(Re) Composición de modelos ER con Idioms

Juan Marcelo Flores Soliz
marcelo@memi.umss.edu.bo

Pablo Azero Alcocer
pablo@memi.umss.edu.bo

Programa MEMI, FCyT
Universidad Mayor de San Simón
Cochabamba, Bolivia

Resumen

El Paradigma OO se ha consolidado como referente tecnológico para el desarrollo de software, unos de sus mayores méritos es que provee un marco conceptual para modelar casi cualquier aspecto de una forma muy natural. Sin embargo, la ventaja que su carácter semántico ofrece, pierde efecto cuando se diseña aspectos de persistencia a tiempo de implementar estos en una base de datos relacional. Aún cuando existen métodos de diseño maduros, patrones de diseño y otras herramientas, estos no reflejan una solución especializada para el manejo de la persistencia y mas aún, ellos no son una referencia para el diseño ER y relacional.
Este estudio presenta un lenguaje de Idioms [1] como un subconjunto de ER, que define estructuras sintácticas con alta cohesión semántica, la cual no es considerada en el lenguaje ER original, el lenguaje de Idioms adicionalmente provee un modo composicional de construcción sorprendentemente similar a las abstracciones usadas con el paradigma OO, además de ofertar estructuras como referente de correctitud sintáctica y semántica.

Keywords: Reuso, Refactorización, Patrones, Abstración, Semántica, modelos ER

Introducción

Una de las grandes ventajas de ER es su riqueza representativa como lenguaje, sin embargo esa riqueza otorga demasiados grados de libertad a los desarrolladores, tal que es difícil encontrar estructuras de referencia para asegurar la calidad de los modelos ER. Esto contrasta con el paradigma OO cuya ventaja semántica se traduce en definición de abstracciones primitivas de alto contenido estructural y semántico, patrones, refactorización, etc. Es claro sin embargo que los conceptos aplicables en el paradigma OO no son aplicables cuando se tiene un modelo ER. Los modelos ER tienen estructura, abstracciones y semántica diferente de los modelos OO, la eficiencia de un modelo y programas OO están dados por la correcta aplicación de los conceptos OO, por el contrario, la correctitud y eficiencia de un modelo ER está dado principalmente por la aplicación de normalización y conceptos de navegabilidad.

Este estudio analizará las posibilidades de refactorización de modelos ER basadas en estructuras del mismo lenguaje llamadas Idioms, estos se constituyen en estructuras básicas como elementos de referencia para la refactorización.

En la sección 1, se da un marco referencial para los Idioms, pues estos están detallados en [1].

En la sección 2 se presenta los metamodelos que definen de manera formal los lenguajes de Idioms y el lenguaje ER como lenguaje nativo. También se presenta una arquitectura de referencia que define la dependencia de los metamodelos.

En la sección 3 se presenta a los idioms como lenguaje cuyas estructuras pueden ser referentes de correctitud y calidad de modelos ER. También se presenta una estrategia de trasformación de un modelo ER sin idioms a un modelo ER mejorado y aceptado en el lenguaje de Idioms.

En la sección 4 se presentan algunas conclusiones acerca de los idioms, su modelo, su formalismo y sus ventajas.

1. Idioms como referente estructural

Idioms en ER se refieren a estructuras sintácticas en lenguaje ER con un fuerte componente semántico [1]. Bajo la perspectiva de Idioms, las construcciones ER tienen una semántica correcta solamente si encajan en la

semántica de algún Idiom. Los Idioms se caracterizan por conformar un catálogo de estructuras con aspectos semánticos precisos además de poseer cualidades deseadas sobre algunas formas normales y eficiencia en la navegabilidad [1]. Entonces las labores de refactorización en modelos ER se pueden enmarcar a la búsqueda de estructuras cuya "medida de distancia" de los Idioms se convierta en métrica de su calidad, con referencia a su contenido semántico y navegabilidad.

La refactorización en modelos ER podría tener algunas consideraciones adicionales:

- Actualmente es bastante común realizar modelos ER ayudados con algún tipo de asistente de desarrollo de software (CASE). Estos asistentes implementan herramientas que podrían guardar los elementos de los modelos ER en algún tipo de repositorio o Base de Datos.
- Es bastante común usar dialectos ER bastante dirigidos a su implementación en algún tipo de Bases de Datos en particular.

Con estas consideraciones, la refactorización podría realizarse de forma bastante sistemática, si se piensa en la inspección de los metamodelos[1] y/o metadatos[2], en cuyas instancias se realiza el modelo ER, tal que se puede buscar estructuras con malos olores (bad smells)[12] y verificándolos con alguna medida de distancia. Es decir; existe buena posibilidad de obtener resultados determinísticos referidos a verificación de estructuras o fragmentos de modelos ER, una vez que las estructuras de Idioms proveerán de forma correcta la sintaxis y semántica de las estructuras en el modelo analizado.

Dado que los Idioms son estructuras que se pueden representar sintácticamente de forma precisa, definen de forma completa un lenguaje subconjunto del lenguaje ER. El lenguaje de los Idioms recorta de forma precisa las construcciones del lenguaje ER que no contienen una semántica precisa o necesaria, es decir, supone que la estructura de los Idioms dará la semántica suficiente y completa para todos los aspectos posibles de modelar en el ámbito de sistemas de información

[1] Estructura genérica que soporta la instanciación de los modelos

[2] Instancias de una estructura genérica, almacenadas y manejadas como cualquier dato.

2. Idioms vs. ER

La arquitectura de los idioms sigue una concepción de extensibilidad, puesto que los conceptos de los idioms son formados por conceptos nativos de ER, entonces su lenguaje también es derivado del lenguaje ER, en otras palabras, los idioms extienden el lenguaje ER

En la figura *1*, se traza de forma explícita una relación de dependencia del modelo de Idioms hacia el modelo ER. La relación de dependencia es básicamente por la necesidad del modelo de idioms de alimentarse de definiciones y/o implementaciones realizadas por el modelo ER, tal que son reusadas por el modelo de idioms.

Para los idioms, todos los conceptos de ER son válidos más no así la totalidad de construcciones que se puedan realizar con éstos. La idea es definir de forma concreta cuales son las construcciones válidas en el modelo de idioms de forma que su definición reescriba el lenguaje original ER tornándolo en un lenguaje limitado por las construcciones posibles/ válidas del modelo de idioms.

Los atributos de las clases en el metamodelo representan los elementos estructurales que los constructores deben preservar.

Los roles *owned* y *owner* definidos en la *figura 2*, son dados por la participación de una entidad en una relación. El metamodelo ER prevé que las relaciones *owned* se realizan a través de una llave foránea cuando las relaciones son distantes entre las entidades participantes, y cuando las relaciones son cercanas, crean una dependencia entre ellas, definiéndose estas en una relación especializada denominada *dependency* cuya relación *owned* se redefine a través de una llave primaria en lugar de la llave foránea.

En la *figura 3,* se define un estereotipo a ser usado en la definición de cada idiom. Este estereotipo es llamado *idiom*, y es una extensión particular a UML realizada mediante su mecanismo *Profile* [2]. Cada estructura en el metamodelo que representa un idiom llevará este estereotipo identificándolo del resto de las clases que son constructores de estos idioms. Cada definición de cada idiom, especializa de una manera particular su estructura, estableciendo sus características de obligatoriedad de la relación y cardinalidad de la relación.

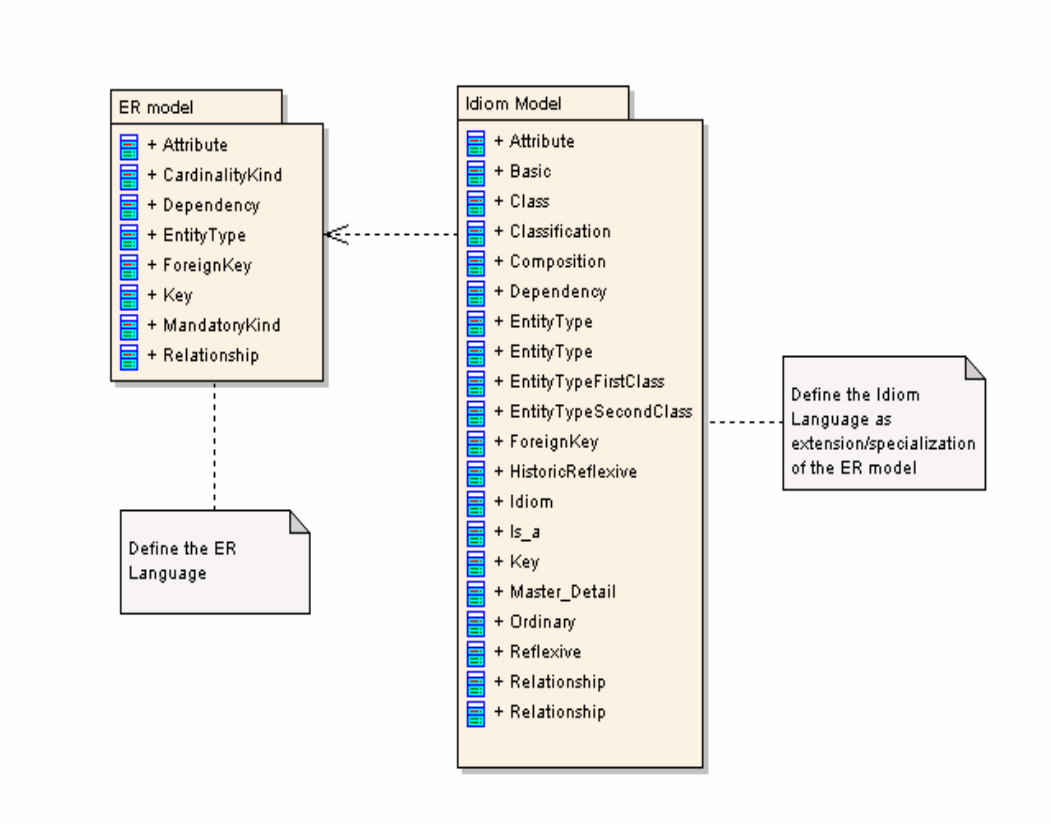


Figura 1. Arquitectura de referencia

Según sea una relación ordinaria o de dependencia, la especialización parte de las definiciones realizadas en el metamodelo ER respectivamente.

Para la definición de los idioms es necesario distinguir dos categorías de entidades tipo: tipos entidades de primera clase y tipos entidades de segunda clase.

La primera categoría de tipos entidades denota a aquellos tipos que podrían conformar puertos de ensamble con otros idioms. Cada idiom debe prever su posibilidad de composición con otros idioms definiendo sus puertos de acoplamiento de una forma precisa. De hecho la segunda categoría de tipos entidad, denota a tipos que participan en la construcción de un idiom que no tienen la cualidad de convertirse en puertos de ensamble con otros idioms.

Ha sido necesario redefinir algunas clases del metamodelo ER para definir de forma precisa las características estructurales de las dos categorías de tipos entidad.

El metamodelo de idioms también define la semántica de la participación de las entidades en una relación. Esta definición es observada en los nombres estáticos de los roles de cada entidad participante de un idiom.

Estos roles son importantes, pues su nombre denota de forma intuitiva la semántica de la participación de una entidad en una relación, más aún, la búsqueda del concepto descrito con el rol en el UoD[3] [5], ayudará fuertemente en la definición del uso del idiom apropiado.

[3] Universe of Discourse

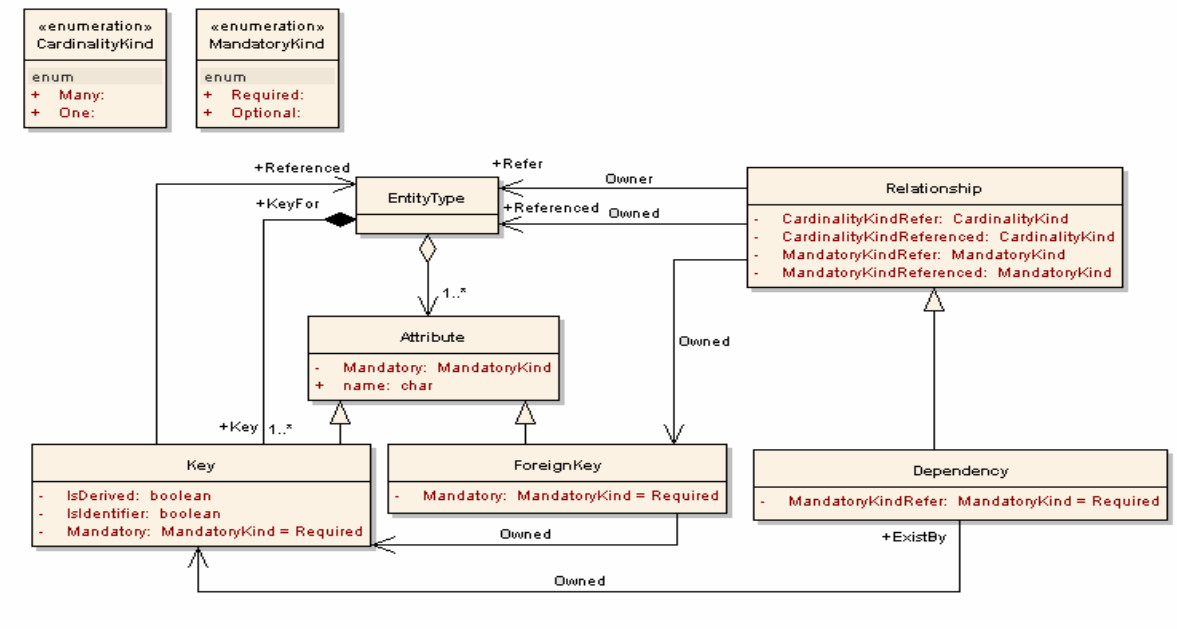


Figura 2. El Metamodelo ER

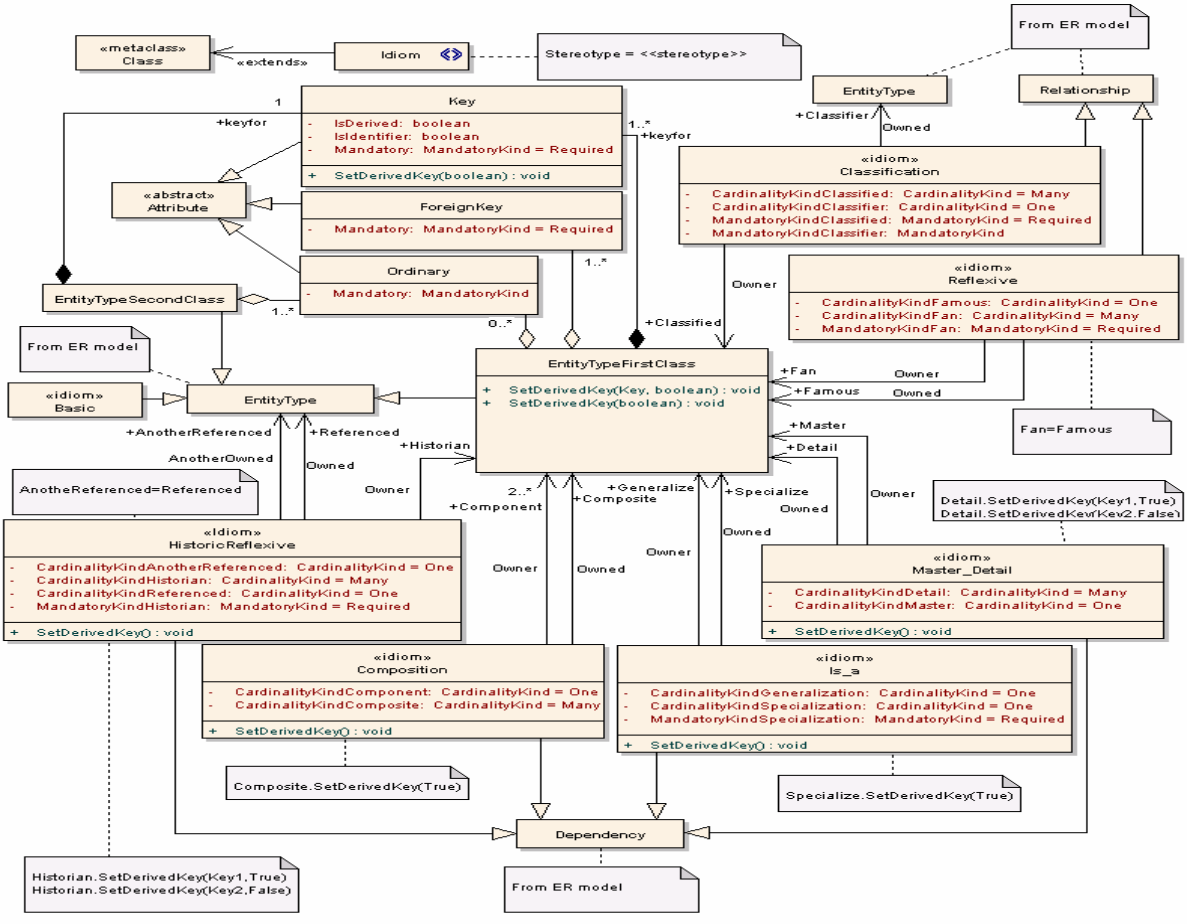


Figura 3. El metamodelo de Idioms

3. Refactorización

La construcción de un lenguaje que define los Idioms otorga un soporte formal para la refactorización de modelos ER. La revisión sobre lo síntomas en el modelo que se aplica refactorización se realizará con las reglas sintácticas del lenguaje de los Idioms, de hecho, es muy probable que gran parte de la refactorización se realice por una simple revisión sintáctica del modelo ER en la contrastación con la sintaxis de los Idioms.

Sin embargo, en modelos ER no existe una manera determinística de verificar que un componente en particular: una entidad tipo o un atributo tenga una semántica correcta, el significado de las entidades tipo y/o atributos que componen el modelo es totalmente dependiente al contexto del problema. La estrategia de refactorización con Idioms verifica y propone correctitud sintáctica y correctitud semántica en estructuras del modelo, es decir; en fragmentos del modelo y no así en cada elemento individual.
La refactorización de un modelo ER logrará un modelo con similares capacidades de persistencia, pero con capacidades probadas de navegabilidad, flexibilidad y coherencia semántica.

La refactorización con Idioms como tal no sólo consiste en buscar errores o malos olores en los modelos inspeccionados [12], sino también en proponer transformaciones al modelo que dan una continuidad en los servicios que proporcionan los aspectos a refactorizar.

Para el caso de refactorización con Idioms en modelos ER, se propone seguir la siguiente estrategia:

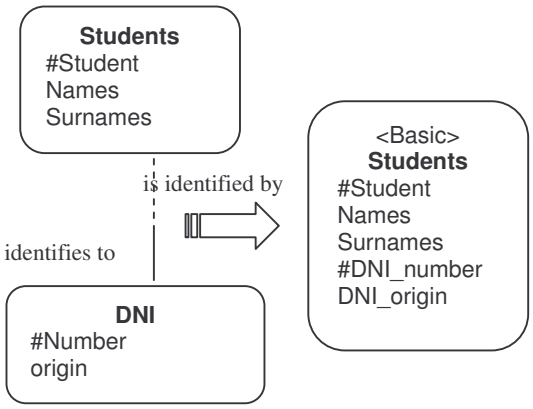
1. Inspeccionar los modelos mediante una verificación de los tipos y relaciones que componen el modelo ER, tal que se marque como *estructura en cuarentena* a aquellas estructuras que no componen un idiom, de aquellos definidos en el metamodelo de idioms.
2. Para cada *estructura en cuarentena*, buscar sus características en el catálogo de estructuras con malos olores que se describen en la siguiente sección.
3. Reemplazar cada entidad tipo y relaciones de la *estructura en cuarentena* con entidades tipo y relaciones nuevas que incorporan los cambios recomendados para ese caso en particular.
4. Verificar nuevamente si todas las estructuras componen algún idioms descrito en el metamodelo de idioms. Si no es verdad iterar nuevamente la refactorización. Si es verdad fin de la refactorización.

A continuación se presenta un catálogo de muestra de síntomas y estructuras recomendadas que trasforman una estructura deficiente en una estructura aceptada por el lenguaje de idioms, incorporando las cualidades descritas para estos.

Las estructuras descritas a continuación, toman en cuenta modelos tal que su construcción por lo menos oferta algunas características de persistencia de los datos que representan y que es deseable mantener luego de la refactorización.

Refactorización usando el Idiom Básico

El *Idiom Básico* es la unidad mínima con la que se puede construir modelos ER.



Transformación 1 : Una relación uno a uno en idiom básico

Como primer paso de la refactorización con Idioms, se puede efectuar una búsqueda de relaciones uno a uno, debido a que estas son fáciles de encontrar y fáciles de corregir.

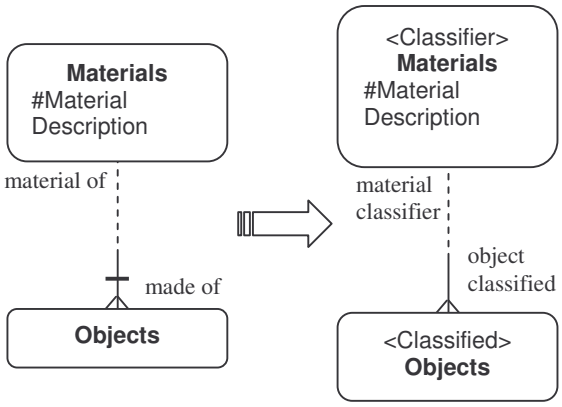
Siempre que se encuentren dos tipos entidades con una relación uno a uno, deberán fusionarse dichos tipos en uno solo, combinando los atributos de ambos en un solo tipo entidad o *Idiom Básico*. (*Transformación 1*). La existencia de una relación uno a uno es inadecuada e ineficiente debido a que origina duplicación de la información en las dos entidades participantes

de la relación, y requiere de un JOIN que implica un costo adicional en las consultas.

Refactorización usando el Idiom Clasificación

El *Idiom clasificación* está compuesto por dos tipos entidades: el tipo entidad "clasificador" tiene generalmente dos atributos: un identificador o llave propia, y un atributo de descripción. Su función, como su nombre lo indica es de clasificar o catalogar a otros tipos entidades, permitiendo registrar un conjunto finito y definido de valores para su utilización en la instanciación del tipo entidad "clasificado" [1].

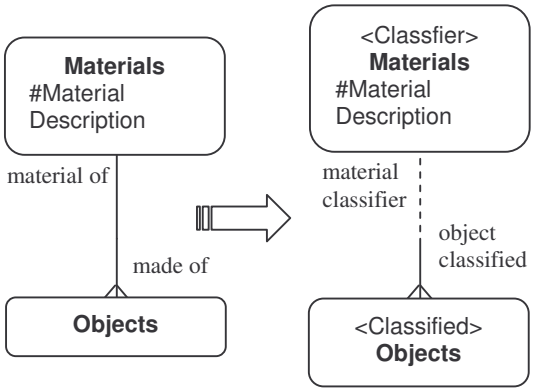
La existencia de una entidad "clasificador", no depende de ninguna otra entidad de otro tipo.



Transformación 2: No debe existir una dependencia entre entidades del clasificador y del clasificado.

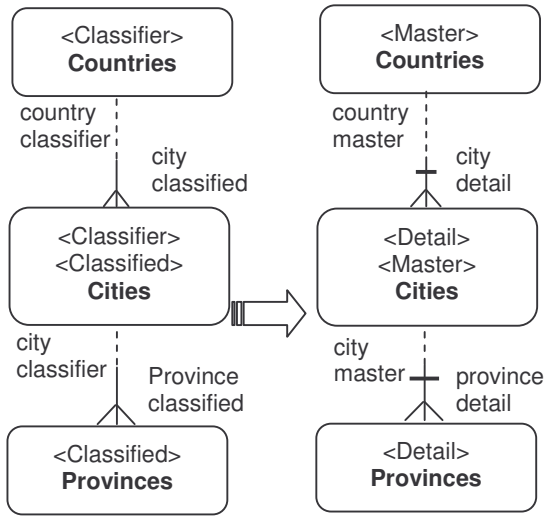
En el modelo se debe verificar que todos los clasificadores mantengan una relación no dependiente con los tipos entidades que clasifican, un problema inmediato será el caso donde se encuentre una relación de dependencia entre dos tipos entidades donde uno ellos tenga todas las características de un tipo entidad clasificador. Para solucionar este problema se deberá eliminar la dependencia, generalmente dada por una llave primaria heredada (*Transformación 2*).

Otro problema también puede ser ocasionado por la opcionalidad errónea del *Idiom clasificación*, es decir, si se tiene una relación completamente obligatoria en el Idiom, se tendrá un problema a la hora de instanciar los valores del "clasificador". Cada valor del "clasificador" deberá ser utilizado obligatoriamente por alguna instancia del tipo entidad "clasificado". Para evitar este problema es preferible mantener independiente al "clasificador" permitiendo que su relación con cualquier tipo entidad sea opcional *(transformación 3)*



Transformación 3: La obligatoriedad de la relación de un Idiom Clasificación ocasiona un problema en la instanciación

La clasificación múltiple, es decir, el caso donde un "clasificador" es a su vez "clasificado", deberá ser tratado con mucho cuidado, ya que las entidades de cada "clasificador-clasificado" serán multiplicadas por el número de entidades que la clasifica. Esta situación se verá reflejada a momento de realizar las consultas cuya respuesta tendrá una lista de valores muy extensa y con el inconveniente de tener que hacer un recorrido largo y secuencial para obtenerlos a tiempo de diseñar la consulta.



Transformación 4: La múltiple clasificación crea un problema a la hora de desplegar información.

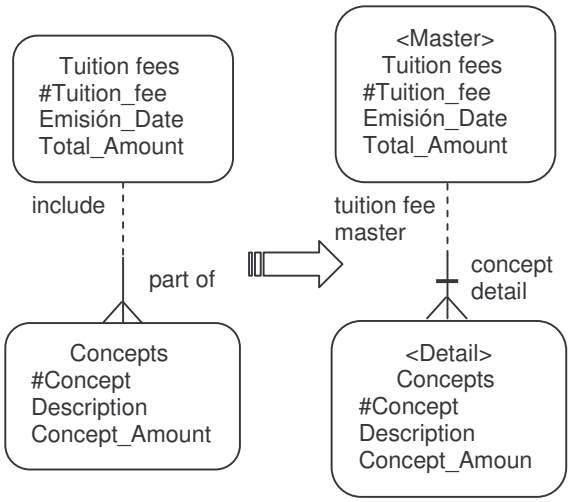
En modelo del ejemplo *Transformación 4*, de múltiple clasificación se reflejará durante las consultas como tres listas de valores, el problema radica en la obtención de las dos listas

subordinadas:"cities" y "provinces". El caso crítico será la lista de "provinces" ya que su búsqueda retornará el conjunto completo de provincias de todos los departamentos y de todos los países. Una lista de valores de esta magnitud no es útil.

Puede haber muchas formas de refactorizar esta estructura, por ejemplo; se puede utilizar el *Idiom Maestro-Detalle* como alternativa identificando la semántica de la dependencia y mejorar la navegabilidad a través del modelo. Esto facilitará el recorte de la lista de valores, dándole principalmente un contenido semántico coherente con la observación.

Refactorización usando el Idiom Maestro-Detalle

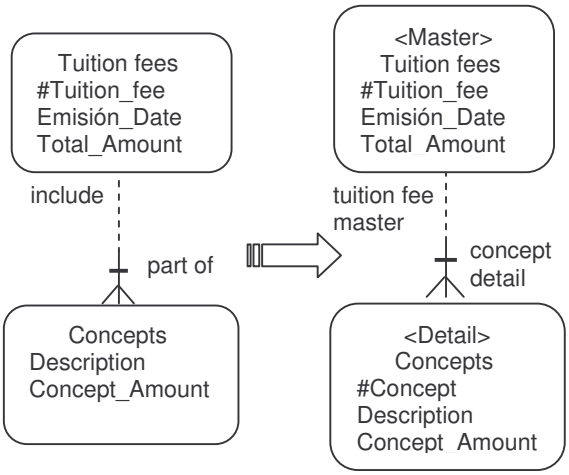
Un *Idiom maestro-detalle* expresa una situación de dependencia de una entidad con otra, donde las entidades que cumplen el papel de "Detalle" podrán existir sólo cuando exista una entidad "Maestro". El tipo entidad "Detalle" debe poseer en su definición, una llave primaria propia para posibilitar la creación de varias instancias por cada valor del tipo entidad "Maestro" (*Transformación 5*)



Transformación 5: Un Idiom maestro-detalle malinterpretado como clasificación.

Otro paso de la refactorización será la búsqueda de Idioms *de Clasificación* falsos, es decir, deberá analizarse los tipos entidades que cumplen el papel de clasificadores para encontrar "impostores", que por sus características deberían ser catalogados como Maestros y formar parte de un Idiom *Maestro-Detalle*. Para confirmar el problema, también convendrá estudiar las características del tipo entidad supuestamente "clasificada", para descubrir algún elemento que exprese la función de complemento que realiza este tipo entidad respecto a su correspondiente "Maestro".

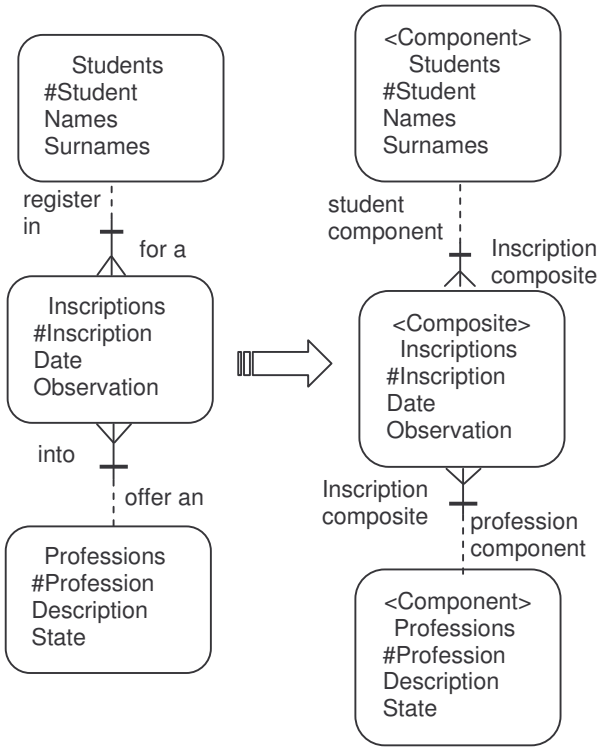
Todos los Idioms *Maestro-Detalle* del modelo deberán tener por lo menos una llave primaria propia en el tipo entidad que juega el papel de detalle. Antes de corregir este error deberá comprobarse que el supuesto Idiom *Maestro-Detalle* sujeto a corrección no sea en realidad un fragmento de un Idiom *Composición*, que en dicho caso no requerirá llave propia (*Transformación 6*).



Transformación 6; Ausencia de llave primaria en el Detalle.

Refactorización usando el Idiom Composición

El Idiom *composición* surge por la existencia de un evento que posibilita la relación entre dos o más entidades.
El tipo entidad "composición" es totalmente dependiente de los otros tipos entidades "componentes", este tipo composición no debe poseer una llave primaria propia para evitar que existan datos replicados en la composición con distintas llaves (*Transformación 7*). Puede darse el caso donde la existencia de una llave primaria en el tipo entidad composición indique la existencia de una mala reutilización de un tipo entidad que cumple el papel de "Detalle" compartido para dos tipos entidades distintos. En tal caso deberá crearse tipos entidades propios para cada componente y formar dos estructuras *Maestro-Detalle* independientes (*Transformación 8*).



Transformación 7: Una composición no debe tener una llave primaria propia.

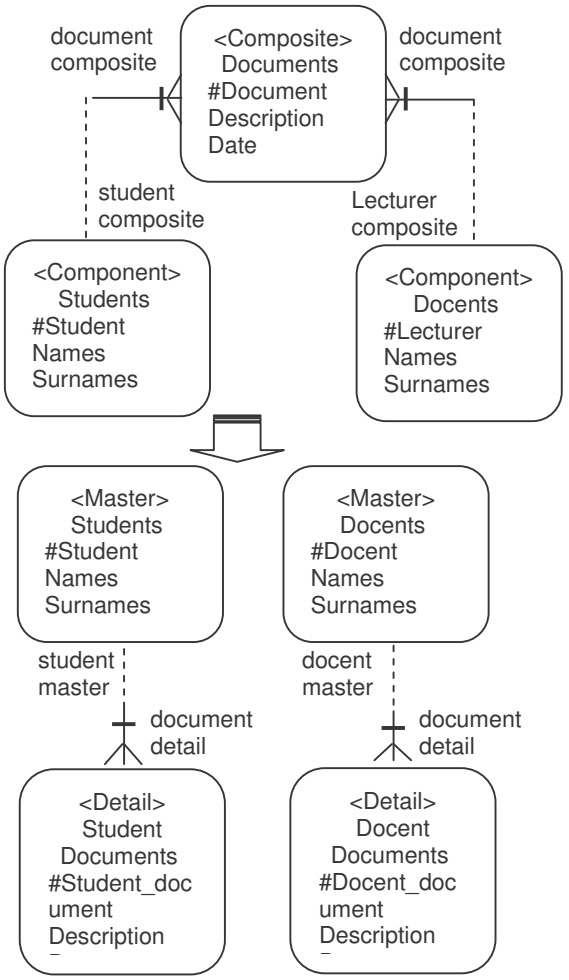
Una solución más correcta a este problema se describe en el apartado que trata sobre la refactorización usando el Idiom *is-a*.

Refactorización usando el Idiom Reflexivo simple

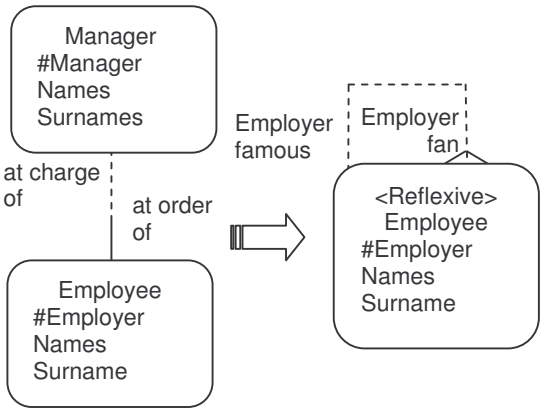
El Idiom *reflexivo simple* se presenta cuando existe alguna relación entre dos entidades de un mismo tipo.

Cuando se presente en un modelo relaciones de jerarquía entre dos tipos entidades se debe verificar si es posible fusionar estos tipos en uno solo, utilizando una relación reflexiva para expresar la jerarquía (*Transformación 9*).

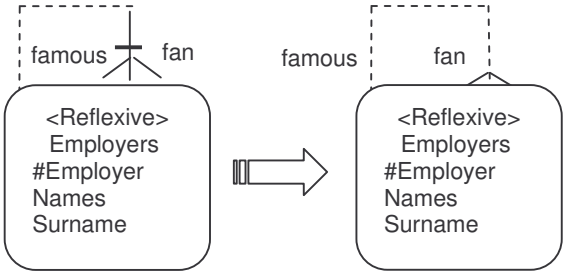
Un problema también surge en la dependencia funcional y opcionalidad definidas para la jerarquía, si estas son rigurosas la jerarquía deberá ser inicializada realizando algún truco, ya que no será posible tener nunca una entidad que no tenga un antecesor (*Transformación 10*).



Transformación 8: Composición utilizada para reutilizar un detalle.



Transformación 9: Idiom reflexivo simple para expresar jerarquía.



Transformación 10: Dependencia y opcionalidad rigurosa dificultan la creación de la raíz de la jerarquía.

Refactorización usando el Idiom Reflexivo histórico

El *Idiom reflexivo histórico* permite la existencia de relaciones especiales entre entidades de un mismo tipo, donde se requiera registrar información propia de la relación.
Un error podría darse con la utilización de dos tipos entidades casi idénticas, cuya composición represente la relación, para permitir registrar información propia de la misma.
Para solucionar este error podrán fusionarse los tipos entidades
"componentes" en uno solo, y utilizar el *Idiom reflexivo histórico* para permitir clasificar o crear atributos propios de la relación recursiva (*Transformación 11*).

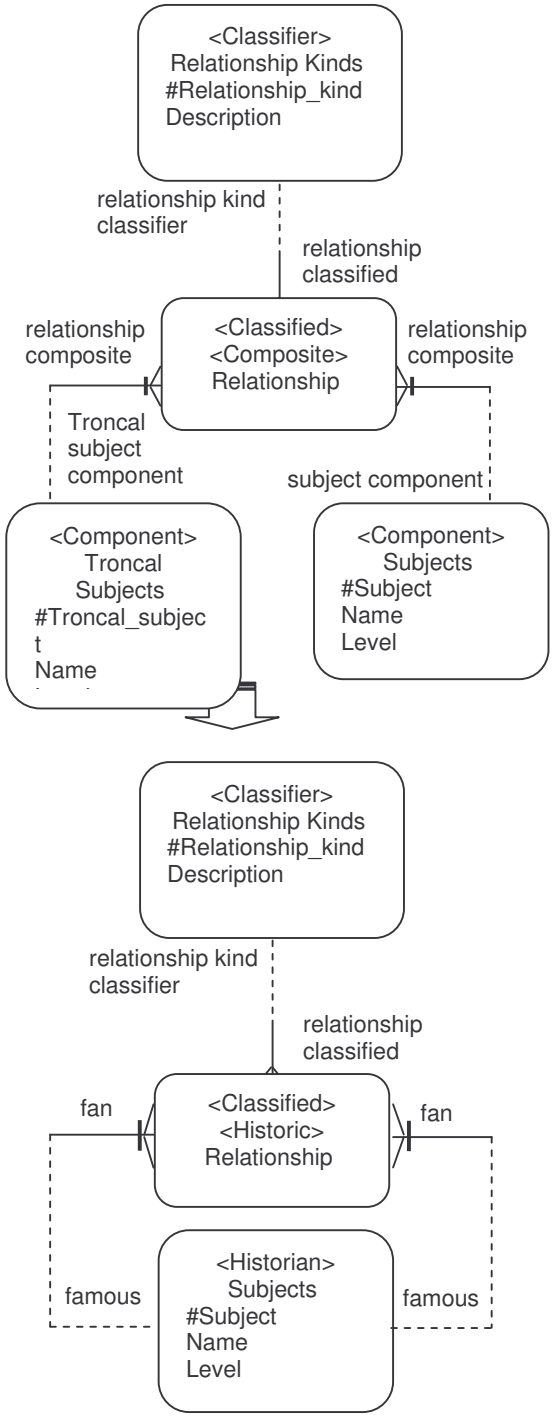
La opcionalidad y dependencia funcional ocasionan los mismos problemas mencionados en el apartado anterior.

Otro posible problema solucionable con este Idiom, surge en la construcción de un grafo dirigido utilizando dos relaciones recursivas para un mismo tipo entidad. Esta solución puede funcionar, pero conllevará problemas tecnológicos a la hora de la instanciación.

Con un *Idiom reflexivo histórico* se pueden construir grafos dirigidos, no-conexos, ponderados (*Transformación 12*).

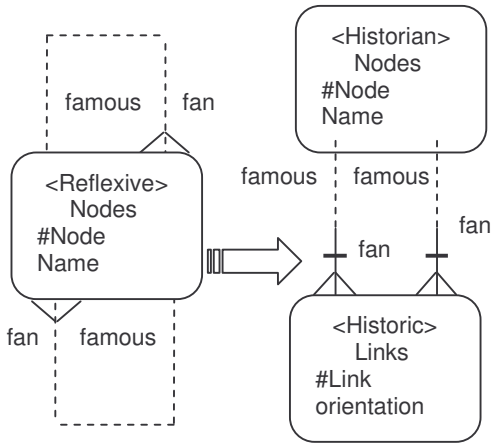
Refactorización usando el Idiom Is-a

El *Idiom Is-a* define una relación de generalización – especialización entre dos tipos entidades.
Uno de los grandes problemas a la hora de construcción de modelos ER es la implementación de la relación *Is-a*, históricamente los DBMS han ido implementando esta relación con una fuerte



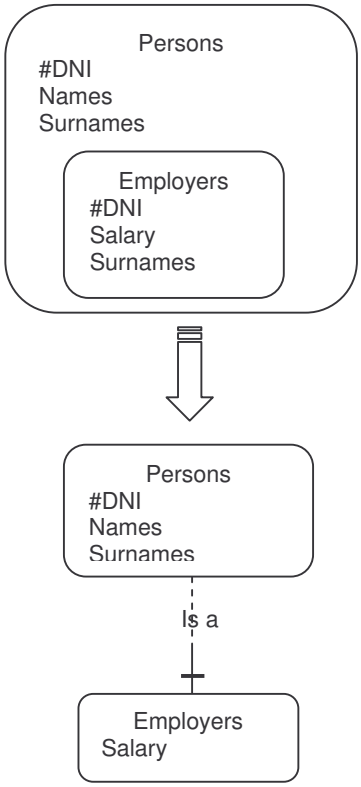
Transformación 11: Idiom reflexivo histórico para relaciones con atributos propios.

influencia y tendencia a incorporar el concepto de Herencia, muy manejado en el paradigma OO [7] [3], el inconveniente es que estas implementaciones han tratado de reflejar también



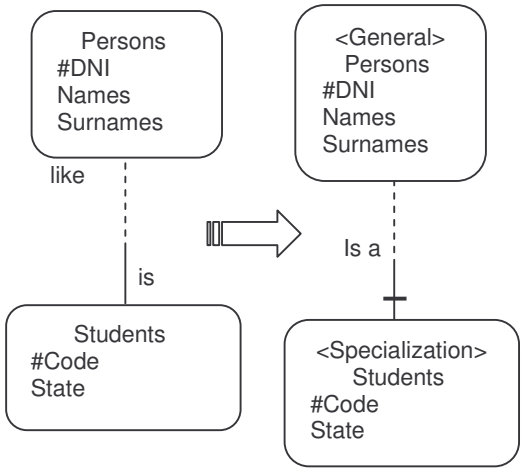
Transformación 12: Grafos dirigidos no-conexos ponderados

la estrategia de implementación de la programación OO, con el inconveniente de violar severamente algunas reglas de normalización.
En la definición presentada en el metamodelo de idioms, la relación *Is-a* es propuesta de forma semántica como una relación con características de dependencia, obligatoriedad y cardinalidad que son implementadas por cualquier DBMS, sin consideraciones de implementación de herencia.



Transformación 13; refactorización de relaciones de herencia (en p.e Oracle)con el idiom IS_A.

Es más, bajo la perspectiva de idioms las estructuras que buscan herencia en lugar de generalización-especialización son posibles de ser refactorizados a una estructura equivalente como lo es *Is-a* (*Transformación 13)*.
Esta sintaxis se refleja en el modelo relacional como dos entidades independientes, sin relación alguna, donde la entidad interior "hereda" los atributos propios de la entidad exterior.
El concepto de generalización va más allá de la herencia como técnica de reusabilidad utilizada comúnmente en programación OO. El Idiom Is-a refleja de manera más adecuada el concepto de generalización estableciendo una relación de orden entre las entidades participantes.

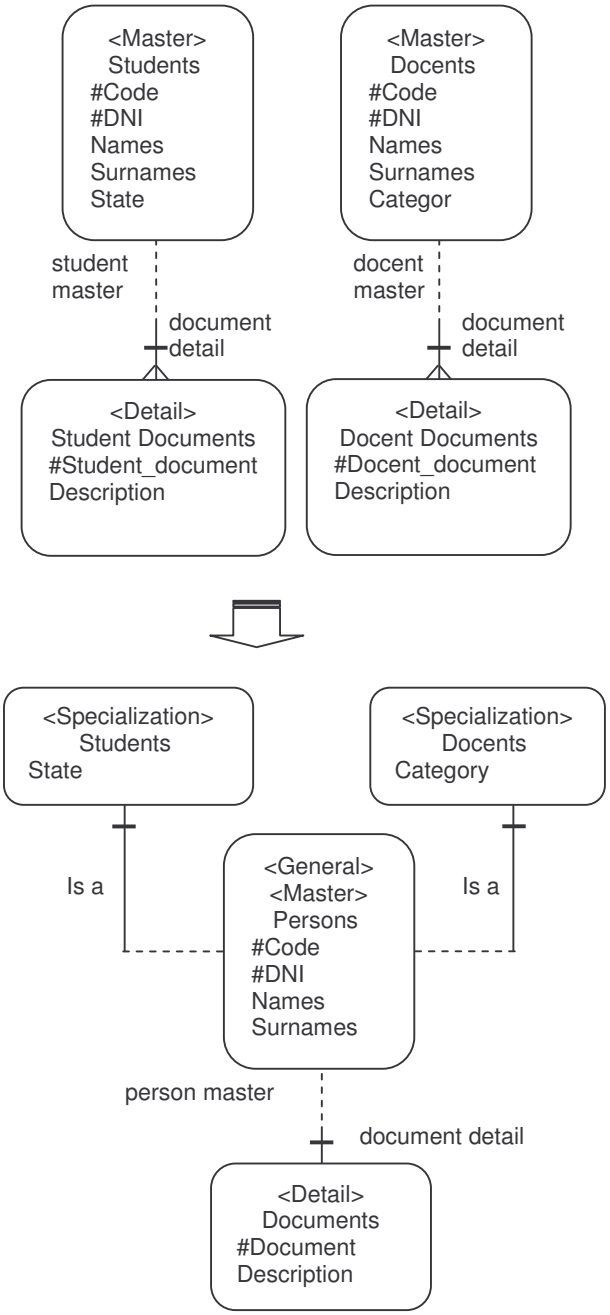


Transformación 14: Relación de generalización entre dos tipos entidades.

Cuando se encuentre una relación uno a uno entre dos tipos entidades se deberá analizar la estructura de ambos tipos buscando la posibilidad de que exista una relación de dependencia funcional que defina una generalización (*Transformación 14*).

Es posible encontrar características comunes entre dos tipos entidades, ya sean relaciones o atributos, que puedan ser generalizados en un solo tipo entidad padre con el que se pueda conformar el Idiom *is-a* (*Transformación 15*).

La semántica apropiada para el uso no difiere realmente de la semántica del uso de la relación de generalización del paradigma OO.



Transformación 15: Descubriendo características comunes para generalizar.

4. Conclusiones

El análisis semántico de modelos ER es posible mediante *Idioms*, el lenguaje de Idioms facilita enormemente y además da un marco formal a la búsqueda de estructuras con alguna deficiencia. Los logros obtenidos con la refactorización basada en Idioms, en modelos ER, da la posibilidad que las transformaciones o refactorizaciones no sólo incluyan aspectos de ER sino también de su implementación relacional, traduciéndose en transformaciones basadas en SQL.

En este trabajo se presentaron los metamodelos que definen los lenguajes de idioms y se presentan las transformaciones o refactorizaciones con una fuerte connotación sintáctica, sin embargo en realidad los idioms y la inmediata consecuencia de refactorizar a idioms da al modelo refactorizado un fuerte añadido semántico.

Una de las más grandes ventajas de realizar una verificación de calidad de modelos ER a partir de una revisión de fragmentos del modelo contrastándolos con el catálogo de Idioms, es que esta labor puede ser completamente automatizada.
Este estudio muestra solamente la refactorización cuando es realizada a nivel conceptual, es decir sobre los modelos ER, sin embargo es posible realizar refactorización basada en idioms a nivel físico, es decir a un nivel en el cual se tienen ya instanciadas las Bases de datos y administradas por un DBMS. Esta visión de la refactorización es mucho más delicada puesto que se debe refactorizar principalmente las consultas SQL asi como los programas SQL que crean y/o alteran la estructura del esquema de la Base de datos.

Adicionalmente el uso de Idioms otorga un estilo de diseño al modelador de Bases de datos tal que su trabajo se torna en elemento mismo de su propia validación. Los resultados de esta mejora se traducen en reducción del tiempo de diseño, eliminación de errores, eliminación de etapas de revisión y readecuación del modelo e incremento en la confianza del diseñador.

Las cualidades semánticas de los idioms hacen posible pensar que una vez definida su estructura, sea posible también definir su funcionalidad traducida en servicios que oferta la estructura a la aplicación que lo necesite, esto da por supuesto una nueva visión de integración objeto-relacional que los autores de este artículo están trabajando actualmente.

5. Referencias bibliográficas

[1] Idioms en ER, J. Marcelo Flores Soliz, 5ta Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del conocimiento JIISIC, 2006, http://memi.umss.edu.bo/idiomsER

[2] The Unified Modeling Languaje, Object Management Group, www.omg.org

[3] Diseño de Bases de Datos con UML, Paul Dorsey-Joseph R. Hudicka, Oracle Press, 1999

[4] Design Patterns, E. Gamma, R. Helm, R. Johnson and J. Vlissides. Addison-Wesley, 1995.

[5] Requirements Engineering: Frameworks for Understanting, R.J. Wieiringa, Wiley, 1996.

[6] UML y Patrones, Craig Larman, Prentice-Hall, 2nd edition 2002.

[7] Oracle Designer-database Generator, OraclePress, 2001

[8] Fundamentos de Sistemas de Bases de Datos, Ramez A. Elmasri-Shamkant B. Navathe, Addison Wesley, 3ra. edición, 2002

[9] The Entity-Relationship Model-Toward a Unified View of Data, Peter Pin-Shan Chen, ACM Transactions on DataBase Systems, Vol 1, Nº1.

[10] Evolution of Data Modeling for Databases, Shamkant B. Navathe, Communications of the ACM, Vol.35,Nº 9, 1992.

[11] Design Methods for Reactive Systems: Yourdon, Statemate and the UML, R.J. Wieringa, Morgan Kaufmann, 2003

[12] Refactoring- Improving the design of existing code, Martin Fowler, Addison Wesley, 1999.
[13] Analysis Patterns - Reusable Object Models, Martin Fowler, Addison-Wesley, 1999

[14] Patterns of enterprise application architecture, M. Fowler et al., Addison Wesley, 2002.

[15] Patterns, Martin Fowler, IEEE software magazine, March/April 2003

[16] A system Of Patterns, F. Buschman – R. Meunier – H. Rohnert – P. Sommerland – M. Stal, John Wiley & Sons, 1996