoría de objetos. Acercar los objetos a los sistemas induce un nuevo concepto de modelo de datos, holístico, denominado "MOdelo de Datos Reorientado a Objetos", MODRO (ORDAM en inglés), para llevar más allá al paradigma de objetos. El MODRO soluciona lo que los OODBMS de hoy no: encapsular dentro de una estructura organizacional su respectiva funcionalidad transaccional, tanto de las bases de datos como de la organización; igual que el hardware (lo estructural) contiene software (lo funcional) (e.g. teléfono celular - sim card), para que puedan tener utilidad, tal como sucede con todos los demás objetos auto contenidos del mundo real.

Palabras clave: *Teoría de Sistemas, Teoría de Objetos, Reingeniería, Modelos de datos, Bases de datos, Componentes, Aspectos.*

Abstract

The Modeling of an organization using the relational, the semantic, or even objects models results in fragmented and amorphous models. It reveals a fragmented organizational functionality and an amorphous organizational structure. Reengineering states that the traditional organization's functionality is fragmented. The current objects paradigm cannot solve the fragmentation of functionality because of its small granularity; that is why a systems paradigm, based on systems theory, is proposed. A new holistic data model called “Objects Reoriented Data Model” – ORDAM is suggested. ORDAM does reflect a solid organizational structure, which is not amorphous. It also encapsulates a compacted organizational functionality integrating both its functionality and structure. Likewise, it offers a greater granularity for the reduction of concerns resistant to being encapsulated (in Aspects Oriented Programming: AOP), and to increase the level of abstraction of reusable components (in Component Based Software Engineering: CBSE).

Keywords: Theory of Systems, Theory of Objects, Reengineering, Data Models, Databases, Components, Aspects.

---

* Universidad Industrial de Santander, Ciudad Universitaria UIS, Bucaramanga, Colombia, {jaimealb,frojas}@uis.edu.co

1 Introducción

Algunos estudios de la reingeniería han determinado que las organizaciones se encuentran jerárquicamente parceladas [1, 14, 16]. Esta parcelación deriva en la poca articulación entre la disposición organizativa y la lógica de los procesos; a tal fragmentación estructural se la denomina "*primera fragmentación*". También se ha observado que la funcionalidad se encuentra fragmentada, pues sus procesos y transacciones aparecen conformados por actividades innecesariamente distribuidas a través de diferentes áreas (o 'parcelas'), quedando así fragmentadas en una "*segunda fragmentación*", y dispersas además a través del tiempo, es decir fragmentadas en una "*tercera fragmentación*", constituyendo todo esto el modelo industrial de Adam Smith [2]. A pesar de que el modelo de Smith esté siendo parcialmente superado, queda todavía una fragmentación escondida en el modo de operación de los procesos organizacionales especialmente en el proceso contable, que al nivel global es típicamente ejecutado por etapas, y que de esa forma se ha reflejado en los sistemas de información computarizados y en las bases de datos [3, 4, 5]. Finalmente, en caso de haber empresas que hayan superado este paradigma, no se conocen publicaciones formales que lo acrediten.

La desintegración tiene el poder de hacer *incompetente* a cualquier organización, aunque ésta tuviese el hardware más poderoso y el software más moderno y, aunque tuviese bases de datos con todas las reglas de integridad conocidas aplicadas rigurosamente [1, 16]; mantener la integridad de las bases de datos es uno de los principales retos de los diseñadores [6, 7, 8, 9, 10].

De esta forma, la desintegración de la organización obedece en parte a la desintegración de sus datos, o sea a la desintegración de sus bases de datos. De modo que, si se pudiera obtener un modelo para desarrollar bases de datos, que aunque fuesen corporativas no dejasen de estar integradas, se estaría logrando una solución para la desintegración de la organización. Pero tal solución sería parcial, toda vez que solamente atendería el aspecto estructural de la organización.

Para obtener una solución más completa, es oportuna una filosofía de objetos, que considere también el aspecto funcional de la organización. Porque una organización al ser vista como un objeto [11], deberá tener tanto una estructura como una funcionalidad no disociada sino encapsulada en lo estructural. De esta manera, la organización estaría en un estado de "*cuarta fragmentación*": de un lado lo estructural (la descripción de sus bases de datos), y del otro lo funcional (sus programas de computación). De aquí la importancia de una filosofía integradora como la filosofía de objetos.

Se plantea entonces un problema de reingeniería para lo *funcional* [1, 6, 16] y de modelos de datos para lo *estructural* [6, 8, 10, 12, 15]. Puesto como un problema de objetos, que comienza por entender la organización como un objeto [8, 11], con su *estructura de áreas* modelada por rutas referenciales[1] pertinentes a cada una de sus áreas, y su respectiva *funcionalidad transaccional* encapsulada en forma de métodos en el modelo de datos, identificado como *Modelo de Datos Reorientado a Objetos,* MODRO. Reorientado porque las simples entidades individuales de inapropiado grano fino, error señalado por C. J. Date como "el gran error" [15], ya no son directamente las clases [12, 17, 24], sino que son concebidas como conjuntos y subconjuntos de entidades interrelacionadas, cada una con su respectiva transacción como método común, dentro de un enfoque holístico de mayor granularidad. De este modo se minimiza la cantidad de incumbencias transversales (*crosscutting concerns*) que oponen resistencia a ser encapsuladas [28, 29, 30] dañando la legibilidad del código, y, también se incrementa el nivel de abstracción potenciando la reutilización del código, uno de los principales objetivos de la Ingeniería del Software Basada en Componentes (CBSE) y uno de los desafíos más difíciles de aplicar en la Programación Orientada a Objetos [31, 32, 33].

El planteamiento anterior se aborda de la siguiente forma, en la sección 2, se hace una descripción del modelo y los elementos que lo componen: conjuntos referenciales, tipos, clases, componentes, aspectos e instancias, haciendo referencia a un proceso contable como caso de estudio. En la sección 3 se presentan algunos trabajos relacionados pertinentes. En la sección 4 se abordan perspectivas del modelo. Finalmente, en la sección 5 se dan algunas conclusiones sobre el MODRO.

2. Descripción del Modelo Aplicado al Proceso Contable

En la construcción de un modelo E/R se encuentran normalmente entidades fuertes y entidades débiles [10, 7]. Si el modelo E/R describe la estructura de información de toda una organización, semejante a como un plan contable refleja la información financiera, se reconocerá en el *libro mayor* que toda organización posee, como la entidad más fuerte, puesto que a esta entidad llegan todas las rutas referenciales del modelo, figura 1. En la vida real esto significa que en el *libro mayor* se resumen todas las transacciones ocurridas durante algunos días de un período en el cual tal libro persiste aunque la organización no ejecutase transacciones el resto de días del mismo período [4, 34, 35]. Evidencia de la fortaleza de la persistencia es su independencia de la ocurrencia de transacciones.

1 En un modelo E/R una ruta referencial es un camino formado por entidades entrelazadas [10].

Así mismo, en el *libro diario* que también posee toda organización, se podrá reconocer la entidad más débil, puesto que de esta entidad parten todas las rutas referenciales, figura 1. En la vida real esto significa que el *libro diario* no mantiene nada consolidado sino que sólo registra detalles transaccionales durante breves plazos, generalmente de un día. Así, en los días en que la organización no ejecuta transacciones, el libro *Diario* no contendrá registros. Esta dependencia de la ocurrencia de transacciones evidencia su debilidad. En otras palabras, todas las entidades de un modelo E/R corporativo se polarizan con la entidad más fuerte y en la más débil. Las demás entidades intermedias se localizan alineadas en cada ruta referencial, entre estos polos. Dichas entidades intermedias son como *Libros Auxiliares*, pero con más atributos que los que podría contener un simple formato de columnas saldo, débito y crédito, y, por ser intermedias, poseen relaciones que van de polo a polo, pues son atravesadas por las rutas referenciales, como se puede apreciar en la figura 1. Esta figura no muestra el diagrama E/R sino el diagrama referencial [10] en el que los rombos de las interrelaciones entre entidades ya están resueltos; la forma de globo del modelo es arbitraria.

Este modelo, polarizado y convergente por efectos de la teoría contable [4, 5, 35], almacena muchos detalles en el Diario y mucha generalidad en el Mayor, por lo que las interrelaciones entre sus entidades van de "muchos" a "uno", de Diario a Mayor. Pueden existir eventuales interrelaciones entre entidades no alineadas en la misma ruta referencial que generan relaciones ecuatoriales, representadas en la figura 1 como "x" y "y". En general, podría presentarse una interrelación entre cualquier par de entidades intermedias, pertenecientes a rutas referenciales distintas no siempre contiguas, por lo que favorece más concebir el modelo como un volumen esférico que como una superficie. Sin embargo, cuando una transacción evoluciona a lo largo de algún meridiano, las flechas ecuatoriales no la desvían sino que va y vuelve, ajustándose siempre al meridiano, pues toda transacción inicia en el Diario, pasa por los Auxiliares y termina siempre en el Mayor [4, 5, 35].

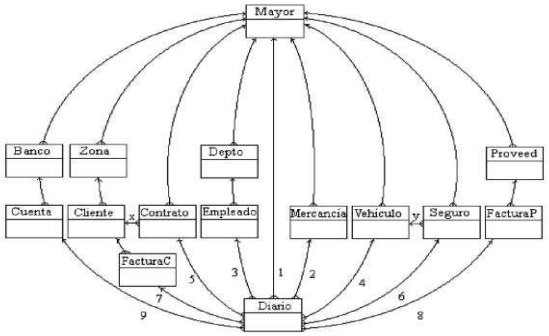


Fig.1: El MODRO (Súper tipo).

Expuesto lo anterior se reconocen dos hechos, por un lado la filosofía de objetos indica que la anterior estructura debe estar asociada a una funcionalidad, y por otro, que la más importante funcionalidad que ocupa a toda organización, es de orden transaccional; una organización en la que no ocurriesen transacciones colapsaría. Por eso, se puede decir de manera simplificada, que una contabilidad consiste el registro técnico de las transacciones de una empresa.

2.1 Descripción de Conjuntos Referenciales

Estructuralmente una organización está compuesta de diversas áreas o departamentos[2] como Compras, Ventas, Producción o Tesorería. Cada área ejecuta sus propias transacciones y cada transacción sólo visita y actualiza las entidades (generalmente como archivos) que le corresponden en el área que le corresponde. Por ejemplo, Compras sólo realiza transacciones de compras, y una transacción o compra específica nunca visitará la entidad Clientes ni la entidad Empleados sino posiblemente las entidades FacturasP (facturas de proveedor) y Proveedores, además de Diario y Mayor. Es decir, a cada área le pertenecen sólo ciertas entidades, aunque una entidad puede ser visitada por diversas transacciones de diferentes áreas, por lo que algunas entidades estarán en la intersección de diferentes conjuntos referenciales, como es el caso de *Diario* y *Mayor*. El conjunto de entidades visitadas por las transacciones de un área, están enlazadas en una o más rutas referenciales: a este conjunto de rutas referenciales se le denomina aquí "*conjunto referencial*" (de grano más grueso), el cual representa a cada área, como se muestra en la figura 2.

Por ejemplo, el conjunto de rutas {2, 8, 9}del *tipo compras* sería el conjunto referencial del departamento de Compras. Note que cada una de las rutas inicia en el *Diario* y termina en el *Mayor*, convergencia que refleja el sello de la contabilidad organizacional. Así mismo, el conjunto de rutas {1, 2, 7, 9}del *tipo ventas* podría ser el conjunto referencial del almacén encargado de las ventas. Si tesorería fuese la encargada de los cobros y de los pagos, su conjunto referencial podría ser {7, 8, 9}aunque todo depende de las reglas del negocio, que aquí se asumen simplificadas al extremo. Las rutas referenciales son los meridianos numerados que se señalan en la figura 1

2 Estas áreas o departamentos deberían ser entendidos más como funcionalidades específicas que como parcelas de la empresa.

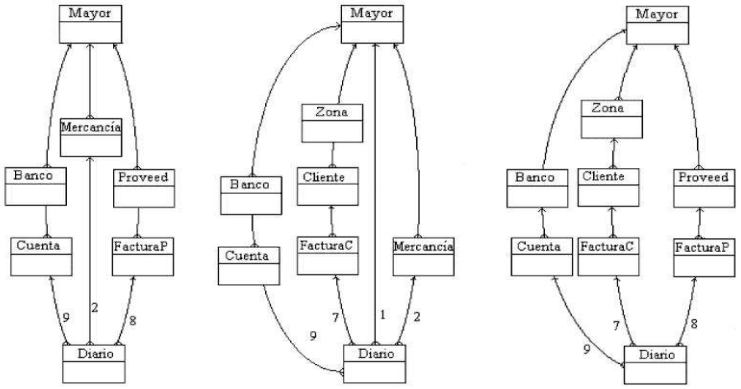


Fig 2. Rutas Referenciales del Conjunto Referencial

El MODRO propone llevar las cosas más lejos que una contabilidad convencional que sólo debita o acredita unas cuentas como Mercancías, Bancos y/o Proveedores, hoy en el *Diario*, mañana en los *Auxiliares* y al final del mes en el *Mayor*, en etapas y además por lotes. En su lugar el MODRO permitirá actualizar todas las entidades del conjunto referencial, al mismo instante en que ocurran cada una de las transacciones, una a una (no por lotes), y además registrando un mayor número de datos que los que se pueden mantener en un simple esquema de columnas saldo, débito y crédito. Por eso el MODRO no reconoce métodos para entidades individuales específicas, como sí lo hacen los modelos de datos orientados a objetos convencionales [12, 17], sino para conjuntos referenciales completos, entre otras cosas porque cada transacción siendo atómica es ejecutada por completo en el menor plazo posible.

Además el MODRO, con una mayor granularidad, contiene la totalidad de conjuntos referenciales y encapsula la totalidad de patrones de transacciones, modelando así a la organización completa y convirtiéndose de este modo en contexto para todos los posibles componentes. Estos componentes de software no fragmentarían la organización porque no habría un módulo de Inventarios, un módulo de Cartera, un módulo de Contabilidad, etc., como suele ser convencionalmente, sería más bien un software integrado según las transacciones, es decir, un software compuesto de componentes transaccionales, como se verá en la siguiente sección.

2.2 Descripción de Tipos

En el MODRO, un tipo corresponde a un conjunto referencial en el que cada una de sus entidades tiene pertinencia para la misma área de la

empresa. Por ejemplo, el conjunto de entidades *Diario*, *FacturasP*, *Proveedores*, *Mercancía*, *Cuentas*, *Bancos* y *Mayor*, es un conjunto con pertinencia al área de Compras, pero no al área de Ventas (por la impertinencia de *FacturasP* y *Proveedores*), ni al área de *Tesorería* (por la impertinencia de *Mercancía*), como se describe en la figura 2. Luego el conjunto referencial {2, 8, 9}es del "*tipo compras*" y la signatura de su método podría llamarse *comprar*, que al implementarse haría que todas las clases de compras hereden dinámicamente de la entidad *Mercancía* incrementando la cantidad en existencia.

De esta forma, por cada una de las áreas de la organización, como se describe en la figura 2, hay un conjunto referencial que representa un tipo: el *tipo compras*, el *tipo ventas*, el *tipo tesorería*, el *tipo producción,...* Cada tipo con la signatura de su respectivo método: *comprar, vender, pagar y cobrar, producir,...* Cada una de estas signaturas llevaría a la implementación de un método para seleccionar y modificar atributos, y/o insertar y/o eliminar tuplas de una manera muy singular y característica a la transacción en cuestión.

Sin embargo, pasar directamente de las simples signaturas en los tipos, a su implementación, resultaría muy abstracto para la verdadera funcionalidad empresarial, que de hecho es más específica. Por ejemplo, *comprar* requiere una especificación más detallada de cómo sería la compra (i.e. a crédito o al contado), a quién se le compraría (i.e. proveedor extranjero o nacional), con o sin impuesto, con o sin fletes, y así sucesivamente. Cada modalidad específica de compra es lo que en contabilidad se identifica y registra como transacción, con asientos de valores débitos y valores créditos muy singulares. Así pues, las signaturas generarían dificultad al intentar ser implementarlas directamente sin un adecuado nivel de especificación. Es necesario por tanto, implementar primero los diferentes tipos mediante las clases y las subclases apropiadas al nivel de especificación requerido. Por ejemplo, si el negocio realiza compras al contado y compras a plazos, la implementación del tipo compras requerirá de una clase y dos subclases, como se muestra en la figura 3. El método de la clase sería *comprar* y se implementaría sumando la cantidad comprada a la existencia de *Mercancía* (ver arriba en los asientos de contabilidad, el asiento "Ruta 2", común a los dos casos). El método de las subclases sería en parte heredado (por polimorfismo) y en parte propio (restar al saldo de una cuenta bancaria y al saldo total del banco, en un caso, asiento "Ruta 9", o insertar factura nueva y aumentar la deuda con el proveedor, en el otro, asiento "Ruta 8").

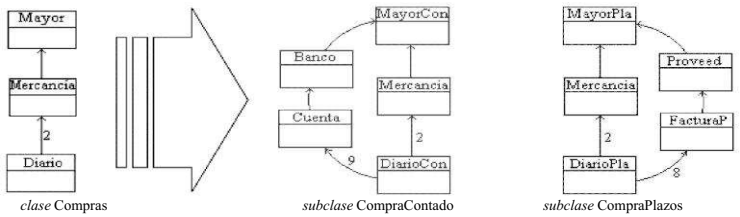


Fig. 3 Implementación del tipo compra

En otras palabras, una clase es la implementación de un tipo [17]. Para este artículo, un tipo es implementado mediante una clase y un número de subclases igual al número de transacciones específicas que el área en cuestión lleve a cabo. Por eso el tipo compras se implementa mediante una clase y dos subclases: La clase compras y las subclases compras al contado y CompraPlazos, como se ve en la figura 3, que son las transacciones específicas del tipo compras que la empresa en cuestión realiza. Estas implementaciones son de grano más grueso que el de una entidad individual, pero son también lo suficientemente específicas para describir la verdadera funcionalidad empresarial, transaccional, pudiendo reconocerse en ellas, además, los componentes de software que compondrían el MODRO.

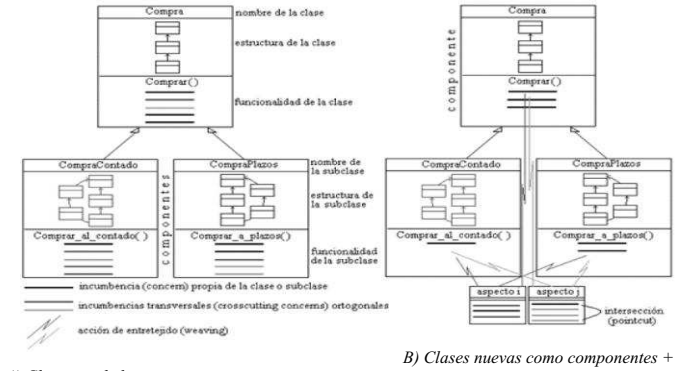
2.3 Descripción de Clases

Convencionalmente una clase se define como un grupo de objetos que comparten el mismo conjunto de atributos, interrelaciones y métodos [17, 18, 19, 20, 21, 22]. Sin embargo, esta definición se extiende en este artículo pues el concepto de clase supera a la entidad individual de grano fino, implícita en la anterior definición. En efecto, aquí una clase es planteada, no como una entidad, sino como un conjunto de entidades enlazadas e inter actuantes de acuerdo a un mismo método, el cual no es particular a cada entidad sino común a todas ellas.

La diferencia estriba en la idea de que un objeto es una entidad auto contenida del mundo real [17, 18, 19, 20, 21, 22], mientras que este artículo sostiene que son conjuntos de entidades, y no entidades específicas, los que deben ser auto contenidos. Por ejemplo, la entidad *Automóvil* podrá ser en la sala de ventas una entidad auto contenida que "arranca", "frena", "acelera" y "gira", pero no lo es autosuficiente en el tráfico urbano, donde está inter relacionado según reglas con otras entidades como *Personas*, *Calles*, *Señales* y *Semáforos*. Las entidades individuales son de grano muy fino para el contexto de la realidad en donde están holísticamente [27], y sistémicamente interrelacionadas, no sólo estructuralmente sino también funcionalmente, mediante un método común. ¿Qué podría suceder con un automóvil en una ciudad, que "arranca", "frena", "acelera" y "gira", que ignore señales, transeúntes y

demás? Lo auto contenido es el conjunto completo, el sistema, y no solamente sus elementos. En este artículo entonces, se definen operaciones y métodos para el conjunto de entidades completo (aplicando Reingeniería), en vez de hacerlo para cada una en particular.

Además, siguiendo el caso del proceso contable planteado, se juzga 0elemental la pertenencia de la operación "hacer descuento" a la entidad *Factura*, o la operación "restar cantidad" a la entidad *Artículo*, pues estas operaciones son sólo parte de las operaciones de una transacción que es más compleja y más real, como podría ser la transacción "vender al contado" o "vender a crédito" respectivamente.



A) Clases y subclases con estructura propuesta

B) Clases nuevas como componentes + separación de aspectos

Fig. 4: Diagrama de Clases

De esta manera, el diagrama de clases en UML correspondiente a la figura 3 se extendería un poco, ya que el espacio convencional destinado para los atributos [23, 24], en este artículo es redestinado para inscribir la estructura de la clase, o sea el conjunto completo de entidades que conforman las nuevas clases o subclases, como se observa en la figura 4, aunque dichas entidades contienen atributos.

```
class Compra
{
 {
   entity Mayor
   (extent cuentas_mayores key
codigo_cta_mayor)
   {
   // Define los atributos
      ….
   // Define su interrelación con la
entidad Mercancía
      …
   };
   entity Mercancía
   (extent articulos key
codigo_articulo)
   {
   // Define los atributos
      …
   // Define interrelaciones con
entidades Mayor y Diario
      …
   };
   entity Diario
   (extent bitácora key
codigo_transaccion)
```

```
    {
    // Define los atributos
        …
    // Define su interrelación con la entidad Mercancía
        …
    }
 };
// Define de la clase Compras su método comprar
void insertar_entrada _mercancia _en_diario (…
// "Ruta 2"
void sumar_a_existencia_en_mercancia (…
// "Ruta 2"
void sumar_entrada_mercancia _en_mayor (…
// "Ruta 2"
};
// Fin definición de la clase Compras

Class CompraContado extend Compras
// Hereda de la clase, entidades, interrelaciones y método
// Define además entidades propias de esta subclase
{
 {
    entity MayorCon extends Mayor
    (extent cuentas_mayores_contado)
    {
    // Hereda todos los atributos de la entidad Mayor
    // Define una interrelación adicional con la entidad Banco
        …
    };
    entity Banco
    (extent bancos key nombre_banco)
    {
    // Define los atributos
        …
    // Define interrelaciones con Cuenta y MayorCon
        …
    };
    entity Cuenta
    (extent cuentas key codigo_cta_banc)
    {
    // Define los atributos
        …
    // Define interrelaciones con Banco y DiarioCon
        …
    };
    entity DiarioCon extends Diario
    (extent bitacora_contado)
    {
    // Hereda todos los atributos de la entidad Diario
    // Define una interrelación adicional con la entidad Cuenta
    }
 };
/* Hereda el método comprar y lo completa para dejar la transacción "Comprar al contado" balanceada según los estándares internacionales de Contabilidad [34] */
void insertar_retiro_de_bancos _en_diario (…
// "Ruta 9"
void restar_a_saldo_cuenta_banco (…
// "Ruta 9"
void sumar_retiro_de_bancos _en_mayor (…
// "Ruta 9"
};
// Fin definición de la subclase CompraContado
Class CompraPlazos extend Compras
// Hereda de la clase, entidades, interrelaciones y método
// Define además entidades propias de esta subclase
{
  {
    entity MayorPla extends Mayor
    (extent cuentas_mayores_plazos)
    {
    // Hereda todos los atributos de la entidad Mayor
    // Define una interrelación adicional con entidad Proveed
        …
    };
    entity Proveed
    (extent proveedores key dni_proveedor)
```

```
    {
    // Define los atributos
        …
    // Define interrelaciones con FacturaP y MayorPla
        …
    };
    entity FacturaP
    (extent facturasp key numero_facturap)
    {
    // Define los atributos
        …
    //Define interrelaciones con Proveed y DiarioPla
        …
    };
    entity DiarioPla extends Diario
    (extent bitacora_plazos)
    {
    // Hereda todos los atributos de la entidad Diario
        // Define una interrelación adicional con entidad FacturaP
            …
      }
    };
/* Hereda el método comprar y lo completa para dejar la transacción "Comprar a plazos" balanceada según los estándares internacionales de Contabilidad [34] */
void insertar_deuda_ proveedor _en_diario (…
// "Ruta 8"
void insertar_nueva_factura_proveedor (…
// "Ruta 8 "void sumar_ deuda_ proveedor_en_mayor (…
// "Ruta 8"
};
// Fin definición de la subclase CompraPlazos
                …..
```

Para ilustrar estas ideas, se ha construido en un ODL modificado el esqueleto de un prototipo que muestre la implementación del tipo c*ompras*, mediante la clase y las subclases de la figura 3, expresando las estructuras de la clase y de las subclases, constituidas por conjuntos de entidades (no por atributos), por lo cual el término *entity* aparece declarado dentro de la definición de clase *class*. Además, para imprimir mayor legibilidad al código y también en correspondencia con la figura 3, las palabras clave de la clase se resaltan en negro (***class***, ***entity***, y su método ***comprar***), las de la subclase "CompraContado" en rojo (**class**, **entity**, y su método ***comprar_al_contado***), y las de la subclase "CompraPlazos" en azul (***class***, ***entity***, *y su método* ***Compra_ a_plazos***). El código es el mostrado arriba.

El término *entity* es parte del léxico añadido a ODL para extender el concepto de clase, en la cual las operaciones son comunes para todo el conjunto de entidades que conforman cada clase o subclase y no para cada entidad en particular, según la premisa atrás señalada. Por esto es importante observar cómo las operaciones del anterior prototipo han sido codificadas como un común denominador para todo el conjunto de entidades que conforman cada clase o cada subclase. Sin embargo, aunque a cada entidad (*entity*) sólo se le codifican atributos e interrelaciones, no operaciones, nada prohíbe que estas le sean eventualmente codificadas en forma particular, a una entidad cualquiera que necesite ser auto contenida. Lo que si le faltaría a un paquete (package) UML sería un método propio y común a todo el conjunto de clases contenidas en él, para ser equiparado a una clase del

MODRO [24]. Por su parte, el estándar ODMG no hace referencia a paquetes ni a clases internas [22].

Según la teoría de conjuntos, aplicable a las clases del MODRO por ser éstas conjuntos referenciales la intersección de todas las subclases de un mismo tipo, es la clase que implementa al tipo (Figura 3), mientras que su unión es el mismo tipo (Figura 2). Así mismo, la intersección de la totalidad universal de clases (en el MODRO), es la superclase mostrada en la figura 5.

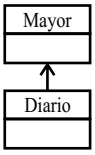


Fig. 5. Superclase

En efecto, estas dos entidades, el libro *Mayor* y el libro *Diario*, son las dos únicas entidades universalmente comunes a cualquier organización del mundo, porque son los dos libros mínimamente requeridos para el registro técnico de transacciones en cualquier empresa, es decir, para el mantenimiento mínimo de una contabilidad financiera [4, 5, 35].

2.4 Componentes.

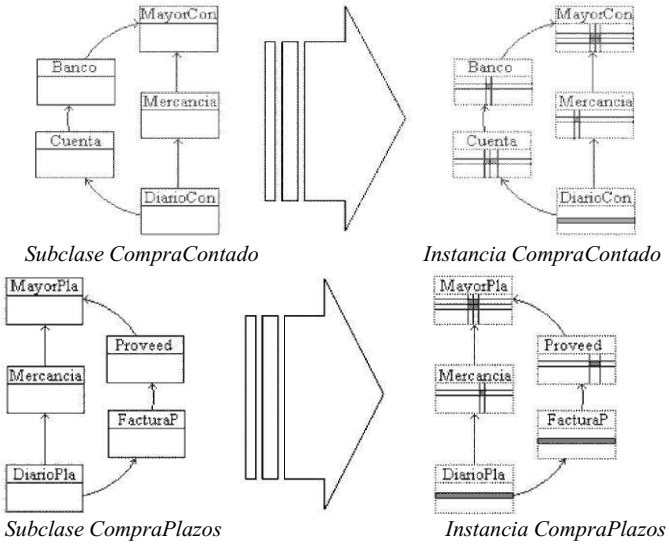
En el contexto del MODRO, este enfoque de superclases, clases y subclases, es una regresión de abstracciones desde la superclase de la figura 5 de mayor abstracción, hasta las subclases más específicas de la figura 3, aunque no al extremo de pulverizar las subclases en entidades individuales sueltas. Las entidades individuales como clases, ocultan muchas limitaciones para la reutilización de software, pues sus métodos resultan muy elementales e inútiles para su eficaz reutilización para el ámbito de una organización con transacciones múltiples y complejas. Por el contrario, la especificación de las clases propuestas tiene sentido corporativo, y no de simples tablas o relaciones. Además sus métodos describen el comportamiento de transacciones completas como "comprar al contado", "exportar al Japón", "vender a crédito", "pagar nómina", que son del tipo de transacción que ocupan a la mayoría de empresas. De modo que, si clases y subclases como las propuestas en este artículo fueran diseñadas, implementadas y compiladas, con interfaces bien definidas como su identificación dentro de un menú de pantalla, podrían constituirse en componentes listos para ser conectados y utilizados (plug and play) y reutilizados. Porque ante la estandarización de las normas internacionales de Contabilidad (NIC) [34] y ante la inclinación de la estructura y de la funcionalidad de las empresas hacia estándares de clase mundial, nacen enormes posibilidades para la utilización y reutilización de componentes de software como los bosquejados aquí.

2.5 Aspectos.

Por otro lado, asuntos como el *logging* (registro de la actividad de un programa), la seguridad, la persistencia, la presentación, la autenticación, la sincronización, la concurrencia, e inclusive la misma función específica que debe llevar a cabo un programa, son conocidos en programación orientada a aspectos (AOP en inglés) como incumbencias (*concerns*). Muchas de estas incumbencias son ortogonales y se repiten de programa en programa (líneas verdes y rojas de la figura 4.A), o de clase en clase, por lo que se denominan incumbencias transversales (*crosscutting concerns*), generando código desparramado y enredado (*scattering and tangling*). La solución que propone la comunidad de AOP consiste en la creación de clases adicionales a las tradicionales o módulos llamados aspectos (*aspects*), como se esquematiza en la figura 4.B con el "*aspecto i*" y el "*aspecto j*", lo cual se conoce como "separación de aspectos". La contribución de este artículo a la AOP estriba en la definición de métodos para clases y subclases de mayor grosor y alcance, figura 4.B, y no para entidades individuales como clases con métodos mucho más elementales, efectivamente desparramados con proliferación de incumbencias transversales y gran entrelazamiento.

2.6 Instancias.

Finalmente, una instancia es el conjunto de valores de aquellos atributos actualizados en la ocurrencia de una transacción específica, los cuales se interrelacionan entre sí de la misma forma en que lo están las entidades de la subclase de la cual proviene (véase figura 6).



Subclase CompraContado *Instancia CompraContado*

Subclase CompraPlazos *Instancia CompraPlazos*

Fig. 6. Diagramas de Subclases e Instancias.

Una instancia hereda pues, los enlaces de su subclase, los atributos que le son pertinentes, y su comportamiento, aunque una instancia no está propiamente constituida por entidades (razón por lo cual están punteadas). Así por ejemplo, en la figura 6 se señalan dos instancias, una de la subclase CompraContado (roja) y la otra de la subclase CompraPlazos (azul). La primera hereda de su propia subclase el método *comprar_al_contado*, como se muestra en el siguiente código:

void insertar_entrada_mercancia_en_diario (…
void sumar_a_existencia_en_mercancia (…
void sumar_entrada_mercancia_en_mayor (…
void insertar_retiro_de_bancos_en_diario (…
void restar_saldo_cuenta_banco (…
void sumar_retiro_de_bancos_en_mayor (…

Las tres primeras líneas son actualizaciones sobre la Base de datos que sucedieron por la ruta 2, y las tres siguientes que sucedieron por la ruta 9. La segunda instancia hereda también de su propia subclase el método *comprar_a_plazos*, como se ve en:

void insertar_entrada_mercancia_en_diario (…
void sumar_a_existencia_en_mercancia (…
void sumar_entrada_mercancia_en_mayor (…
void insertar_deuda_proveedor_en_diario (…
void insertar_nueva_factura_proveedor (…
void sumar_deuda_proveedor_en_mayor (…

En donde las tres primeras líneas son actualizaciones sobre la base de datos que sucedieron por la ruta 2, y las tres siguientes que sucedieron por la ruta 8. Este práctica de actualizar todo al momento, libros *Diario* , *Auxiliares y Mayor* en el mismo paso, sin dejar nada pendiente para después, se conoce como atomicidad transaccional, lo cual deja protegida la contabilidad contra eventuales descuadres [4, 5, 35].

En cualquier caso, una instancia siempre es un rastro dejado por la transacción acaecida, el cual queda íntegramente registrado en el *Diario*. El *Diario* es precisamente, según los preceptos de contabilidad financiera [4, 5, 35], el libro para el registro detallado de las transacciones sucedidas en una empresa, la bitácora en donde quedan consignadas minuciosa y absolutamente todas las transacciones instanciadas diariamente, con lo cual el MODRO expuesto adquiere características temporales.

Por lo tanto, estructuralmente cada instancia se representará por medio de tuplas (filas) insertadas en el *Diario*, por medio de atributos

específicos y renovados (columnas de débitos o créditos) en el *Mayor*, y por medio de atributos específicos y también renovados en los *Auxiliares*. O sea que cada instancia dejará la base de datos en un nuevo estado consistente, pues funcionalmente es una transacción, pero una transacción empresarial que es mayor y más compleja que una simple transacción informática.

En suma, todo el modelo conceptual es el súper tipo, es decir el MODRO, que modela a la organización entera con un súper método como su misión. Además, el MODRO obra como modelo de componentes con mayor grado de precisión. Luego vienen los conjuntos referenciales consistentes en sub esquemas o vistas externas, las cuales son diferentes a las vistas convencionales aquí se conciben como enlazadas. En realidad cada conjunto referencial es una vista externa de todo el modelo conceptual, visible sólo al área pertinente, es decir un tipo. Después vienen las clases y las subclases, que pueden constituir componentes de software, con las cuales se implementan los tipos. En la tabla 1 se presenta una organización de los conceptos en columnas equivalentes.

Concepto	*Teoría*		
	Conjuntos	**Bases de datos**	**Objetos**
	Universo	*Modelo conceptual*	*Súper tipo con súper método (modelo de componentes)*
	Conjunto referencial	*Vista externa (enlazada)*	*Tipo con signatura de método*
	Subconjunto	*Sub vista (enlazada)*	*Clases y subclases con métodos o Componentes*

Tabla 1. Conceptos planteados.

3. Trabajos Relacionados.

Quien explícitamente observó y cuestionó por primera vez la fragmentación de los procesos en la organización, fue Michael Hammer [1, 16], proponiendo una metodología de integración que denominó Reingeniería. En síntesis ésta consiste en la integración de todas las actividades que conforman un proceso, en unas pocas y nuevas actividades, rediseñando totalmente el proceso, lo cual implica frecuentemente la sustitución de paradigmas.

Otro protagonista de la Reingeniería es James Champy, que junto con Michael Hammer, escribió en 1993 el libro que desencadenó la nueva ola de la consultoría, provocando una revolución en la gestión empresarial conducente a la integración [1].

Pero, si la Reingeniería es la metodología de integración de los procesos, los BPMS y los flujos de trabajo (*workflow*) son la tecnología.

En efecto, dicha tecnología tiene como objetivo mejorar la eficiencia a través de la gestión computarizada de los procesos que se deben modelar, automatizar, integrar, monitorizar y optimizar de forma continua [25, 26].

En general, trabajos como el de Booch [11], entre otros, son considerados importantes para el desarrollo de software dentro del dominante paradigma de orientación a objetos. Así mismo, la contribución del grupo ODMG liderado por Catell [22] ha estandarizado un modelo de objetos que abre camino hacia el desarrollo de Sistemas de Administración de Bases de Datos con Orientación a Objetos (OODBMS en Inglés). Sin embargo, el paradigma de orientación a objetos por sí sólo no permite satisfacer todos los requisitos que el software, cada vez más complejo, debe cumplir. Una de las limitaciones que se presentan es la existencia de determinados asuntos (*loggin*, seguridad, persistencia,…) que se resisten a ser encapsulados en las clases tradicionales porque se repiten y son ortogonales, generando código desparramado y entrealazado. Si bien el MODRO reduce este problema replanteando clases con grano de mayor grosor, los trabajos de Laddad [28], Shukla [29] y Lesiecki [30] contribuyen desde la Programación Orientada a Aspectos (AOP) con soluciones, que aunque no aumentan el grosor de la granularidad, si son complementarias.

Otra limitación de la orientación a objetos consiste en la reutilización, que ha sido una de sus mayores promesas. El problema aquí estriba también en la concepción tradicional de clases y objetos muy específicos y de grano muy fino para ser efectivamente reutilizados. La intención del MODRO es aumentar el grosor de la granularidad coincidiendo con los trabajos de Szyperski [31], Councill & Heineman [32] y Veryard [33] y en general con los objetivos de la Ingeniería del Software Basado en Componentes (CBSE por sus siglas en inglés).

4. Perspectivas del Modelo

Con la reingeniería y los BPM´s (*Business Process Management*) se resuelve la *primera, segunda* y *tercera,* aunque no la *cuarta fragmentación*, puesto que sólo enfrentan la funcionalidad de la organización olvidando su estructura [1, 25, 26]. Con los modelos relacionales actuales se podría resolver, aunque con gran dificultad, la *primera, segunda* y *tercera fragmentación*, pero sería imposible resolver la *cuarta fragmentación*, puesto que sólo enfocan la estructura empresarial ignorando su funcionalidad. Con los modelos de objetos actuales tampoco se resolvería la *cuarta fragmentación* puesto que no enfocan la verdadera funcionalidad empresarial, que es transaccional,

ni la verdadera estructura empresarial, polarizada en los libros de contabilidad *Mayor* y *Diario*. Los MODROS aquí planteados, sí pueden resolver todas las cuatro fragmentaciones señaladas. Sin embargo, esta solución implica un nuevo problema: no existe un lenguaje apropiado para describir las clases y las subclases explicadas, con todas las implicaciones de la teoría de Sistemas, de la teoría de Objetos, de la teoría de Componentes, de la teoría de Bases de datos, de la Reingeniería y de la Contabilidad empresarial.

5. Conclusiones

Siendo las transacciones una modalidad de proceso, y estando los MODROs cimentados en transacciones, es posible generalizar estos MODROs para que no sean sólo transaccionales, no que sirvan para cualquier clase de proceso. La diferencia entre un proceso no transaccional y una transacción, radica en que ésta termina con asientos de contabilidad, mientras que un proceso no transaccional adolece de ellos. En consecuencia, los MODROs para procesos no transaccionales carecen de los polos característicos de los MODROs transaccionales. En otras palabras, al menos en el sentido contable, un proceso no transaccional es menos complejo que uno transaccional, por lo que un MODRO de procesos no transaccionales está implícito y resulta relativamente más elemental y simple, que los MODROs presentados en este artículo. Lo importante es saber reconocer en procesos completos, los diferentes métodos de clases y subclases como las expuestas aquí, superando así el primitivismo y la inadecuada fina granularidad de los métodos convencionales. Además, encapsulando procesos enteros dentro de estructuras enteras se ofrece una opción para componentes software.

Aunque el paradigma de Sistemas no es nada nuevo, la conclusión más importante es que con el MODRO el paradigma de Objetos evoluciona hacia el paradigma de Sistemas. En general, la concepción de Sistema lo describe como un conjunto de elementos u objetos interrelacionados entre síde forma tal que cada uno afecta el comportamiento de todo el conjunto. Tales elementos, en teoría de Objetos, suelen ser auto contenidos puesto que cada uno encapsula su propia funcionalidad, lo que impide su comportamiento como conjunto. A lo sumo, se puede llegar a tener objetos intercomunicados, pero cada uno con su propia funcionalidad individual. De ahí que un conjunto de objetos, aun estando intercomunicados, no pueda entenderse como sistema, excepto que dicho conjunto también encapsule su propia funcionalidad. De ser así, la teoría de objetos puede extrapolarse a la teoría de sistemas, originando un paradigma más evolucionado que el paradigma de objetos: El paradigma de Sistemas. Así el paradigma de Sistemas abarca el paradigma de Objetos, pero no al contrario.

Finalmente, aquel concepto informático en el que las transacciones eran simples unidades lógicas de trabajo que llevaban una base de datos de un estado consistente a otro estado consistente [13], ha sido replanteado en este artículo, recuperándolo del concepto de transacción en los negocios y en la contabilidad empresarial.

Referencias

[1] Hammer Michael; Beyond Reengineering: How the Processed Centered Organization is Changing Our Work and Our Lives; Harper Business; ISBN: 0 88730 880 5, 1996.

[2] Smith Adam; The Wealth of Nations; http://www.readprint.com/work 1384/Adam Smith.

[3] Jerry J. Weygandt, Donald E. Kieso and Paul D. Kimmel; Financial Accounting: Tool for Business Decision Making, Fourth edition, ISBN: 0 471 07241 9, 2003.

[4] Paul Steinbart & Marshall Romney; Accounting Information Systems; Pearson, 10th Edition, ISBN: 0 1319 6855 6, 2005.

[5] George Bodnar & William Hopwood; Accounting Information Systems; Pearson, 9th Edition, ISBN: 0 1312 2851 X, 2003.

[6] Whitten Jeffrey L., Bentlby L. D., Dittman Kevin C.; Analysis Systems and Design Methods; McGraw Hill, th edition; ISBN: 0072474173, 2004.

[7] Korth Henry F. & Silberschatz Abraham; Databases Foundations; McGraw Hill; 1993.

[8] Martín James & Odell James; Objects Oriented Analysis and Design; Prentice Hall; 1994.

[9] Pressman Roger S; Software Engineering: A Practitioner's Approach; sixth edition, McGraw Hill, ISBN 0 07 285318 2, 2005.

[10] Date C. J., An Introduction to Database Systems, 6th Edition. Addison Wesley 1995.

[11] Booch Grady; Object oriented design with applications; The Benjamin/Cummings Publishing Company, Inc;1990.

[12] Kroenke David M.; Object Oriented Database Processing; Prentice Hall 2002, (Slides).

[13] Marqués Merche; Bases de datos orientadas a objetos; Universitat Jaume I de Castelló; 2002. http://nuvol.uji.es/~mmarques/e16/teoria/cap2.pdf.

[14] Drucker Peter; Management: Tasks, Responsibilities, Practices; Harper and Row Publishers Inc.; ISBN: 0887306152, 1985.

[15] Date C. J.; An introduction to databases systems, 7th Edition, Addison Wesley Longman, Inc.; 2000; ISBN0 201 38590 2.

[16] http://www.hammerandco.com/

[17] Bertino Elisa & MartinoLorenzo; Object Oriented Database Systems; Addison Wesley Publishing Company; 1993.

[18] Marín Ruíz Nicolás; Tesis doctoral "Estudio de vaguedad en los Sistemas de Bases de Datos Orientados a Objetos"; Universidad de Granada, España; 2001.

[19] Mengchi Liu & Tok Wang Ling; A Logical Foundation for Deductive Object Oriented Databases; ACM Transactions on Database Systems, Vol 27, Nº 1, March 2002, Pages 117 151.

[20] Capretz Luis Fernando; A Brief History on the Object Oriented Approach; ACM SIGSOFT, Software Engineering Notes, Vol 28 Nº 2, March 2003 Page 1.

[21] Rahayu J. Wenny, Taniar Davis & Xiaoyan Lu; Aggregation Query Model for OODBMS; ACM Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications Volume 10, Sydney, Australia, Pages: 143 – 150, 2002.

[22] R. G. G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Velez; The Object Data Standard: ODMG 3.0; 2000.

[23] Grady Booch, James Rumbaugh, and Ivar Jacobson; Unified Modeling Language User Guide, 2nd Edition; Addison Wesley, Pearson Education, ISBN: 0 201 57168 4, 2005.

[24] James Rumbaugh & Ivar Jacobson & Grady Bocch; The unified Modeling Language, Reference Manual; Addison Wesley; 1999.

[25] Laury Verter; BPM: The promise and the Challenge; ACM Press, Queue Volume 2, Issue 1 March 2004.

[26] Howard Smith & Peter Fingar; BPM´s Third Wave; ISBN0929652339, www.bpm3.com, 2003.

[27] http://www.holon.se/folke/kurs/Distans/Ekofys/Recirk/Eng/holarchy_en.shtml

[28] Laddad Ramnivas; http://www.javaworld.com/javaworld/jw 01 2002/jw 0118 aspect.html.

[29] Shukla Dharma & Fell Simon & Sells Chris; http://msdn.micro soft.com/msdnmag/issues/02/03/AOP/print.asp.

[30] Lesiecki Nicholas; http://www 106.ibm.com/develo perworks/java/library/j aspectj

[31] Szyperski Clemens & Gruntz Dominik & Murer Stephan; Component Software Beyond Object Oriented Programming; Addison Wesley/ACM press, ISBN0 201 74572 0, second edition, 2002.

[32] Councill William T & Heinemann George; Component Based Software Engineering: Putting the pieces together; Addison Wesley professional; ISBN 0 201 70485 4, 2001.

[33] Veryard Richard; The Component Based Business: Plug and play; Springer, 2000; ISBN 1852333618.

[34] http://www.iasb.org/.

Nota: los sitios Web referenciados fueron accedidos satisfactoriamente el 20 de Enero de 2009.