DIBAO: MÉTODO PARA EL DISEÑO DE LA BASE DE DATOS A PARTIR DEL MODELO ORIENTADO A OBJETOS

Thesis · December 2001

CITATION
1

READS
4,218

1 author:

Anaisa Hernández González
Universidad Tecnológica de la Habana, José Antonio Echeverría
66 PUBLICATIONS **114** CITATIONS

SEE PROFILE

ISPJAE
FACULTAD DE INGENIERÍA INDUSTRIAL
CENTRO DE ESTUDIOS DE INGENIERÍA Y SISTEMAS

DIBAO: MÉTODO PARA EL DISEÑO DE LA BASE DE DATOS A PARTIR DEL MODELO ORIENTADO A OBJETOS

Tesis presentada en opción al grado científico de Doctor en Ciencias Técnicas

Autor: Ing. Anaisa Hernández González, Ms.C.

Tutor: Dra. Sofía Álvarez Cárdenas

La Habana

2001

SÍNTESIS

En la calidad de los productos de software, tiene una gran importancia el diseño que de las bases de datos se realice.

Se describe un método para el DIseño de la BAse de datos a partir del modelo orientado a Objetos (DIBAO), formado por un modelo de persistencia y una capa persistente de clases.

En el modelo de persistencia se sistematizan los pasos para el diseño de la base de datos, se incluyen recomendaciones para obtener los comportamientos estático y dinámico de los objetos; así como el procedimiento a seguir para interpretar la información representada para su almacenamiento en ficheros, un gestor de objetos, o un gestor relacional u objeto/relacional, que permita o no trabajar con un lenguaje de alto nivel orientado a objetos. El modelo se basa en el modelo formal O^3 (Orientado a objetos y ontológico) y satisface el modelo de objetos de ODMG (**Object Database Management Group**).

Se propone la estructura de una capa persistente de clases que incluye las subcapas de especificación e interfaz, que aíslan a las clases del dominio del medio de almacenamiento y responde al modelo de persistencia propuesto..

Se describen las herramientas automatizadas que soportan el método propuesto.

Los resultados de esta investigación han sido aplicados en la construcción de software, por estudiantes informáticos y otros profesionales de la industria.

INTRODUCCIÓN

Para producir software de calidad es necesario el desarrollo de un proceso disciplinado [67]. Esta calidad no se logra con el esfuerzo puntual en alguna o algunas de las fases o etapas del ciclo de vida del software, es un trabajo consciente en la aplicación de técnicas ingenieriles, junto al establecimiento de métricas de calidad desde el inicio de su construcción. En la producción de software el esfuerzo fundamental está en el diseño, no en la codificación y la prueba, que solo ocupan el 15% del tiempo total de desarrollo [68].

La calidad es un tema complejo por la cantidad y diversidad de aristas que involucra, por lo que este trabajo se propone abordar un punto dentro de todos ellos, crítico en la mayoría de las aplicaciones que se desarrollan en la actualidad, *el diseño de la información almacenada.* La representación de los objetos tienen que ser completa, consistente, correcta, íntegra, fácilmente mantenible, flexible, fácil de usar y fiable.

En el diseño de la base de datos es importante la forma en que los objetos se guardan en los medios de almacenamiento. El diseño de la base de datos (BD) implica negociaciones entre usuarios y diseñadores de acuerdo a las necesidades de la aplicación y las capacidades del gestor [95].

El paradigma de la orientación a objetos (OO) ha provocado una revolución en los conceptos de la Informática, siendo el cambio más importante desde que surgieron los métodos estructurados [19], impactando en mayor escala en los lenguajes de programación de alto nivel y en las metodologías de análisis y diseño de sistemas informáticos. En el campo de las BD, en la actualidad se comercializan productos que soportan el modelo relacional, gestores de objetos y sistemas de gestión relacionales que extienden sus capacidades hacia el modelo orientado a objetos (MOO).

Esta diversidad hace difícil el diseño de la BD cuando se modela un sistema siguiendo el enfoque OO. El tema ha comenzado a ser tratado con mayor profundidad en los últimos años, aunque todavía quedan puntos en los que hay mucho que hacer, como se verá más adelante.

Todas las ventajas del enfoque OO giran alrededor de la posibilidad de aumentar la productividad, pero esto no es una tarea fácil. Depende del analista lograr un diseño que explote las posibilidades que brinda el enfoque porque depende de la manera que se mire y represente el mundo. Como plantea Timothy Budd: *"La programación OO es una nueva forma de pensar acerca de lo que significa computar, acerca de cómo pensamos estructurar la información dentro de un computador. (...) Aplicar con efectividad los nuevos recursos exige un cambio de percepción, es decir, una forma de pensar completamente nueva en cuanto a la resolución de problemas"* [37].

A diferencia de lo ocurrido con otros modelos, como el relacional y el deductivo, el MOO se ha caracterizado por una fuerte actividad experimental, antes de poseer una base teórica consolidada.

Los trabajos de formalización del MOO han seguido varios caminos [1][72] [103][104][105][135][160][170][179] que, aunque pueden tener puntos comunes, definen los principales conceptos de este paradigma con percepciones diferentes. Desde el punto de vista estructural o estático hay bastante consenso ya que por lo general están definidos los conceptos de clase, atributo, relaciones entre clases, herencia y agregación, pero no siempre se tienen en cuenta todas las restricciones estáticas que validan la inserción

apropiada de elementos. El comportamiento dinámico de los objetos es abordado insuficientemente ya que, aunque se incluyen casi siempre diagramas que reflejan estas características, no se construyen en función de su aporte al diseño de la BD, ni se define cómo utilizar la información que contienen en la etapa de implementación.

Las propuestas de metodologías por lo general comienzan con una definición de los principales conceptos que caracterizan al enfoque, pero salvo excepciones como en [63][135], se basan en definiciones informales. Muchos métodos se han introducido sin formalización [96][97].

Desde el punto de vista metodológico, sobre el diseño de la BD la mayoría de las metodologías no incluyen un modelo de persistencia (procesos que describen como modelar la persistencia de los objetos). Esta situación era lógica si se tiene en cuenta la poca fortaleza de los gestores orientados a objetos. Hay una tendencia a convertir clases a tablas [11][63][119][123][154], solo analizando, y no de forma completa, la parte estructural. Por lo general olvidan que los objetos en su definición incluyen el comportamiento, y que el modelo al que se transforma (el modelo relacional) centra su atención en los datos. Además las metodologías definen por lo general los pasos para construir una aplicación, pero no cómo diseñar la BD; ofrecen poca o ninguna recomendación para construir los modelos y diagramas que incluyen, y resultan insuficientes en la explicación de cómo transformar la información contenida en los diagramas hacia el medio de almacenamiento.

Algunos trabajos presentados en los últimos años [9][10][17][106][116][119][177], muestran una inclinación hacia la definición de una capa de interfaz entre la información almacenada y la aplicación que aisla a las clases del dominio de la forma en que se almacenó la información. A esta capa se le conoce como capa persistente o capa de objetos. En este trabajo se extiende el concepto de capa persistente, añadiendo las clases que describen las características de las relaciones entre los objetos asociadas al método de diseño propuesto, y que se incluye en la subcapa de especificación.

Con el desarrollo de metodologías para abordar el ciclo de vida del desarrollo de un sistema, comenzaron a surgir instrumentos automatizados que las respaldaban de forma tal que su utilización, con respecto a los métodos tradicionales de lápiz y papel, ha sido un acontecimiento en el mundo de la informática [85][144]. A estos instrumentos se les conoce como CASE (**Computer-Aided Software Engineering**).

A partir de esta situación, se ha definido como **problema de investigación**: "***El desarrollo de un método para el diseño de la base de datos a partir de un modelo orientado a objetos, que se base en un formalismo y que garantice:***

- ***la transformación de la semántica asociada a un objeto hacia el medio de almacenamiento que se utilice, y***
- ***conservar en la transformación las principales características del paradigma de la orientación a objetos."***

Por lo tanto se definió como **objeto de la investigación**:

" El diseño de la base de datos que es una de las actividades más importantes que se desarrollan durante la producción de software y que constituye parte del proceso de aplicación de técnicas y principios ingenieriles al software; y el paradigma de la orientación a objetos, como un enfoque que ha revolucionado la forma de construir aplicaciones"

Como consecuencia se plantea como **objetivo principal** de esta tesis:

"Proponer un método para el diseño de la base de datos partiendo del análisis de un problema bajo el paradigma de ***la orientación a objetos, que garantice una base de datos que refleje correctamente a los objetos del mundo real."***

A partir de este objetivo general se ha definido un conjunto de **objetivos específicos** que han caracterizado las diferentes etapas del desarrollo de este tesis. Ellos son:

1. ***Seleccionar como base un formalismo del MOO que se adecue a las características del modelo de persistencia a desarrollar.***
2. ***Definir los procedimientos y herramientas que se incluirán en el método DIBAO y determinarán cómo diseñar la BD, a partir del paradigma de la orientación a objetos.***
3. ***Definir la capa persistente que sirva de interfaz entre las clases del dominio y el medio de almacenamiento.***
4. ***Diseñar el esquema de un CASE que sirva como herramienta de ayuda al proceso de diseño de la base de datos y de la generación de su especificación, tomando como base los aspectos estáticos y dinámicos del método DIBAO.***

La **hipótesis de investigación** es: ***"Un modelo de persistencia que tenga en cuenta los comportamientos estáticos y dinámicos de los datos, conservando las principales capacidades del paradigma de la orientación a objetos, permite obtener un diseño de la base de datos que este en concordancia con los objetos del mundo real que modelan la aplicación a desarrollar"***.

Las **tareas** realizadas han sido:

1. ***Estudiar los modelos formales del paradigma de la orientación a objetos.***
2. ***Definir un modelo formal orientado a objetos sobre el que se sustente el modelo de persistencia.***
3. ***Estudiar el estado del arte y las tendencias de los gestores de bases de datos.***
4. ***Estudiar el estado actual de las investigaciones en el diseño de la base de datos a partir de un análisis y diseño usando el enfoque orientado a objetos.***
5. ***Proponer los procedimientos y herramientas definidos dentro del método DIBAO, teniendo en cuenta que la solución al problema se implemente usando:***
   - ***un Sistema de Gestión de Base de Datos Orientado a Objetos (SGBDO), un Sistema de Gestión de Base de Datos Relacional (SGBDR) o un Sistema de Gestión de Base de Datos Objeto-Relacional (SGBDOR);***
   - ***un lenguaje de programación orientado a objetos (LPOO) que permita conectarse con un gestor relacional u objeto-relacional, y***
   - ***un LPOO y sus posibilidades para el almacenamiento en fichero.***
6. ***Definir la capa persistente con las subcapas de especificación e interfaz.***
7. ***Determinar aspectos que puedan garantizar la calidad de la base de datos diseñada.***

8. ***Definir el esquema del CASE para el diseño de la base de datos y la generación de su especificación.***

9. ***Obtener productos que implementen el método propuesto.***

10. ***Validar el modelo y el CASE obtenido.***

Los **métodos de trabajo científico** utilizados para acometer estas tareas fueron:

- Métodos empíricos: de **observación,** empleado en el estudio de las características de los proyectos de software que se desarrollan y de las tendencias seguidas por los diseñadores y programadores en el diseño de la BD y su implementación hacia un medio de almacenamiento persistente; de **medición,** se utilizó para comparar las distintas propuestas de métodos para el diseño de la BD a partir de un MOO, sobre la base de determinadas magnitudes que expresan el grado de completitud y calidad de sus modelos de persistencia; y de **experimentación,** aplicado en la aplicación completa o parcial del método de diseño propuesto en casos de estudio, que se corresponden con aplicaciones reales, para demostrar su validez y detectar aspectos en los cuales se debía profundizar más.

- Métodos lógicos: el **hipotético-deductivo**, se utilizó para la determinación de la hipótesis principal de la investigación y en la definición de los caminos a seguir en la investigación a partir de los resultados parciales alcanzados con la experimentación; al **analítico-sintético**, sirvió de base al descomponer el problema de investigación en partes, profundizando en cada una de ellas por separado e integrándolos en el método de diseño que se propone; el **inductivo-deductivo,** se empleó durante el estudio bibliográfico realizado para descubrir las tendencias seguidas en el diseño de la base de datos y proponer un modelo que resuelva las deficiencias detectadas; y el de la **comparación-clasificación,** fue aplicado en el análisis de los resultados obtenidos en los proyectos que han aplicado el modelo de persistencia propuesto.

- Métodos **histórico** y **dialéctico** se utilizaron en el estudio crítico de otros métodos para el diseño de la BD y de formalismos del MOO y en la determinación de los aspectos positivos de las propuestas valoradas para su utilización como referencia en el trabajo desarrollado.

El método DIBAO está compuesto de un modelo de persistencia y una capa persistente de clases. El modelo de persistencia incluye los pasos que deben realizarse para diseñar la base de datos a partir del análisis de un problema utilizando el paradigma de la orientación a objetos. Se describe qué hacer si la persistencia es hacia ficheros, una base de datos de objetos, o un gestor u objeto/relacional; y cómo interpretar los diagramas y especificaciones textuales en función de este diseño. La capa persistente define las clases que recogen las nuevas características relacionadas con las relaciones entre clases, y otras que encapsulan el trabajo con el medio de almacenamiento.

La **novedad científica** de esta tesis está dada por:

1. ***La sistematización de los pasos para el diseño de la BD, a partir del análisis de un problema bajo el paradigma de la orientación a objetos, teniendo en cuenta que la implementación de la persistencia puedes ser hacia ficheros, un gestor de objetos, un gestor relacional o un gestor objeto/relacional.***

2. ***La definición de recomendaciones que permitan obtener: la información necesaria para la descripción de los comportamientos estático y dinámico de los objetos, y***

la interpretación que se deriva de los diagramas y especificaciones textuales, en función del diseño de la BD.

3. ***La definición de una capa persistente, con las subcapas de especificación y de interfaz, que oculte, a las clases del dominio del problema, la forma de almacenamiento de los objetos y que recoja las características estáticas y dinámicas identificadas como parte de los procesos incluidos en el modelo de persistencia.***
4. ***La concepción del CASE que tenga en cuenta los aspectos estáticos y dinámicos en el diseño de la BD, de acuerdo al método propuesto.***

Su **valor práctico** se expresa:

En un mundo en el que: abundan aplicaciones que requieren almacenar grandes volúmenes de información, existen en el mercado fuertes sistemas de gestión de base de datos soportados sobre el modelo relacional y hay nuevos productos que se basan en otros modelos, y el uso cada vez con mayor frecuencia del enfoque orientado a objetos en la solución de los problemas; es de gran utilidad poseer herramientas y procedimientos que permitan diseñar la base de datos con precisión y corrección.

Las empresas cubanas han comenzado a moverse hacia el enfoque OO, por lo que se ha producido un impacto profundo en la forma de construir aplicaciones. Debido a esto es importante que conozcan cómo diseñar la base de datos a partir de un análisis realizado utilizando una metodología orientada a objetos.

El método DIBAO, desarrollado por la autora de esta tesis, forma parte de las versiones de la metodología ADOOSI (Análisis y Diseño Orientado a Objetos de Sistemas Informáticos) desde 1997 y en cada una de ellas se han introducido los resultados parciales de la investigación. La versión actual de ADOOSI (versión 5.0) [7] contiene todo lo especificado en esta tesis.

El método DIBAO, como parte de ADOOSI, se imparte a estudiantes de pregrado y postgrado. Los resultados de esta tesis han sido utilizados por estudiantes de cuarto y quinto año en el desarrollo de proyectos y por otras empresa del país. Actualmente el 49 % de los proyectos de 4to año de la carrera de Ingeniería Informática utilizan la tecnología OO y de ellos el 97% utilizan el método de diseño propuesto ya que necesitan almacenar información.

La posibilidad de poseer una parte del CASE permite aumentar la productividad, lo que se traduce en disminución del tiempo de realización y por consiguiente disminuye el costo del proyecto.

Tanto el modelo de persistencia como el CASE, abordan una tecnología de punta.

Los principales resultados teóricos y prácticos que se han obtenido han sido presentados en eventos y publicaciones [5][7][81][82][83][84][85][86][87][88][89][90][91][92][93][94].

Capítulo 1
El Modelo Orientado a Objetos y las Bases de Datos

CAPÍTULO 1 EL MODELO ORIENTADO A OBJETO Y LAS BASES DE DATOS

Frecuentemente los datos se organizan y mantienen utilizando la forma de organización conocida “sistemas orientados a bases de datos”, en los que hay una débil interdependencia con los programas de aplicación y la organización física de los datos [50].

“Una base de datos (BD) es una colección de datos dinámicos larga, persistente e integrada, que brinda algunas operaciones para describir, establecer, manipular y accesar estos datos ” [63].

Todos los gestores de BD soportan un modelo de datos. En este capítulo se hace referencia al estado del arte de la formalización del modelo orientado a objetos (MOO), haciendo hincapié en el modelo O^3 (Orientado a Objetos y Ontológico) por ser el escogido de base para esta tesis. Se describe además la situación del mercado en cuanto a los sistemas de gestión de base de datos, para dar paso a la situación actual con respecto a las propuestas metodológicas sobre el diseño de la base de datos partiendo del uso del paradigma de la orientación a objetos.

1.1 Formalización del Modelo Orientado a Objetos.

Un modelo de datos es un formalismo matemático , es una herramienta que permite realizar una interpretación de un dominio de aplicación con determinado nivel de abstracción, de forma que se reflejen los elementos u objetos del dominio dado, así como las formas en que se interrelacionan [167].

> Cuando los objetos del mundo real se intentan llevar a un medio de implementación, no siempre se logra una correspondencia total con la realidad.
>
> El modelo relacional (MR) posee una fundamentación desde el punto de vista teórico que se basa en la matemática, específicamente en los conceptos del álgebra y cálculo relacional y la teoría de predicados de primer orden [109].

El MOO es un modelo que permite describir los datos, sus estructuras y las operaciones válidas que se pueden realizar, pero apoyándose en los conceptos empíricos del paradigma de la orientación a objetos (objeto, clase, herencia, encapsulamiento, identificación de objetos, comunicación a través de mensajes y polimorfismo) [76]. A diferencia del MR, los productos comerciales basados, en el MOO, han antecedido a la fundamentación teórica, aunque se está imponiendo cuando se trabaja con problemas que manejan objetos complejos.

1.1.1 Situación actual.

En [96] se plantea que hay poca literatura dedicada a la formalización del MOO, aunque se reconoce la necesidad de formalizar las técnicas de modelación. Desafortunadamente los fundamentos teóricos del MOO comenzaron después que se desarrollaron los sistemas de gestión bases de datos de objetos (SGBDO). Es por eso que en ocasiones se encuentra definiciones diferentes de un mismo concepto [72][97], entre diferentes modelos formales. La mayoría de los formalismos de este modelo están asociados al desarrollo de un lenguaje de especificación.

Los trabajos del grupo IS-CORE (**Information Systems- COrrectness and Reusability**) del ISTL (Instituto Superior Técnico de Lisboa) [135] formalizan a un objeto como una 4-tupla

(Conjunto de atributos, Conjunto de eventos, Conjunto de ciclos de vida o secuencias admisibles de eventos, Función de observación). En [135] se señala como el problema principal de esta propuesta, es que trata de separar la modelización de la parte estructural (datos) de los objetos, de la modelación de la parte dinámica (procesos).

Asocian a los objetos las fórmulas [160]: **Valuation** (indica los efectos de los eventos sobre los atributos), **Safety** (condiciones a cumplir para que se permita el evento) y **Liveness (**condiciones bajo las cuales un evento es obligado que ocurra). Define además los operadores de agregación y herencia. Esta propuesta ha evolucionado, y en la actualidad cuentan con un lenguaje de especificación (Oblog), que permite la generación de código [20] y se usa la notación de UML (Unified Modeling Language), con algunas extensiones menores para aumentar la precisión [127].

Otra formalización del modelo OO desarrolló un lenguaje de especificación semi-formal conocido como Troll [103][105]. Posteriormente se realizaron trabajos que unieron la metodología OMT (**Object Modeling Technology** - una de las más usada durante gran parte de la década de los '90) con Troll [104].

Los fundamentos de Oblog y Troll están en la teoría de las categorías.

Basado en grafos, en [170] se describe un modelo de datos orientado a objetos (OO), pero relacionado solo con los datos, sus propiedades y sus relaciones con otras entidades, muy cercano al MR.

En [72] se describe un modelo que trata de unificar las tendencias de varios modelos existentes. Este modelo se basa en la teoría de las categorías y los tipos de objetos son nodos de un grafo. Solo define conceptos asociados con la estructura.

En [1] se describe un modelo de base de datos orientado a objetos genérico en el que formalmente están definidos los conceptos de objeto, jerarquía de clase, clase, método y esquema e instancias; pero faltan conceptos de operadores que permiten ampliar el concepto de clase y otros relativos al comportamiento dinámico.

En [110] se describe la transformación del modelo conceptual a una especificación abstracta basada en la **Frame logic** (**F-Logic**). La F-Logic tiene en cuenta los conceptos orientados a objetos y deductivos, pero al modelo le faltan operadores para construir objetos complejos y fórmulas dinámicas. Esto se debe a que se parte de una definición del modelo conceptual en la que se incluye estructura (tipos de objetos, atributos, rol de estos atributos en su relación con los objetos y relaciones de generalización/especialización) y restricciones de integridad (relacionadas con la cardinalidad de las relaciones y el dominio).

Todas estas formalizaciones hacen hincapié en la estructura de los objetos.

La tesis para optar por el grado de Doctor de Roef Johannes Wieringa [179], ofrece una formalización del MOO bastante completa en la definición de conceptos estáticos y dinámicos. El autor reconoce como un problema de su modelo que tiene muchos detalles que lo hacen muy grande, con más elementos que los que se usan. Pero también es cierto que se hacen menos simplificaciones que en otros trabajos porque es más formal. Trabajos posteriores profundizaron en el concepto de herencia [180].

Los trabajos desarrollados por Y. Wand, a finales de los '80 [135], brindan una aproximación ontológica del modelo. La idea es que el MOO permite tener una visión natural del mundo, siendo los objetos la estructura básica de la modelación, por lo tanto, el término ontológico se refiere al estudio de los objetos tal como son.

Los trabajos de la Universidad Politécnica de Madrid (UPM), comenzaron tomando como base la propuesta del grupo IS-CORE. En sus primeras versiones eran aproximaciones algebraicas que hacían poco hincapié en la parte dinámica [145], pero en la actualidad es una de las propuestas más completas.

El modelo O^3 (*orientado a objetos y ontológico)* [135] se basa en la propuesta de Y. Wand, que hace una distinción explícita entre objetos y propiedades y trata la dinámica de un sistema de objetos a través del concepto de Ley (conjunto de estados permitidos o prohibidos, que son propiedad de los objetos). Este modelo:

- añade a la parte estática el concepto de restricciones estáticas, que son leyes que fijan las combinaciones de valores de atributos aceptados como válidas para cualquier estado,
- define restricciones de integridad dinámica, que son leyes que establecen las relaciones entre los valores de los atributos en diferentes estados,
- incluye precondiciones como leyes que rigen la potencial ejecución de eventos,
- y define el disparo como un mecanismo adicional de interacción entre objetos, que se implementan como leyes objetuales asociadas a objetos que pueden actuar como agentes, activando el evento correspondiente cuando la condición expresada en la ley se satisfaga.

Esta formalización del MOO se ha basado en la programación lógica concurrente [117], en especificaciones algebraicas [168]], en la lógica clausal dinámica [43], en las redes de Petri [157], y en la **transaction frame logic** [42][142], que implícitamente está presente en la última versión de su modelo formal [118].

Los conceptos definidos dentro de O^3 son la base de la propuesta de la metodología desarrollada en la UPM (OO-Method), descrita en el epígrafe 1.3.2.2.

Al definir cuál formalismo utilizar, se tomaron en cuenta los principios definidos en [96]. El modelo O^3 los cumple todos, pero en la decisión incidieron fundamentalmente dos: el principio del objetivo principal y el de la ortogonalidad; que se refieren a la correspondencia entre los objetivos del formalismo y del uso que se le dará y que con un conjunto mínimo de conceptos es posible definir las bases de la formalización, respectivamente. Esta propuesta es la tomada como base en este trabajo debido a que el modelo de persistencia debe permitir obtener tanto características estáticas como dinámicas y las bases escogidas para la formalización del modelo O^3 son más adecuadas para la modelación del comportamiento dinámico; lo que permite definir, a diferencia de otros formalismos, un grupo más completo de fórmulas dinámicas que expresen las reglas que rigen el comportamiento de los objetos. Además, es un modelo de fácil comprensión, por los términos en que se define, y logra con pocos conceptos una formalización más completa de una clase a través del uso de operadores.

1.1.2 Modelo formal.

Las definiciones del modelo O^3 que se presentan, fueron tomadas de [118][135][137] [158] y se basan en un marco formal definido sobre una variable de la lógica dinámica que permite representar los operadores de obligación, prohibición y permiso usados en la lógica deóntica [158]. La principal definición sintáctica es el concepto de clase.

- Clase: Una clase C se define formalmente como: *"C={E,new,destroy,r,A,T,α}",* donde:

  A- Conjunto de atributos que toman valor en el dominio a ellos asociado.

E- Conjunto de eventos que condicionan el estado del objeto, distinguiéndose los eventos **new** y **destroy,** que representan los eventos de creación y destrucción del objeto. Para cada evento se identifica una función rango (r), que caracteriza el dominio de sus argumentos.

T- Conjunto de trazas o ciclos de vida que, comenzando por el evento de creación, definen la secuencia de eventos que conforman la vida del objeto.

α- *F*unción de observación que permite expresar el valor de los atributos que conforman la estructura de un objeto en un instante dado, en función del comportamiento. Relaciona trazas con atributos, obteniéndose pares (atributo, valor) como una función de la secuencia de eventos que le han sucedido.

- Agregación: La agregación es por lo general la forma de representar que un objeto tiene como componentes a otros objetos. Desde el punto de vista formal se puede expresar como: *"Dadas dos clases C1, C2; con C1={E1,new1,destroy1,r1,A1,T1,α1} y C2={E2,new2,destroy2,r2,A2,T2,α2}, se define su agregación como una clase C={E,new,destroy,r,A,T,α}."* Donde:

  E- Conjunto de eventos compartidos entre ambas clases y que son propios de la clase compleja agregada, más nuevos eventos definidos para la agregación.

  r- Al definirse sobre E, se toman en cuenta las r1 y r2 de C1 y C2 relacionadas con los eventos compartidos.

  A- Conjunto de atributos formado por los atributos propios de la agregación y los atributos constantes que identifican a las clases C1y C2.

  α- la función de observación no guarda relación con la de las clases componentes, es una propiedad emergente de la clase agregada.

  T- conjunto de trazas que están relacionadas con los eventos de la clase agregada.

  Las formas en que se puede dar esta agregación están relacionadas con la dimensión Estática/Dinámica, que se retoma en el epígrafe 2.2.4.

- Especialización/generalización: Estos operadores permiten definir el concepto de herencia, que es una de las características más importantes del paradigma de la orientación a objetos. Formalmente:

  "Dadas dos clases C1, C2; con C1={E1,new1,destroy1,r1,A1,T1,α1} y C2={E2,new2,destroy2,r2,A2,T2,α2}, se define su generalización como la clase C={E,new,destroy,r,A,T,α}". En la clase C se definen los elementos de intersección de los atributos, eventos, trazas, rangos y función de observación de las clases que se generalizan.

  "Dada una clase C1, con C1={E1,new1,destroy1,r1,A1,T1,α1} se define la especialización de C1 como la clase C, C={E,new,destroy,r,A,T,α}". En la clase C1 se definen todos los elementos de C y otros propios de ella.

La semántica de una clase se define en términos de una estructura de Kripke (W, τ,ρ) [118], donde W es el conjunto de todos los mundos que un objeto puede alcanzar (los mundos son estructuras sobre las que se interpretan las fórmulas dinámicas), la función τ asigna a una fórmula en la lógica de estado (la lógica de predicados de primer orden) el conjunto de

mundos en los cuales se satisface, y la función ρ asigna a cada paso una relación binaria entre mundos.

Cada objeto de una clase va a encapsular su estado y las reglas que rigen su comportamiento, por lo que puede ser visto estáticamente a través de sus atributos y, desde el punto de vista dinámico, a través de las fórmulas de: evaluación, derivación, precondiciones, restricciones de integridad y disparadores (las dos primeras cambian los valores de los atributos y el resto son reglas a cumplirse por los objetos). Los conceptos de estas fórmulas se describen en el epígrafe 1.3.1.2, y en el 2.3.2 se especifica cómo obtenerlas en el proceso de diseño de la base de datos.

1.1.3 ODMG.

El grupo ODMG (**Object Database Management Group**) surgió en 1991 con el objetivo de dotar a los clientes de los SGBDO de un conjunto de normas que les permitieran escribir aplicaciones portables [44], lo que ha dado un impulso en su desarrollo [27]. Este grupo considera que el éxito de los Sistemas de Gestión de Base de Datos Relacionales (SGBDR) no solo está en las potencialidades del modelo relacional (MR), sino además en poseer un lenguaje de consulta estándar que permite un alto grado de portabilidad y simplifica el aprendizaje de un nuevo SGBDR [25][44]. En 1993 se divulgó ODMG-93 como el estándar industrial para el almacenamiento persistente de objetos.

En 1997 aparece la versión 2.0 de ODMG que es usada como referencia por algunos gestores, por ejemplo Objectivity/DB 5.2 [129], uno de los gestores de objetos más utilizados últimamente [99]. ODMG 2.0 está compuesto por: un modelo de objetos, un lenguaje de definición de objetos (**Object Definition Language** - ODL), un lenguaje de consulta (**Object Query Language** - OQL) y un lenguaje de manipulación de objetos (**Object Manipulation Language** – OML) para los lenguajes Java, Smalltalk y C++.

En enero del 2000 se publicó la versión 3.0 [45]. Esta nueva versión hace hincapié en el diseño, desarrollo e implementación en una base de datos de objetos (BDO) y en productos donde se convierten los objetos al MR. Sobre esté último aspecto es importante definir estándares porque ya los SGBDO comienzan a incluir conversiones de clases a tablas, por ejemplo, Poet [59][143].

ANSI (**American National Standards Institute**) lleva varios años trabajando en la versión SQL3 (**Structured Query Language**) o SQL:1999 [60], que tiene en cuenta el concepto de objeto. Ya se han obtenido resultados que se divulgan [61]. ODMG 3.0 posee un lenguaje que genera un esquema que puede ser trasladado al lenguaje de definición de datos de SQL:1999 [131].

1.2 Base de Datos.

El modelo relacional centra su atención en dominios simples. Los gestores relacionales implementan el concepto de tipo de datos y no de dominio, aunque algunos, como SQL, definen predicados para chequear restricciones sobre columnas [52].

El principal problema del MR radica en que a veces se hace difícil, y en ocasiones imposible, capturar la complejidad semántica del mundo real [39][153]. Además, los detalles de implementación dependen de los requerimientos iniciales lo que hace difícil las modificaciones, es decir, la evolución del esquema. Las principales soluciones que se han

adoptado han sido ampliar el modelo relacional de forma apropiada o desecharlo por completo, sustituyéndolo por algo nuevo. El MOO sigue la segunda tendencia .

Tomando como premisas que: la tecnología OO es un nueva filosofía que permite representar más semántica que las tecnologías tradicionales, que los gestores de BD incluyen técnicas que han demostrado su eficacia en el almacenamiento y manipulación de información, y que los usuarios necesitan incorporar persistencia a los lenguajes orientados a objetos; se han desarrollado varios productos que responden a diferentes alternativas para unir las capacidades de las BD y los conceptos de la OO.

Los SGBDO surgen de la convergencia entre la tecnología de las BD y el paradigma de la OO, con el fin de atender a requisitos para los cuales los productos relacionales no dan una respuesta satisfactoria. Un SGBDO integra las capacidades de una BD con las capacidades de los LPOO [26][43].

La fortaleza en la fundamentación del MR y su gran utilización, hacen imposible desecharlo por completo [86]. Entre finales de los ’80 y principio de los ’90 se desarrollan la mayoría de los gestores OO [47], dentro de los que podrían señalarse a GemStone, Poet, Versant y Objectivity [31][54][80][107][128][143][166][173]. La tendencia actual es a la evolución de los productos existentes, surgen pocos nuevos.

Para entender por qué es diferente diseñar la base de datos para el MR y el MOO, en la tabla 1 se muestra una comparación que refleja los principales puntos que los diferencian [15][18][21][46][62][76][114][122].

Tabla 1 Comparación entre MR y MOO.

MR	**MOO**
Se representan los datos normalizados.	Se representan los objetos con datos y comportamiento
Los atributos solo pueden tomar valores atómicos, por lo tanto un objeto del mundo real se representa en muchas tablas normalizadas. Para recuperar objetos complejos, hay que hacer uniones de tablas y esto es costoso en tiempo.	Los objetos están almacenados como un todo coherente por lo que la recuperación de un objeto es una operación única, aunque puede ser grande.
Las operaciones que se pueden hacer sobre las relaciones se restringen a la inserción, modificación, eliminación y recuperación de tuplas.	Cada método responde a un comportamiento del objeto en el mundo real que es de interés para la aplicación abstraer.
Brindan pocas posibilidades para expandir o modificar el esquema.	Están implementados para permitir añadir más semántica.
Los objetos se identifican por sus atributos. Las propiedades de unicidad y constancia no siempre están presentes en el mundo real por lo que a veces hay que introducir identificadores artificiales como llaves.	Todos los objetos tienen un identificador único que no tiene nada que ver con los atributos, sino que es definido automáticamente por el gestor o lenguaje,
Algunos sistemas ofrecen facilidades para especificar las restricciones de integridad,	Las restricciones de integridad son implementadas como métodos de las

MR	MOO
pero aún en estos casos es responsabilidad del programador especificar los procedimientos para garantizarlos.	clases. Es responsabilidad de los programadores su definición y programación.

El MR y el SQL son aceptados como estándares, pero tienen problemas para trabajar con datos complejos, y en la actualidad los sistemas de información que se construyen son más complejos y especializados [29]. Las tecnologías relacional y objeto son complementarias y pueden trabajar juntas en beneficio de los usuarios [58].

No existe una definición estándar de lo que es un SGBD objeto/relacional o híbrido (SGBDOR), pero el principio es que se toma del modelo relacional su seguridad, integridad y fiabilidad; y del MOO su capacidad de almacenar objetos complejos [41] [51][56][77][108][112][134][171]. Según Linthincum [120], el trabajo sería descomponer objetos en tablas relacionales para el almacenamiento.

Optar por el modelo híbrido implica una transición suave hacia el MOO ya que los clientes pueden conservar las inversiones realizadas con BDR y el aprendizaje es más fácil [92]. Asumir esta opción implica conocer, que con respecto al uso de gestores de objetos, el modelo híbrido tiene limitantes ya que el objeto no se ve como un todo, aunque se puede almacenar tanto su estructura como su comportamiento, no es en una unidad conceptual. Además no apoya redes de objetos complejos [26]. Es decir no se implementan algunos conceptos de la tecnología de objetos (clase, herencia de comportamiento, comunicación a través de mensajes), aunque sí otras como la posibilidad de que un atributo referencie a un objeto.

Gestores como Oracle versión 8, Informix Universal Server, IBM DB2 Universal Database, Illustra (se integró a Informix desde 1996), soportan este modelo [36][38][65][66][69][70][71][78][111][113][134][163][164][175] [178].

1.3 Estado del diseño de la base de datos a partir de un Modelo Orientado a Objetos.

Llevar los conceptos del mundo real a una BD física, es un proceso complejo que puede hacerse por intuición o aplicando técnicas ingenieriles . Ambos caminos son posibles, pero no por ser más rápido el primero, se logra una BD consistente y correcta.

El estado del arte en las BD, propicia que la mayoría de las aplicaciones que se desarrollan, se basen en gestores que implementan el MR. Aunque los métodos de diseño pueden tener aspectos diferentes, de manera general se basan en el uso del diagrama entidad-relación y en la ejecución del proceso de normalización.

En el proceso de normalización se aplica a las relaciones, restricciones impuestas por las formas normales (FN). La 1ra FN elimina atributos multivalores, compuestos y sus combinaciones. En la 2da FN los atributos no llaves deben tener una dependencia funcionalmente completa con respecto a la llave. En el caso de la 3ra FN, se exige que no existan atributos no llaves que dependan de otro no llave. Otras formas normales se han definido como consecuencia de otras anomalías.

Sobre el diseño de la BD a partir de un MOO, hay pocos estándares en cuanto a herramientas y procedimientos, aunque hay inclinación hacia la conversión de las clases a

tablas como se verá en el epígrafe 1.3.2.2, y el uso del diagrama de clases como homólogo del Diagrama Entidad-Relación para representar el modelo conceptual [62].

Diseñar la BD a partir de un MOO no puede hacerse aplicando los métodos tradicionales porque no se trabaja con los mismos conceptos, por lo tanto, se requieren nuevos métodos y herramientas de desarrollo [89]. Además, los métodos y herramientas, utilizados para el MR, presentan limitaciones ya que se refieren solo a los datos y sus relaciones, sobre los que no tienen en cuenta todas las restricciones a ellos asociados [87].

El diseño orientado a objetos tiene como objetivo refinar, extender y reorganizar las clases resultantes de la fases de análisis y añadir nuevas si es necesario. Se caracteriza por estar relacionado con la solución del sistema de información y generar la especificación rigurosa de todos los aspectos de las clases.

Sobre la situación actual en cuanto al diseño de la BD a partir del MOO, tratan los epígrafes siguientes, pero antes se definen conceptos importantes que sirven de referencia para el análisis de las metodologías que se describen.

1.3.1 Modelo de persistencia.

La persistencia es la capacidad de un objeto para existir fuera de un programa, proceso, función o hilo de control [57]; de manera que se conserva su estado y su comportamiento. Para garantizar la persistencia los SGBDR disponen de cuatro operaciones: creación, recuperación, actualización (se refiere a la modificación) y eliminación (CRAE). Un modelo es un conjunto de reglas, conceptos y convenciones que permiten describir “algo”. Cuando se le añade el término de “persistente”, se refiere a los procesos definidos para modelar los aspectos persistentes de una aplicación orientada a objetos [18].

Según Jesse Liberty [119], todo modelo o mecanismo de persistencia debe cumplir las siguientes reglas:

- Cada elemento de un objeto (atributo, asociación, jerarquía) tiene que ser proyectado en un elemento de almacenamiento persistente.
- El código fuente de una aplicación incluye las operaciones CRAE. Este código tiene que ser escrito y probado para un mecanismo persistente en particular.
- El formato de almacenamiento persistente tiene que ser mantenido a la par con la definición de los objetos que es necesario almacenar. Durante el desarrollo de un proceso, los objetos pueden cambiar (sus atributos, relaciones, etcétera.) y la definición persistente tiene que corresponderse con estos cambios.

Un diseño de la BD a partir de un análisis OO implicará tomar la decisión de la forma en que se almacenarán los datos, lo que implica en estos momentos decidir por la persistencia a un fichero, a un SGBDO o en BDR con o sin capacidades de la orientación a objetos. La transformación de los objetos tiene que tomar en cuenta que un objeto no solo engloba propiedades sino que también incluye comportamiento.

1.3.1.1 Tendencias de los modelos de persistencia.

Las tendencias que siguen los modelos de persistencia se pueden agrupar en dos variantes [31]:

- Modelo normalizado: los objetos se descomponen en componentes atómicos que se almacenan en archivos diferentes donde la relación entre varios componentes se mantienen por medio de los identificadores de objeto.
- Modelo directo: los objetos se almacenan en la misma forma en que se definen en el esquema conceptual, es decir, la unidad de almacenamiento es la misma que la unidad semántica.

El modelo directo es desventajoso cuando los objetos tiene campos de grandes dimensiones ya que el acceso a los atributos se vuelve costoso, sin embargo, la transferencia de un objeto complejo resulta ser un proceso eficiente al no requerir operaciones de acoplamiento. En el caso del modelo normalizado las desventajas son las mismas que las derivadas de la comparación entre los MR y MOO, descritas en el epígrafe 1.2.

1.3.1.2 Vistas estática y dinámica.

El análisis de cualquier problema requiere de un estudio global en el que se identifique no solo la información a manipular, sino también restricciones sobre esa información, relaciones que se establecen y procedimientos a ejecutarse que actúan sobre ella.

> En el MOO cuando se habla de vista o perspectiva estática se refiere a los conceptos de clase, atributo, relaciones, restricciones de integridad estática, herencia y agregación; por lo que es más rica que el MR. Como vista o perspectiva dinámica se incluyen las definiciones de mensaje, evento, estado, transición, petición o envío de mensaje y restricciones de integridad [49][136].
>
> Los objetos vistos desde una perspectiva estática muestran el conjunto de propiedades (atributos) que describen estructuralmente al objeto, de manera que los valores asociados a cada propiedad del objeto caracterizan el estado en un instante.
>
> Las restricciones estáticas buscan hacer imposible la inserción inapropiada de elementos [115] . Por ejemplo, para el caso de que sea un atributo de tipo numérico se puede asociar una regla de negocio de acuerdo su significado (el presupuesto de una empresa no puede ser mayor que la suma del presupuesto de las áreas y sólo puede tomar valores en el intervalo [0,1000000]). Es decir, estas restricciones están fuertemente relacionadas con el dominio de los atributos de los objetos.
>
> Cooling plantea que los objetos en la modelación dinámica responden a cuatro preguntas: ¿qué interacciones tienen lugar entre los objetos?, ¿por qué ocurren?, ¿cuándo ocurren? y ¿qué efectos generan en los objetos? [49].

A cada clase se le pueden asociar fórmulas de la lógica dinámica [118]:

- Evaluaciones: Caracteriza explícitamente parte del estado de un objeto antes y después de la ocurrencia de una determinada acción.

  $\varphi \rightarrow [a]\phi$ Produce cambios de estado.

- Derivaciones: Son fórmulas de la forma $\phi \rightarrow \phi'$ que permiten definir atributos derivados (ϕ) en términos de una condición de derivación declarada en ϕ.

  [a] ($\phi \rightarrow \phi'$) Expresa como se obtiene el valor de un atributo derivado. Se debe satisfacer en cada estado del objeto. No produce cambio de estado.

- Precondiciones: Prohibiciones para la ocurrencia de acciones.

  $\neg\phi \rightarrow [a]false$ Si $\neg\phi$ se satisface, entonces la ocurrencia de a está prohibida, es decir, ϕ es una fórmula que tiene que ser válida para que pueda ejecutarse la acción.

- Disparo: Se disparan automáticamente, no como consecuencia directa de acciones de los usuarios.

  φ [$\neg\phi$]false La acción se activa cuando la condición que plantea la fórmula ϕ se satisface.

- Restricciones de integridad: Son fórmulas que deben ser satisfechas por lo que se evalúan en el modelo cuando se produce un evento que provoca un cambio de estado. Se clasifican en estáticas y dinámicas. Las estáticas siempre son aplicables, en cambio las dinámicas dependen del estado actual.

  Donde ϕ, ϕ' y φ son fórmulas bien formadas de la lógica de predicado de primer orden y $a \in A$, A es el conjunto de acciones.

El uso de disparadores es muy útil [18][171] en el mantenimiento de restricciones de integridad (cuando no es posible con las restricciones declarativas definidas en el momento de crear un elemento), en la auditoría de la información (registrando los cambios realizados y la identidad de que los llevó a cabo), y en el aviso automático de se debe ejecutar una acción como consecuencia de un cambio en un elemento.

> El término restricciones dinámicas se refiere a la exigencia de la unicidad en el valor que toma(n) el (los) atributo(s) que se defina(n) como llave, en el caso de existir este concepto, o a las implicaciones que puede tener la inserción o eliminación de un elemento cuando hay jerarquía de clases o se trabaja con objetos complejos.
>
> Las restricciones dinámicas también se refieren a las condiciones que deben cumplirse para disparar una operación o que deben asegurarse antes o después de una operación para que ocurra correctamente (precondiciones y postcondiciones).

Para captar está semántica, las metodologías OO incluyen varios diagramas. En el caso de la perspectiva estática por lo general tiene nombres cercanos a los de "diagrama de clases" o "modelo de objeto" [140][154]. Para la perspectiva dinámica son usados los diagramas de transición de estado y otros que recogen la interacción entre los objetos, de ellos se podrían obtener varias de las fórmulas (por ejemplo, los disparadores y precondiciones del diagrama de transición de estado) pero no se explotan suficientemente en el proceso de diseño de la BD.

1.3.2 Tratamiento de la persistencia en las metodologías de análisis y diseño.

Gran parte de las metodologías desarrolladas que siguen el enfoque OO, eluden el tema del diseño de la BD. Incluso hay algunas que reconocen que no abordan este problema (Fusion) [126]. Esta situación es lógica si tomamos en cuenta el estado de los SGBDO y que el desarrollo de métodos para un nuevo enfoque va precedido del desarrollo de lenguajes que lo soporten. Las pocas metodologías que hacen referencia a este punto, por lo general han optado por dar solución a este problema planteando un conjunto de reglas que permiten convertir los objetos, definidos a partir del análisis del problema, en tablas tal como se plantea en el modelo relacional. A continuación se hará referencia a los métodos de diseño de la BD incluidos en algunas de ellas.

Los métodos que se abordan por lo general son específicos para aplicaciones de gestión y han tomado en cuenta la situación del mercado en el cual hay una gran cantidad de SGBDR ampliamente difundidos y pocos SGBDO. Algunos métodos plantean que la definición de esta base se concreta en decidir cuáles clases se desea que sean persistentes [4][34] y utilizar lo que brinda el lenguaje para salvar y recuperar valores o después de un análisis y diseño enseñan a obtener las tablas que se requieren para el modelo relacional, realizando un proceso de serialización (**Object Modeling Technique –** OMT [154] y **OO-Method** [135][136][137][138][139][140][141].

1.3.2.1 Serialización del Modelo Orientado a Objetos.

Al proceso de tomar un objeto y convertir su información en una forma que pueda ser almacenada y transportada, se le conoce como serialización [119], aunque sugiere un concepto más general, es muy cercano a lo definido como modelo normalizado. Para estos casos la idea es convertir un objeto en una o varias entradas a una o varias tablas.

Tomando como base una metodología propia OO desarrollada en nuestro país (ADOOSI - Análisis y Diseño Orientado a Objetos de Sistemas Informáticos), en el epígrafe 2.4 se explica cómo el método DIBAO propone realizar este proceso.

1.3.2.2 Principales propuestas.

OMT [154] tiene puntos de contacto con metodologías tradicionales (usa el Diagrama de Flujo de Datos). En el diseño de la BD centra su atención en los datos.

Unified Modeling Language (UML) [146] es aceptado como estándar por ODMG en noviembre de 1997. A partir de ese momento un conjunto de investigadores han desarrollado metodologías propias para el desarrollo de proyectos OO basándose en UML. Tomar como base las herramientas que ofrece UML sin una guía, es una tarea difícil cuando se pretende realizar un software.

UML es un lenguaje gráfico que brinda un vocabulario y reglas para visualizar, especificar, construir y documentar un sistema [100]. UML es un lenguaje de modelación no un método porque no incluye los pasos a seguir para desarrollar un software usando el enfoque OO [14][67][150]. Es un modelo bien formado en el que se explica semánticamente el significado de sus componentes [35], pero sin basarse en un modelo formal. En [165] se presenta un trabajo que aborda la formalización de su diagrama de clases (DC).

En el DC de UML [116][155] se representan atributos, relaciones entre objetos con sus cardinalidades máxima y mínima, herencias y agregaciones. Usando una sintaxis particular se pueden indicar de los atributos: nivel de protección, multiplicidad, posibilidades de cambiar su valor una vez asignado, valor por defecto y tipo de dato. En UML [35] se recomienda usar disparadores y procedimientos almacenados para operaciones complejas, pero no dice cómo obtenerlos. Además se plantea que las operaciones CRAE se deben implementar usando el lenguaje de consulta. En UML se incluye un lenguaje de especificación para describir eventos, acciones y actividades [155].

La riqueza en los diagramas para la modelación en UML no es explotada en función del diseño de la BD, sobre todo en lo concerniente a la dinámica [12][91].

Dentro de un desarrollo caótico de metodologías OO sin una estandarización de conceptos, es un paso importante la aparición de UML ya que ayuda a los diseñadores a comunicarse usando un lenguaje común [67]. Su definición hace el proceso de análisis y diseño bastante complejo (por la complejidad de los diagramas), en los que hay que tener cuidado porque arrastran (en menor escala) los problemas de la metodología de Booch de representar en más de una ocasión la misma información de manera distinta; y se pierde la vista global del problema al estar fragmentada en nueve diagramas la representación del mundo real. Aunque está notación tiene una alta capacidad expresiva, en [79] en esta tesis se muestra que hay aspectos en los que todavía es insuficiente. Por ejemplo, la señalización de algunas restricciones; se puede hacer usando estereotipos, pero no tiene un significado semántico como parte de un diagrama de persistencia [12] porque UML no incluye explícitamente un modelo de persistencia [11].

Rational Rose es una herramienta de modelación visual que se basa en UML [148]. Genera la estructura de las tablas, a partir del Diagrama de clases, aplicando las reglas básicas de conversión al MR a las clases, la cardinalidad de las relaciones y el tipo de dato de los atributos.

Ambler propone un modelo de persistencia para UML que se refiere a la parte estática [11] y que consiste en definir varios diagramas de clases cuyo título o los estereotipos utilizados indican el objetivo del diagrama. Por ejemplo, “modelo de persistencia lógico” o estereotipo <<entity>> (con las clases persistentes a las que llama entidades y sus relaciones) o “modelo de persistencia físico” o estereotipo <<table>> (con las tablas, los atributos definidos todos a partir del estándar ANSI-SQL y las relaciones entre tablas). Indica además cómo especificar vistas, índices, disparadores y procedimientos almacenados usando la notación del diagrama de clases. Por lo que hay aspectos estáticos y dinámicos, de los definidos en el epígrafe 1.3.1.2, que no se tienen en cuenta (por ejemplo, dominio de atributos y fórmulas de eveluación)

OO-Method [135][140] es una metodología de producción automática de software que proporciona una estrategia de traducción bien definida que permite implementar los modelos conceptuales en entornos relacionales avanzados.

Es una metodología completa que hace que el esfuerzo inicial en la fase de análisis pueda usarse en el diseño e implementación de manera fácil y predecible. En cualquier momento es posible obtener una aplicación ejecutable siempre que en los modelos de análisis esté correctamente definida la información, pues usa un lenguaje de especificación formal OO como almacén central de información. No está disponible la metodología ni la herramienta CASE (**Computer-Aided Software Engineering).** Aunque cada nueva versión ha avanzado en la simplificación del problema, el uso de la metodología desde la perspectiva del analista, específicamente en lo referente a la claridad y simplicidad conceptual de métodos y gráficos, sigue siendo un proceso complejo. Sus diagramas son muy ricos en cuanto a la capacidad semántica que permiten representar, pero la metodología es muy pobre en recomendaciones para obtener estos diagramas.

En los últimos años existe una tendencia a definir una capa intermedia, conocida como capa de objetos, que sirva de interfaz entre el almacenamiento de la información (en la que se puede usar una BDR o incluso jerárquica) y las aplicaciones que utilizan esta [9][10][17][106][116][119][177]. La definición más completa de esta capa se ha encontrado en los trabajos de Jesse Liberty y Ambler Scott.

La metodología de Liberty [119] propone la creación de una clase (a la que llama serialización), que en su interfaz tenga métodos para salvar y leer, y que cada clase persistente defina los métodos para implementar las operaciones CRAE. Esta propuesta es similar a la que brindan los lenguajes de alto nivel, con la diferencia de que es responsabilidad de diseñadores y programadores, la definición e implementación de la clase serialización y de los métodos.

Scott Ambler [17] propone la creación de una capa robusta de persistencia que encapsule el proceso de almacenamiento y consulta de las clases del dominio, en el medio de almacenamiento. Ambler define una arquitectura de capas, siendo la capa de las clases persistentes la única que se relaciona con la información almacenada a través de las operaciones CRAE. En este artículo se definen todas las clases asociadas al mecanismo de persistencia, las experiencias de su implementación se tiene en Java, C++ y Smalltalk. Solo se refiere a la implementación del mecanismo de persistencia hacia BDR y la definición de las clases de la capa persistente, está asociada a las características de su modelo, que solo toma en cuenta el comportamiento estático.

En los últimos años se ha extendido el uso de UML [63][119][123][161]. Hay autores que definen un modelo de persistencia [119], en otro se describe cómo transformar los elementos del diagrama de clases a tablas si se usa el MR [123] y existen trabajos donde no se aborda el tema [161].

En [63] se enfatiza en el desarrollo de aplicaciones para BD objeto/relacional (BDOR), BDO, BD activas y BDR. La metodología de Embley no usa UML. Tiene muy bien definidos los conceptos sobre los que se sustenta y se especifica formalmente, lo que permite la conversión automática. Es muy compleja y difícil de usar por la forma en que se representa la información y se trabaja sobre los diagramas en las distintas etapas. Si se siguen las recomendaciones de [162] para evaluar un buen estilo de diseño, aquí radica su principal desventaja ya que hay muchos modelos y notaciones y diferentes descomposiciones en un mismo nivel. A pesar de que incluye varios diagramas, es muy pobre en la explicación del modelo de persistencia utilizado.

En la tabla 2 se comparan estas metodologías tomando como referencias a [32][33] [66][119][123][132][133][135][136][140][154][158].

Tabla 2 Comparación de metodologías.

Parámetro	**OMT**	**OO-Method**	**Embley**	**Liberty**	**Martín**
Modelo de persistencia	Persistencia hacia BDR y BDO.	Persistencia hacia BDR	Persistencia hacia BDR, BDO, BDOR y BD activas.	Persistencia hacia ficheros, BDR y BDO.	Persistencia hacia BDR.
Herramientas para el diseño de la BD	Modelo de objetos. Los conceptos que se representan son similares a los definidos en el modelo entidad-relación. (MER).	Modelo de objetos para estructura estática. Modelo dinámico para eventos que actúan sobre objetos e interacciones entre ellos Modelo funcional para fórmulas de evaluación de atributos variables.	Modelo de objeto-relación que muestra la misma información del MER. Modelo de comportamiento de objetos. Modelo de interacción entre objetos.	Los de UML.	Los de UML.
CASE asociado	Varios que incluyen otras metodologías y otros específicos (OMTTool).	Sí, pero no disponible.	_	Los de UML.	Los de UML.
Definición formal de los conceptos	No	Sí, basado en el modelo O^3.	Sí, se basa en una lógica temporal que muestra una descripción ontológica del mundo real.	No	No
Pasos para el diseño de la BD	Pasos para la creación de una aplicación de gestión, que es muy general para la fase de diseño de la BD y para construir el modelo.	Pasos para construir una aplicación y cómo se obtiene información de los diagramas.	Pasos para construir una aplicación. No dice cómo convertir a los medios de almacenamiento.	Pasos para construir una aplicación.	-
Criterios para construir	Para identificar clases, atributos y relaciones.	Muy bien descritos los elementos que lo componen, pero casi nada	Para simplificar diagramas de forma iterativa, pero nada para	Para identificar clases, atributos y	_

Parámetro	OMT	OO-Method	Embley	Liberty	Martín
diagramas		de cómo construirlos.	su construcción inicial, que es bastante compleja.	asociaciones.	
Perspectiva Estática	Se representan clases, atributos, asociaciones con roles y atributos y modeladas como clase, herencia y agregación. Se definen restricciones sobre los atributos. Solo aparece cardinalidad máxima.	Se representan las clases elementales y complejas indicando eventos y atributos. Se indica la condición que provoca las especializaciones. De las agregaciones se indica cardinalidad y las dimensiones Inclusiva/ Referencial y Estática/ Dinámica.	Se representan clases, atributos y asociaciones con roles. Se indican las cardinalidades máxima y mínima. En las relaciones de herencia se especifican características de la especializada con respecto a la generalizada y viceversa. Se pueden expresar restricciones del dominio de los atributos.	Los de UML.	Los de UML. En cuanto a la agregación se añaden conceptos sobre la relación entre agregada y componentes.
Perspectiva Dinámica	No se describe cómo obtener las fórmulas dinámicas.	De los modelos dinámico y funcional se obtienen las evaluaciones, disparos y las precondiciones.	En el modelo de comportamiento se indica el evento o condición que provoca un cambio de estado. Del modelo de interacción se pueden obtener consultas y	Los de UML. Para cada clase persistente se definen métodos para las	Los de UML. Para el caso de las precondiciones se pueden asociar al diagrama de actividad o usando el lenguaje de especificación de
	Se describe cómo diseñar algoritmos para implementar operaciones, pero es muy general.	No se explica cómo transformar estas fórmulas al medio de almacenamiento porque no se asocian las fórmulas al diseño de la base de	actualizaciones a la BD. Además se puede expresar, usando el cálculo de predicados, restricciones de integridad de la BD.	operaciones CRAE y consultas para obtener estos datos.	UML. Se obtienen disparadores del diagrama de actividad o especificando reglas asociada al

Parámetro	OMT	OO-Method	Embley	Liberty	Martín
		datos.			evento.
Persistencia a BDR	Igual a lo propuesto por el MR pues centra su atención en los datos.	Se define el procedimiento para convertir clases elementales y compuestas, relaciones de herencia y asociaciones.	Solo se refiere a las ventajas de dividir en varias tablas o implementar en una sola.	Se refiere a la conversión de la herencia, asociaciones y colecciones.	Conversión a tablas, pero no profundiza.
Persistencia a BDO	Hay que añadir nuevas definiciones en dependencia de la forma en que el gestor maneje la persistencia. Propone las mismas reglas que si se convirtiera al MR.	_	Se define un algoritmo para llevar los modelos al estándar ODMG.	Plantea que no es necesario transformar nada porque es lo natural.	Solo se menciona que también podría almacenarse la información en BDO.
Persistencia a ficheros	_	_	-	Se explica cómo salvar los objetos a un fichero.	_

Las principales conclusiones que se derivan de la comparación de los métodos son:

- Estructura estática: siempre se incluyen diagramas, y en ocasiones se proponen especificaciones textuales cuando es difícil expresar algunas restricciones usando una notación gráfica (por ejemplo, Embley). No obstante **todas presentan carencias en su capacidad expresiva**, por ejemplo, OMT en cualidades que caracterizan la relación padre/hijo y entre la clase agregada y sus componentes. Embley y Liberty también presentan problemas en esto último. Martín y Liberty, son bastantes ricas en cuanto a las características de los atributos, pero no tienen en cuenta algunas propiedades; además son pobres en el tratamiento de las relaciones padre/hijo. OO-Method es insuficiente en la definición de restricciones de dominio.

- Estructura dinámica: casi todas incluyen diagramas que permiten obtener parte de esta información, **pero no reflejan el modo de utilizar esta información cuando se diseña la BD**. Las relaciones de disparo que se representan no están asociadas directamente al concepto de disparo de la BD porque necesariamente la activación del mecanismo no se da por un cambio en la BD.

- **En ninguna se explica cuáles son los pasos a seguir para diseñar la BD para los diferentes medios de almacenamiento**, por lo general solo los pasos para construir una aplicación, y en algunos casos no está explícitamente definido que el diseño de la BD es uno de ellos (por ejemplo OO-Method).

- **Solo OMT y Liberty dan recomendaciones para identificar los elementos representados en los diagramas de la parte estática**. En cuanto a la **vista dinámica no se dan recomendaciones para obtener el DTE**, en OO-Method se definen conceptos que pudieran contribuir, pero no se explotan en función de esto.

- Por lo general se hacen acompañar de herramientas **CASE, que generan parte de la estructura estática**.

- **No existe una tendencia a formalizar los conceptos sobre los que se sustentan.**

- **A excepción de los trabajos de Ambler y Liberty, no se definen mecanismos para encapsular la forma en que se almacenan los objetos.**

1.4 Conclusiones.

En estos últimos años se ha avanzado algo en la fundamentación del modelo y los SGBDO tienen un soporte teórico más sólido. El modelo O3 tiene una fundamentación en la lógica que permite describir con mayor precisión los aspectos estáticos y dinámicos de los objetos a través de un conjunto mínimo de conceptos (clase, operadores que permiten extender el concepto de clases y fórmulas dinámicas). Después de un estudio de varios formalismos, por estas razones, se decidió escogerlo como base para el modelo de persistencia que incluye el método DIBAO.

En la actualidad hay una coexistencia entre el MR y el MOO que se manifiesta por la presencia en el mercado de gestores que soportan ambos modelo, y otros que mezclan las técnicas OO con las relacionales [155]. El diseño requiere por lo tanto más esfuerzo por lo que esta no fue una actividad en la que se hicieron grandes

esfuerzos en los inicios del desarrollo de metodologías de análisis y diseño. La tendencia más explotada ha sido la conversión al MR.

Una BD es válida si sus datos satisfacen todas las restricciones de integridad especificadas en su definición [121]. Por lo tanto, el modelo de persistencia que se defina, debe facilitar la definición de todas las restricciones asociadas a los datos.

Como se apreció en este capítulo hay problemas en la propuestas metodológicas, sobre todo desde el punto de vista dinámico. Las deficiencias fundamentales se refieren a:

- Todas las propuestas tienen carencias en su capacidad expresiva, desde el punto de vista de la estructura de los objetos, aunque la suma de sus potencialidades permite una definición bastante completa.
- No reflejan cómo obtener las fórmulas dinámicas.
- No brindan recomendaciones para construir diagramas, sobre todo en lo relativo al comportamiento dinámico.
- Por lo general, parten de una definición informal de los conceptos del enfoque.
- No se definen los pasos para el diseño de la BD.

Por lo tanto, el método de diseño que se proponga debe tomar lo mejor de estas propuestas y solucionar las deficiencias encontradas.

CAPÍTULO 2
SOLUCIONES AL DISEÑO DE LA BASE DE DATOS

CAPÍTULO 2 SOLUCIONES PROPUESTAS PARA EL DISEÑO DE LA BASE DE DATOS.

El estado del arte en el campo de las bases de datos de objetos (BDO), condiciona las alternativas del diseño de la base de datos (BD) que se proponen en este capítulo, tomando como base los resultados del análisis realizado utilizando la metodología **A**nálisis y **D**iseño **O**rientado a **O**bjetos de **S**istemas **I**nformáticos (ADOOSI) [5][7].

En la propuesta presentada se hace mayor hincapié en el caso de la utilización, como medio de almacenamiento, de un gestor relacional u objeto/relacional, por las transformaciones que implica pasar de un pensamiento en términos de objetos a uno de entidades. En la conversión se mantienen algunos de los conceptos de la orientación a objetos (OO). Se incluyen las especificaciones para el caso en que se utilicen los ficheros o un sistema de gestión de base de datos de objetos (SGBDO).

El método para el DIseño de la BAse de datos a partir del modelo orientado a Objetos (DIBAO), está compuesto por dos partes: un modelo de persistencia y una capa persistente de clases (Figura 2.1). El modelo de persistencia describe los pasos para el diseño de la BD, las herramientas que se deben utilizar en este proceso, cómo convertir las clases al medio de almacenamiento seleccionado y cómo interpretar la información contenida en los diagramas y especificaciones textuales, en función de este diseño. La capa persistente de clases incluye las subcapas de especificación y de interfaz, que se encargan de definir las clases para describir las nuevas especificaciones asociadas a las relaciones entre los objetos, que se describen en el diccionario de clases; y la relación con el medio de almacenamiento, respectivamente.

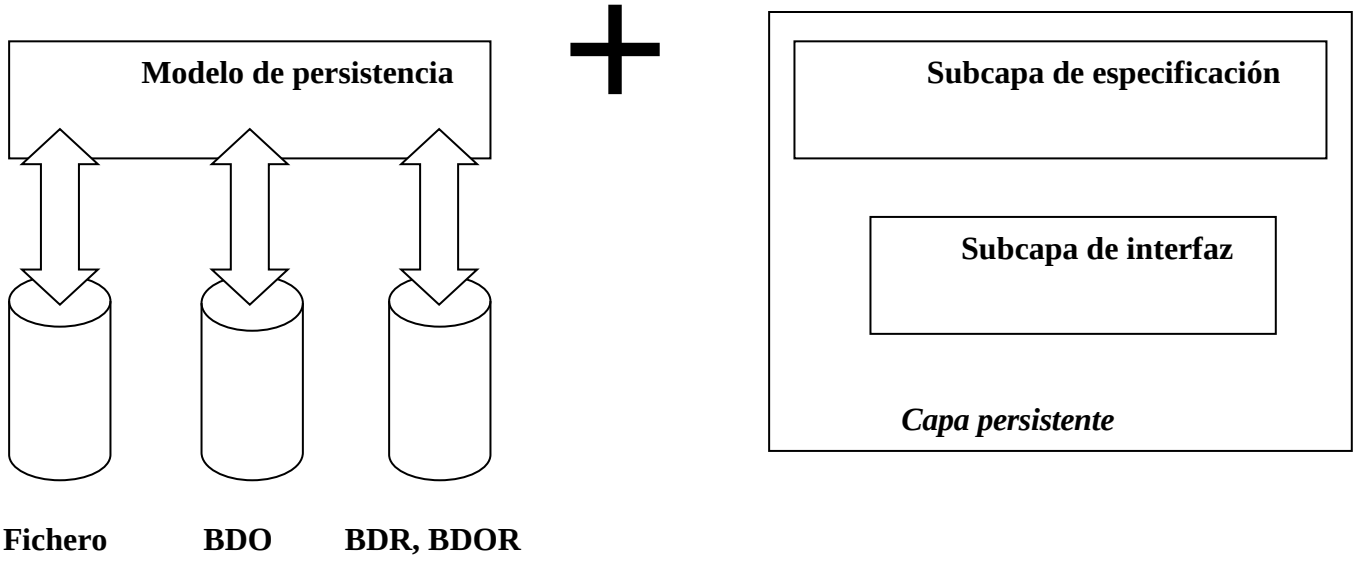


Figura 2.1 Estructura del método DIBAO.

En este capítulo se describen las principales características del método DIBAO, haciendo hincapié en los puntos que lo hacen superior a otros métodos de diseño.

2.1 Metodología ADOOSI.

La metodología ADOOSI es utilizada en Cuba para enfrentar proyectos que sigan este enfoque y con ella se han desarrollado gran cantidad de proyectos desde el año 1993. Abarca todas las etapas del ciclo de vida de producción de un software, aunque tal como se

planteó en el capítulo anterior, en sus primeras versiones abordaba poco el diseño de la BD. Tiene en cuenta las nuevas tendencias en los medios de programación que incluyen características visuales. Es una metodología pura OO por lo que se basa en el principio cliente-servidor.

La fase del diseño de la BD se define en la metodología después de otras que incluyen herramientas y procedimientos para determinar las clases con sus atributos y responsabilidades. De cada atributo se ha especificado es su nombre, el tipo de dato y su visibilidad y de las responsabilidades su nombre, parámetros, valor que devuelve en caso de que lo requiera, y una descripción en lenguaje natural del algoritmo asociado.

En [23] se plantea que si hay una cadena de pasos en el proceso de modelación, cada eslabón debe trabajar. Lo importante en este momento no son las herramientas y técnicas aplicadas para obtener los requerimientos del software, sino las clases que se obtuvieron como resultado para representar el dominio a modelar.

En sus continuas versiones ADOOSI ha incluido algunas especificaciones que se adaptan a las potencialidades de los gestores de BD presentes en el mercado, en correspondencia con los resultados parciales de esta tesis. De manera que el método DIBAO forma parte de la metodología, dotándola de procedimientos y herramientas para el diseño de la BD, de los cuales carecía en sus primeras versiones [4].

2.2 Pasos para el diseño de la BD.

Independientemente del medio de persistencia seleccionado para almacenar los objetos persistentes, se debe realizar un grupo de pasos que permitan completar la semántica de los objetos en aspectos que son importantes referentes a su estructura estática y comportamiento dinámico.

Los pasos que propone esta tesis en el diseño de la BD son:

1. *Definir las clases persistentes.*
2. *Refinar las clases.*
3. *Clasificar las clases y los atributos.*
4. *Realizar el diagrama de clases.*
5. *Realizar el diagrama de transición de estado.*
6. *Obtener las restricciones estáticas y las fórmulas dinámicas.*
7. *Convertir las clases al medio de almacenamiento.*

En el último paso es en el que se tiene en cuenta hacia qué lugar se guardarán los objetos, por lo que se particulariza qué hacer en cada caso.

2.2.1 Definir las clases persistentes.

No todas las clases [...] son persistentes.

Todas las clases identificadas en el dominio del análisis no son persistentes. La persistencia es la capacidad de un objeto de mantener su valor en el espacio y en el tiempo. Las clases temporales son manejadas y almacenadas por el sistema en tiempo de ejecución por lo que dejan de existir cuando termina el programa [101]. Es responsabilidad del diseñador definir

cuáles clases son las que deben ser persistentes. En este proceso la autora recomienda aplicar algunas reglas, ellas son [84][90]:

1. Cuando una clase que está formada por otras clases es persistente, automáticamente las clases componentes también son persistentes. Lo contrario no se cumple necesariamente.
2. Cuando una clase hija en una jerarquía de clases es persistente, automáticamente son persistentes sus ancestros en el árbol de jerarquía. Lo contrario no se cumple necesariamente.
3. Cuando se define como persistente a una clase que agrupa a objetos de un mismo tipo de clase base (se refiere a las clases listas, colecciones, registros), entonces automáticamente son persistentes todas las clases hijas a partir de la clase base, incluyendo a la clase base.
4. Cuando hay herencia múltiple, esta debe ser resuelta antes si el medio de almacenamiento a utilizar no soporta este concepto. Una solución es que la clase hija herede de la clase de la que redefine sus métodos y añada un atributo pasivo del tipo de la(s) otra(s) clase(s) de la que heredaba. Si se redefinía comportamiento de más de una clase padre, hay que escoger de cual se quedaría heredando y añadir un atributo pasivo por cada una de las clases padres de las que no heredará. Los métodos que se redefinen, que ya no se reciben por herencia, en su implementación incluirán las relaciones con las clases padres.

En fases anteriores dentro del ciclo de vida del desarrollo de un proyecto, puede que se hayan definido clases que coordinan el trabajo de varias clases, por lo que el comportamiento que de ellas interesa es el asociado a la función controladora. El uso de clases controladoras ha tomado en los últimos tiempos una importancia crucial cuando se describe un problema [116]. En las propuestas de diseño de la BD, mencionadas en el capítulo 1, no se hace alusión a que hacer con este tipo de clases dentro del modelo de persistencia. En esta tesis se recomienda no definirlas como persistentes porque conceptualmente no lo son.

2.2.2 Refinar las clases.

El objetivo de este paso es la obtención de la jerarquía de clases y la definición de nuevas clases [5]. No es obligatorio que aparezcan nuevas clases, se recomienda revisar si existe algún comportamiento de interés que no haya sido tomado en cuenta y sea trascendental en la solución del problema, para incluirlo.

En etapas anteriores puede que se hayan identificado relaciones del tipo “es un” (generalización/especialización), en este punto se revisan ya que si la clase hija no hereda todo lo definido en la clase padre, debe incluirse un nueva clase con lo común, de la que ambas hereden [5].

También hay que revisar si existen clases con comportamiento y atributos comunes. En estos casos se define, para cada grupo de clases análogas, una clase con lo común. Las clases hijas se quedarán con lo diferente y el comportamiento que sea necesario redefinir. Con este procedimiento se crean nuevas relaciones padre/hijo [5].

2.2.3 Clasificar las clases y los atributos.

Las clases en este trabajo se clasifican en dependencia de los valores de los atributos que las integran [90] en:

- *Simples: todos sus atributos toman valores atómicos.*
- *Compuestas: es el resultado de la unión de varias clases para formar una entidad conceptual con significado en el dominio. Este tipo de asociación puede ser una composición o una agregación y puede tener otros atributos que tomen valores atómicos asociados a la relación.*
- *Complejas: son clases que tienen uno o varios atributos que referencian a otros objetos de otras clases o de ella misma. La relación de ella con respecto a las clases que contiene es referencial, es decir, un cambio en la primera no afecta a la segunda. Tradicionalmente se reconocen como las relaciones de asociación entre dos o más clases.*

Existen varias clasificaciones de los atributos de una clase. En este trabajo se parte de la clasificación propuesta en [140], que identifica a los atributos en alguna de las siguientes categorías: dinámico, estático o derivado.

Un atributo estático no cambia de valor en el tiempo por lo tanto no puede ser actualizado. El único evento que lo afecta es el que provoca la creación de la clase que como consecuencia le da valor. A diferencia de los atributos estáticos, los dinámicos se reconocen porque son afectados por otros eventos que son los que hacen que cambien de valor [55]. Los atributos derivados cambian cuando se modifican otros atributos. Estos otros atributos integran la fórmula de derivación y pueden pertenecer o no a la clase a la que pertenece el atributo derivado. Estos atributos se pueden especificar utilizando disparadores (por ejemplo, SQL Server).

2.2.4 Realizar el diagrama de clases.

La mayoría de las características de la perspectiva estática son capturadas gráficamente en el diagrama de clases (DC) cuya notación toma como base la del diagrama de estructura de UML [14][155], aunque se añaden nuevas características usando estereotipos y notas. En el anexo 1 se describe la notación.

La autora toma la definición de OO-Method [140] de dimensión estática/dinámica (E/D), para las relaciones entre la clase compuesta y cada una de sus clases componentes, porque añade una restricción de integridad no representada en la notación UML. Esta dimensión se define como [118]:

Estática	*El objeto de la clase componente se mantiene sin cambiar su valor, independientemente de los eventos que afecten a la clase compuesta .*
Dinámica	*Producto de eventos que afectan a la clase compuesta, puede cambiar el objeto componente asociado a la clase compuesta.*

Cuando dos clases se relacionan, ya sea por composición o por complejidad, hay que especificar en ambos extremos de la relación las cardinalidades máximas y mínimas.

En la realización del DC esta tesis recomienda seguir los siguientes pasos [91]:

1. *Una clase simple se representa directamente.*
2. *Revisar las asociaciones "es un" para representar la herencia. Se debe indicar el atributo del padre y el valor que toma, para que un padre se especialice en cada hija. Si este atributo no existe en la definición de la clase padre, se añade. Se especifica si las herencias que se derivan del padre, son: total/parcial (si es total todos los objetos del tipo de la clase padre se especializan en un objeto de una clase hija) y con o sin solapamiento (solapamiento implica que al menos un objeto del tipo de la clase padre se especializa en dos o más objetos de clases hijas diferentes).*
3. *Identificar y representar clase compuestas, especificando la cardinalidad y la dimensión E/D. Para la cardinalidad, esta tesis recomienda tener en cuenta los siguientes criterios:*

   - *En el extremo de la clase compuesta como máximo la cardinalidad es 1, para la agregación puede ser mayor.*
   - *En el extremo de la clase componente la cardinalidad mínima por defecto es 1, la máxima depende de: si no es una agrupación es 1, si es una agrupación es m y si la clase compuesta tiene más de un atributo de igual tipo (por ejemplo, la clase matrimonio tiene dos atributos de tipo persona, una para cada testigo), la relación es con la clase y con la cardinalidad máxima igual a la cantidad de atributos iguales.*

   Estos valores por defecto son los mínimos que se pueden tomar a partir de las relaciones reflejadas en la definición de las clases.
4. *Identificar y representar las clases complejas, especificando la cardinalidad. Son válidas las recomendaciones anteriores en cuanto a la cardinalidad, excepto en el extremo de la compleja que la cardinalidad mínima debe ser <0,1> porque no es una relación de todoparte.*

Un ejemplo de un DC se representa en la figura 2.1, en el anexo 2 se describe el fragmento del caso de estudio que se utiliza en los ejemplos en esta tesis.

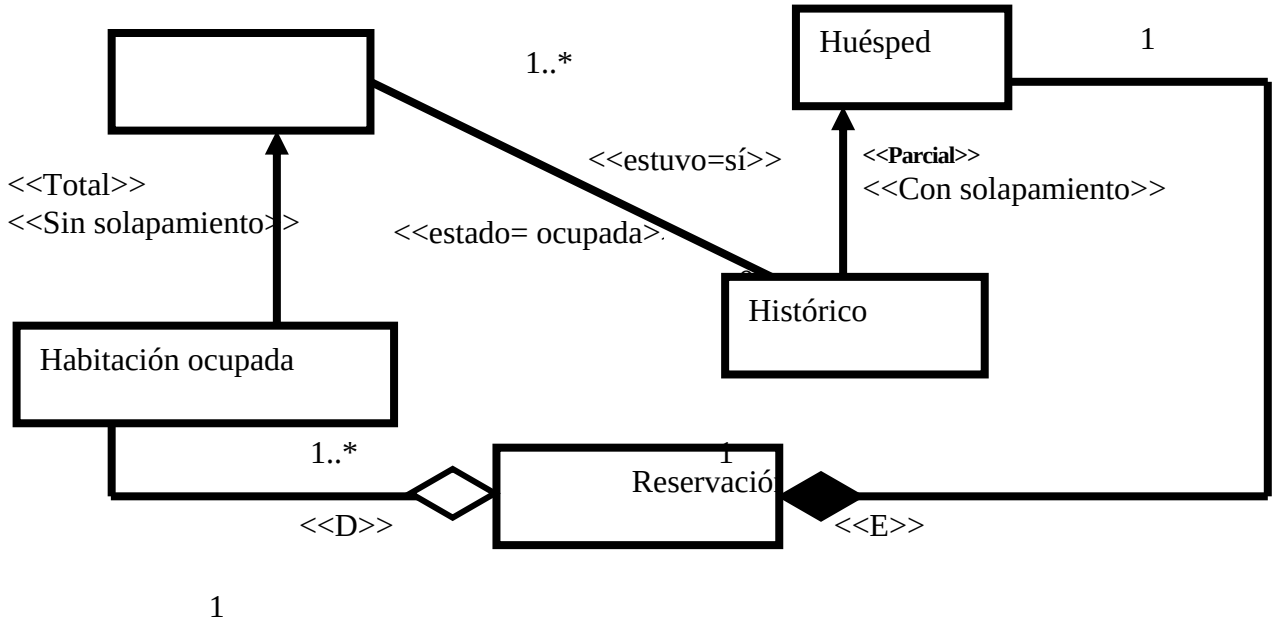


Figura 2.1 Fragmento de un DC de un sistema de reservación de habitaciones.

2.2.5 Realizar el diagrama de transición de estados. SEÑALADOR

Para mostrar la dinámica del comportamiento de un sistema, se ha hecho uso durante años de los diagramas de transición de estado (DTE) [48]. Es una herramienta que ha demostrado su utilidad para capturar restricciones de integridad dinámica [180], aunque no es suficiente. Esta tesis recomienda utilizarlos en función del diseño de la BD, construyéndolos para aquellas clases que posean atributos dinámicos pues en función de estos es que están los posibles estados por los que transita un objeto. Un DTE está compuesto por: estado regulares, estados agregados, estados finales, estados iniciales y transiciones.

Los estados regulares contienen restricciones de integridad, que son reglas que especifican restricciones y requerimientos que deben afectar a la estructura y al comportamiento de los objetos y deben ser tratados en los métodos de la clase.

Las transiciones tienen asociadas acciones (que son operaciones que se realizan instantáneamente), eventos (es la lista de eventos que pueden provocar –o disparar- la transición, relacionados entre sí por los operadores lógicos OR y AND, y pueden estar precedidos del operador lógico NOT) y condiciones (es una lista de condiciones relacionadas entre sí de forma similar a los eventos, que serán evaluadas en el momento en que se produzcan los eventos que dan lugar a la transición y esta sólo será disparada si el resultado de dicha evaluación es verdadero, de lo contrario se esperará a que ocurran nuevamente los eventos). Para especificar una transición se sigue el siguiente formato:

[<clase que genera evento>]:<evento>[[condición]][/acción].

La descomposición en varios niveles del DTE, utilizando los estados agregados, es un tópico poco definido en cuanto a cuáles criterios utilizar para agrupar. En [49] se recomiendan los estudios de George Miller que indican la cantidad de 7 ± 2 elementos por nivel y que lo general esté en los niveles más altos y lo particular en los más bajos. La autora de la tesis propone dos criterios para agrupar por niveles. Un criterio está relacionado con los atributos, de manera que si existen varios eventos que afectan a un mismo atributo ubicando al objeto en estados diferentes, con todos estos estados podría definirse uno agregado. Otro criterio sería agrupar a estados que están fuertemente acoplados, en el ejemplo de la figura 2.3 este es el criterio que se aplica.

Los diferentes niveles de diagramas deben estar balanceados en cuanto a las transiciones de entrada y salida cumpliéndose las siguientes reglas [53]:

- Para el caso de las transiciones de entrada es necesario que cada uno de los eventos en OR, de la lista de eventos de cada una de las transiciones de entrada al estado agregado, esté formando parte de la lista de eventos de al menos una de las transiciones que parten del estado inicial hacia los subestados.
- Para el caso de las transiciones de salida debe cumplirse: cada subestado tendrá una transición al subestado final y cada una de estas, contendrá entre su lista de eventos las listas de eventos de todas las transiciones de salida del estado agregado relacionadas entre sí por el operador lógico OR.

En la figura 2.3 y tablas 2.1 y 2.2 se muestra el DTE, y fragmentos del diccionario de datos que complementa el diagrama; en el que se describen las transiciones y nodos, respectivamente, correspondientes a la clase Reservación. En la representación gráfica se

puede incluir la especificación contenida en el diccionario de datos, indicando en vez del nombre de la transición (por ejemplo, T8) la descripción de la transición. En el caso de las actividades, se divide en dos partes el símbolo de un estado, tal como se aprecia en el estado *Prorrogando reservación*.

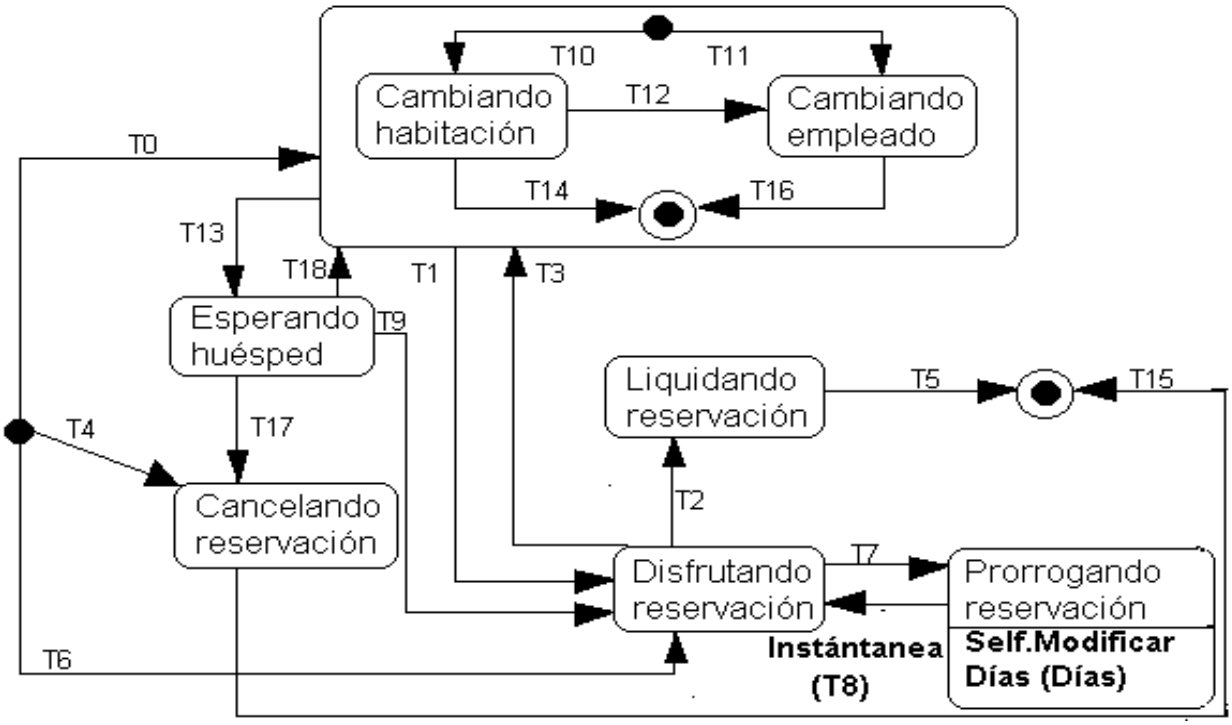


Figura 2.3 DTE de la clase reservación de habitaciones de un hotel.

Tabla 2.1 Fragmentos del diccionario de datos. Transiciones.

Transición	**Descripción**
T1	Instántanea[Carpetero.Existehuespéd(Huésped dado)]
T2	(Carpetero:Huésped se va antes) OR (self.Vence reservación)
T3	Carpetero:Cambia Habitación OR Cambia empleado/ New Modificación('cambia habitación o empleado')
T6,T9	Carpetero:Llega huésped
T7	Carpetero:Huésped pide prórroga[Fecha actual=self:.Obtener día de salida]/New Modificación('prórroga')
T16	Instantánea
T11	Carpetero:Cambia empleado
T12	Ama de llaves:Habitación pertenece a otro grupo

Tabla 2.2 Fragmento del diccionario de datos. Nodos.

Nodo	**Actividad**	**Restricción**
Disfrutando reservación	Carpetero.Registrar llamadas telefónicas Ama de llaves.Registrar servicio de habitaciones	not self.Vence reservación

Nodo	Actividad	Restricción
Cambiando empleado	SI J'Personal.Devolver estado (Empleado que lo atiende)=ocupado J'Personal.Cambia estado(Empleado que la atiende) FINSI J'Personal.Buscar empleado(Habitación ocupada; Nuevo empleado) Empleado que la atiende:=Nuevo empleado SI J'Personal.Empleado a tope (Empleado que lo atiende) J'Personal.Cambia estado(Empleado que la atiende) FINSI	-

En la especificación de los DTE es importante clasificar los atributos dinámicos de acuerdo a la forma en que cambian su valor en el tiempo, por esto esta tesis toma como base la propuesta descrita en [140], que clasifica a los atributos dinámicos en:

a. *Cardinales: el efecto en el atributo es el incremento/decremento en 1 o una cantidad dada (tabla 2.3).*

Tabla 2.3 Ejemplo de atributo cardinal: Cantidad de días.

Acción incrementadora	***Acción decrementadora***	***Acción reinicializadora***	***Condición***	***Efecto de la acción***
Carpetero: Huésped pide Prórroga	-	-	-	*+ Días*

La condición restringe el dominio del atributo.

b. *Característicos de un estado: el atributo adquiere un valor que es independiente del valor que haya tenido con anterioridad (tabla 2.4)*

Tabla 2.4 Ejemplo de atributo característico de un estado: Número de habitación ocupada (refleja el número de la habitación asignada a un Huésped).

Acción portadora	***Efecto de la acción***
Carpetero: Cambia habitación	*Ama de llaves.Buscar habitación disponible (Tipo; Nueva habitación)*

c. *Perteneciente a una situación: el atributo toma valor en un dominio limitado. El nuevo valor depende del anterior y en un estado dado solo se pueden tomar determinados*

valores. En el caso de la reservación no hay ningún atributo con estas características. El ejemplo clásico es el atributo estado civil (tabla 2.5).

Tabla 2.5 Ejemplo de atributo perteneciente a una situación: Estado civil.

Valor actual	***Acción portadora***	***Nuevo valor***
-	*Registro civil: Nacer*	*Soltero*
Casado	*Notario: Divorciar*	*Divorciado*
Casado	*Registro civil: Enviudar*	*Viudo*
Not casado	*Notario: Casar*	*Casado*

Las acciones se corresponden con los eventos definidos en el DTE. En las clasificaciones a y b, el efecto de la acción está descrito dentro de las acciones a ejecutar en el estado. Para los atributos clasificados como c, la especificación describe restricciones de integridad dinámica de la clase que se obtienen cuando se sigue la traza de las transiciones de estado que están asociadas con un mismo atributo.

Para construir el DTE esta tesis recomienda seguir los siguientes criterios:

- *Clasificar los atributos dinámicos en alguna de estas tres categorías.*
- *Si el atributo es clasificado como a, identificar los eventos que lo afectan teniendo en cuenta cuáles aumentan su valor, cuáles lo decrementan y cuáles lo reinicializan, agrupar todos los eventos que tengan el mismo efecto y el mismo tipo de acción (incrementadora, decrementadora y reinicializadora), y definir un estado que refleje esta situación. Si hay más de un evento en está unión, se relacionan usando el operador OR.*
- *Si el atributo se clasifica como b, identificar los eventos y el efecto que provocan, agrupar usando el operador OR los eventos que provocan una misma forma de obtener el nuevo valor, y definir un estado para ellos.*
- *Si el atributo es clasificado como c, identificar todos los eventos, el nuevo valor que provocan y el valor del atributo para el cual ese evento tiene sentido. Se definen tantos estados, como posibles valores existan.*

Si se tiene en cuenta la primera de las recomendaciones en cuanto a particionamiento que se daba en el epígrafe 2.2.1, los atributos clasificados como a y c son fuente para definir estados agregados.

2.3 Realización de los comportamientos estático y dinámico.

En la implementación de los comportamientos estático y dinámico se ha tomado como referencia las definiciones formales del modelo O^3 ya que:

- *Las clases representadas en el diagrama de clases tienen perfectamente definidos sus atributos, a los cuales, este modelo de persistencia, define que se le asocie un estereotipo que indique su tipo (<<estático>>, <<derivado>>, <<dinámico-cardinal>>, <<dinámico-característico>> o <<dinámico-perteneciente>>).*
- *Asociadas a las transiciones se indican los eventos, que provocan los cambios de estado de los objetos vinculados a un tipo de clase, y qué clases generan esos eventos. La suma*

de todos los eventos definidos en las transiciones del DTE de una clase, constituyen el conjunto de eventos (E) que condicionan el estado de sus objetos. Los eventos de creación y destrucción están contenidos dentro de los estados inicial y final, respectivamente.

- *En las tablas asociadas a los atributos clasificados como dinámicos en el epígrafe anterior, se especifican los eventos que afectan a cada atributo, y en caso de que deban seguir un orden lógico de ocurrencia, la clasificación del atributo como* perteneciente a una situación *indica las secuencias válidas. De esta forma también es posible identificar para cada clase la secuencia de eventos que conforman la vida del objeto y cómo se relacionan estos eventos con los valores que pueden tener los atributos dinámicos.*
- *Los operadores de agregación y especialización/generalización, se representan directamente en el DC. Las restricciones asociadas a estos operadores (dimensión, cardinalidad, relación padre/hijo en el árbol de jerarquía) constituyen restricciones de integridad dinámica.*
- *En el DC se especifica para cada atributo el tipo de datos asociado. Otras restricciones de dominio se pueden especificar siguiendo las recomendaciones propuestas en el epígrafe 2.3.1.*
- *Las fórmulas dinámicas de precondiciones y disparadores; se pueden obtener del DTE y en el caso de las derivaciones se especifica textualmente, tal como se verá ene el epígrafe 2.3.2.*

2.3.1 Vista estática.

El DC incluye la definición de restricciones de integridad estática del tipo de datos asociado a cada atributo. Para el resto de estas restricciones, el método DIBAO sugiere que se especifiquen textualmente completando para cada clase lo indicado en la tabla 2.6.

Tabla 2.6 Especificación de atributos de una clase.

Atributo	Único	Nulo	Rango	Valor por defecto	Fórmula de derivación

/ /

Para el caso de numérico o enumerativo *Para los atributos derivados*

Las restricciones a columnas descritas en la tabla 2.6, se corresponden con diferentes categorías que se referencian en la literatura. Existen otras condiciones que deben satisfacer los atributos y que no es posible categorizarlas, por lo que se pueden definir como restricciones. Para todos aquellos atributos, que requieran especificar restricciones al dominio más restrictivas, hay que especificar reglas que indiquen qué valores pueden tomar, o lo que es igual, qué condiciones deben cumplir los atributos. El formato general que se propone en esta tesis para expresar estas restricciones es: <atributo> = <condición>. En la condición hay que especificar las clases involucradas y se puede usar NOT, EXIST, funciones de agregación, AND, OR, IN, BETWEEN. Usando la simbología de notas de UML, se pueden incluir en el DC. Por ejemplo,

Pasaporte=NOT EXIST IN Huésped

(Huésped.#pasaporte = Nuevo#pasaporte) AND
(Huésped.País=Nuevo.País)

Otra característica de un atributo que puede ser implementada de esta forma es cuando el atributo es tipo enumerativo. Este tipo de dato aparece por primera vez en el estándar SQL3 [70], por lo que aún varios gestores no lo incluyen. Por ejemplo, Tipo de habitación =’Suite’ OR ’Simple’ OR ’Doble’.

Las restricciones se validan, para cualquier atributo independientemente de su clasificación, cada vez que se intente cambiar su valor. Su implementación depende de los predicados que brinde el gestor seleccionado.

2.3.2 Vista dinámica.

Las fórmulas dinámicas definidas en el modelo O^3 están presentes en la metodología OO-Method [135][140], pero se utilizan otras herramientas y procedimientos para obtenerlas (excepto las evaluaciones) y no se realiza este proceso en función del diseño de la BD.

Del DTE se pueden obtener varias de las fórmulas dinámicas. En función del diseño de la BD, a continuación se describe cómo esta tesis propone derivarlas del diagrama, aunque hay algunas características dinámicas que requieren de nuevas formas de especificación.

Precondiciones

Las precondiciones, se obtienen directamente del DTE, como las condiciones que deben cumplirse para que ocurra un cambio de estado. Estas precondiciones quedan reflejadas, en la descripción de las transiciones, como la unión del evento que ocurre y las condiciones que deben evaluarse para provocar el cambio. En esta tesis se concluye que las restricciones que tienen los estados se deben cumplir para que el objeto se mantenga en ese estado, por lo tanto, lo contrario de una restricción tiene que ser una condición de alguna de las transiciones de salida que se deriven de ese estado.

Evaluación

Si en el DTE solo se representa de un estado su nombre, es difícil determinar cuáles atributos han sido afectados [73]. En la descripción de las actividades, que se ejecutan cuando se está en un estado, están definidos los pasos que se efectúan para dar un nuevo valor a el o los atributos dinámicos que se afectan (por ejemplo, los estados “cambiando habitación” y “cambiando empleado”) y también se pueden incluir las acciones que se están haciendo mientras el objeto está en ese estado (por ejemplo, el estado “disfrutando reservación”). En esta descripción se puede identificar al o los atributos que se modifican pues, directamente o como parte de un método de la clase, se utiliza una asignación en la que en la parte izquierda está un atributo de la clase, y en la derecha el nuevo valor.

Las fórmulas de evaluación describen los posibles valores que puede tomar un atributo dinámico, a partir de un estado dado, y cómo se determina ese nuevo valor. Su especificación (al igual que en OO-Method) se obtiene de las tablas presentadas en el epígrafe 2.2.5, que se construyen para cada atributo dinámico en dependencia de su tipo. Una vez procesados todos los atributos dinámicos de todas las clases, se tienen todas las fórmulas de evaluación.

Disparadores

Una forma de mantener la integridad de la BD es definir disparadores [52]. Un disparador de una BD es una sentencia que el sistema ejecuta automáticamente como efecto

secundario de una modificación de la BD [109]. Siguiendo las reglas propuestas en [40], solo se deben usar disparadores para garantizar la integridad y la lógica de control y no como una herramienta para la validación de los datos porque implican operaciones internas costosas en tiempo y reducen la portabilidad de la BD ya que cada gestor tiene su forma particular de implementarlos.

No siempre un cambio de estado tiene como antecedente o evento que lo provoque, un valor específico de uno o varios atributos, por lo que hay disparadores de la BD que no se obtienen del DTE. Un mecanismo de disparo se activa cuando en se adicionan, modifican o eliminan elementos o cuando un atributo toma un valor determinado.

El método DIBAO propone que la especificación de los mecanismos de disparo debe seguir el siguiente formato:

<causa que lo provoca>	*<clase [condición >:]*	*/<acción >*	*{ANTES/DESPUES}*
↓	↓	↓	
CREAR *ELIMINAR* *MODIFICAR*	*Nombre de la clase Nombre de la clase Nombre de la clase,* *atributo y valor*	*Acciones que se ejecutan como efecto del disparo.*	*Momento en que se producen las acciones con relación al cambio que provoca el disparo.*

Por ejemplo,

MODIFICAR Fecha.FechaActual: /Reservación.Envía nota a huéspedes {DESPUES}

Algunas transiciones tienen asociadas acciones. Hay ocasiones en que estas acciones podrían implementarse como las acciones definidas en el disparador. En estos casos la causa sería la modificación (MODIFICAR) del o los atributos que se afectan cuando se pasa al nuevo estado. La propiedad antes/después tomaría el valor de antes. Esto es válido cuando ocurre algo de lo siguiente:

- Solo hay un estado y una transición que llega a ese estado, para el o los atributos afectados.
- Solo hay un estado relacionado con el cambio del o los atributos y todas las transiciones que llegan a él tienen asociadas las mismas acciones.
- Hay varios estados relacionados con el cambio del o los atributos y todas las transiciones que llegan a ellos tienen asociadas las mismas acciones.

En el ejemplo de la figura 2.3 y el fragmento del diccionario de datos de la tabla 2.1, la acción instantánea que se produce en la transición T7, genera una regla de disparo. Cada vez que se modifica el atributo *cantidad de días*; se inserta automáticamente una tupla a la tabla Modificación. Es responsabilidad del diseñador, en caso de no poseer una herramienta CASE, verificar y garantizar que no se duplique el disparador, cuando se especifique textualmente estos mecanismos.

Las fórmulas de disparo del modelo O3, se pueden obtener del DTE de la misma forma que se definen en la metodología OO-Method, ya que queda reflejadas en la especificación de la

transición. Aunque el formato para especificar transiciones en este método es más rico porque incluye también las acciones

Derivación

Las fórmulas de derivación de los atributos derivados, se definen textualmente utilizando los operadores matemáticos necesarios, valores constantes y atributos (pueden ser de la clase a la que pertenece el atributo derivado o de otra clase, en cuyo caso hay que especificar de cuál clase: <nombre de la clase>.<nombre del atributo>).

Especificación de actividades, acciones y métodos de las clases

Se propone utilizar en la especificación el lenguaje en UML [155] para describir las acciones y actividades del DTE, las acciones asociadas a los mecanismos de disparo y los métodos de las clases, si no es suficiente, se puede usar otras construcciones como las decisiones, selección y repetición. En estos casos se utiliza un subconjunto del lenguaje natural como el siguiente, que toma como referencias las recomendaciones descritas en [3][6][64][171], definiendo esta tesis las siguientes construcciones:

- En el caso de que el método a ejecutar sea de la propia clase a la que corresponda el DTE, se sustituye el nombre de la clase por self.
- Cuando un mensaje se envía a todas las clases, se antecede por la palabra ALL.
- *Decisión*

  SI <condición>

  <acciones>

  [sino

  <acciones>]

  FIN SI
- *Selección*

  SELECCIONAR CASO

  <nombre del caso>[<condición>]:<acciones>

  ...

  [otro caso:<acciones>]

  FIN CASO
- *Repetición*
  - *PARA CADA [<condición>] DE <nombre de la clase>*

    <acciones>

    FIN CADA
  - *MIENTRAS <condición>*

    <acciones>

    FIN MIENTRAS

- *REPETIR*

  <acciones>

  HASTA <condición>

- *Precondiciones/Postcondiciones: Es una de las alternativas para describir que implica que no es necesario utilizar las construcciones anteriores pues no se describe paso a paso el algoritmo. Se indica en lenguaje natural el estado antes y después de los objetos y sus relaciones, especificando qué objetos se crean, cuáles se eliminan o cuáles objetos y qué modificación ocurre.*

En todos los casos la condición restringe el subconjunto de objetos sobre los que se ejecutan las acciones indicadas.

Esta propuesta difiere del lenguaje natural ya que tiene un vocabulario y una sintaxis limitados, la sintaxis es rígida y no permite variaciones de estilo [169]. La especificación permiten expresar toda la semántica de un algoritmo.

En la descripción se pueden usar las funciones de agregación: contar, sum (sumar), avg (promedio), máx (máximo) y mín (mínimo), indicando sobre qué elementos se hacen efectivas. Se puede especificar una condición que restringe la cantidad de objetos que se analizan. El formato de la especificación se propone que sea:

<función de agregación> (<clase o clase.atributo>) [[condición]].

2.4 Alternativas en la conversión de las clases al medio de almacenamiento.

La propuesta de diseño de base de datos que incluye esta tesis se divide en tres partes, que dependen del sistema que se utilice para implementar la solución al problema planteado. Estas tres soluciones son [87]:

- Si se tiene un SGBDO, hay que garantizar que las definiciones obtenidas se correspondan con el modelo ODMG. La conversión es directa [30].
- Si lo que se tiene es un LPOO que permite el trabajo con una base de datos relacional (BDR) o un SGBDOR, la solución es llevar las clases definidas a tablas.
- Si lo que se tiene es un LPOO que no permite el trabajo con base de datos relacional, pero que permite salvar registros, entonces se deben utilizar las clases que brinda para almacenar los atributos de las clases persistentes en ficheros. Puede que el lenguaje permita la relación con BDR, pero el diseñador escoge como forma de almacenamiento la organización en ficheros.

La autora coincide con [28] en que la selección depende de las aplicaciones y los datos que maneja.

2.4.1 Implementación de la persistencia para el almacenamiento en ficheros.

Cuando se utiliza esta forma de almacenamiento, se debe garantizar para las clases persistentes que: se añadan métodos para salvar y recuperar información y se defina el código de estos métodos, de acuerdo al tipo de los atributos [88].

Almacenar los objetos de esta forma implica que se crea un fichero cuya estructura no es fija pues se almacenan objetos de diferentes clases, se leen y escriben bytes.

La forma en que los lenguajes de programación implementan la persistencia es similar a la forma en que la garantizaban los productos de la 1ra generación de SGBDO. La diferencia principal es que los lenguajes almacenan los atributos de los objetos y los gestores guardan también parte del comportamiento asociado a las clases.

Lo anterior implica que si se desea que sea persistente el comportamiento de los objetos, se puede almacenar su implementación y la definición de la clase, pero es responsabilidad del diseñador definir todo lo necesario para recuperar y salvar esta información. La posibilidad de almacenar la estructura completa de una clase está sustentada en el concepto de metaclase. Una metaclase define la estructura y comportamiento de una clase [31], por lo que en tiempo de ejecución se crean clases. Esto puede hacerse siempre y cuando el lenguaje de programación lo permita.

2..4.2 Modelo de objeto de ODMG.

El modelo de objetos es la base del estándar ODMG [25]. Este modelo plantea que un objeto está formado por propiedades y comportamiento; de manera que el estado de un objeto está definido por los valores de sus propiedades y la conducta se define por las operaciones (pueden tener parámetros de entrada y salida, y retornar o no un valor). Establece las relaciones que se pueden dar entre los objetos de herencia, asociación y composición; cómo los objetos pueden ser nombrados e identificados; y que tienen un identificador único no asociado al valor de sus atributos, sino que lo genera automáticamente el gestor de objetos [102]. Como conceptos importantes incluye, además, los de listas, colecciones, conjuntos y herencia múltiple (esto implica que no es necesario darle solución en el paso del refinamiento de las clases). Estas definiciones básicas se mantienen en la versión 3.0 [131]. En [130] se plantea que el modelo de objetos es el esquema de la BD.

En el método DIBAO las definiciones de clases, con sus comportamientos estático y dinámico, incluyen totalmente los conceptos de este modelo y van más allá. Esto último sugiere que con el gestor y el lenguaje de programación con que se trabaje tendrán que implementarse algunas características (por ejemplo, las fórmulas de evaluación, las restricciones asociadas a las relaciones entre clases).

En el análisis de cómo el modelo de persistencia se basa en el modelo O^3 (epígrafe 2.3), se describe cómo se obtienen los conceptos asociados a las propiedades y comportamiento de los objetos.

Este modelo de persistencia se inserta dentro de la etapa de diseño de la metodología ADOOSI 5.0 [7]. Con anterioridad al diseño de la base de datos, se han identificado las clases del dominio del problema con sus atributos y responsabilidades, usando como herramientas los diagramas de casos de uso y de interacción, y definiendo el modelo conceptual de acuerdo al objeto de automatización.

Otro punto importante a señalar es que ODMG desde su versión 1.2, incluye un lenguaje de manipulación que extiende LPOO para soportar el trabajo con objetos persistentes. Primeramente trabajó con C++, y desde la versión 2.0 incluye Java y Smalltalk. Estos lenguajes agregan clases y otras estructuras al ambiente para

apoyar el modelo de objetos de ODMG y mecanismos para construir preguntas usando el lenguaje de consulta de objetos. El LPOO Object Pascal (de amplio uso en el país), a través del ambiente de Borland Delphi, ha mejorado en cada versión en su trabajo con BD relacionales y objeto/relacionales (como SQL y Oracle versión 8.0), pero no con BDO y todavía el estándar ODMG no ha desarrollado un enlace con este lenguaje.

2.4.3 Conversión de clases a tablas.

El proceso de normalización centra su atención en los datos, pero con la OO se adiciona comportamiento. La autora de la presente tesis coincide con Ambler [8] cuando plantea que la normalización de una clase es un proceso durante el cual se organiza la conducta dentro de un modelo de la clase, buscando aumentar la cohesión de clases y minimizar el acoplamiento.

En esta tesis no se propone realizar los pasos de la normalización que se mencionaban en el epígrafe 1.2.2 porque:

- *La implementación de los atributos multivalores, compuestos y sus combinaciones, es una de las fortalezas del MOO. La propuesta de conversión que se presenta trata de no renunciar a esto y es por eso que lo definido en la primera forma normal (FN) no se tiene en cuenta exactamente. En el DC estos atributos se representan como parte de una clase compleja o compuesta.*
- *En la definición y refinamiento de las clases, desde la etapa de análisis, los atributos se obtuvieron a partir del comportamiento asociado a la clase que es necesario abstraer para una aplicación en particular, por lo tanto los atributos de una clase no deben tener dependencia incompleta. Si hay problemas con este aspecto, hay que revisar la definición pues la dependencia completa es una característica del mundo real que el MOO representa.*
- *Los problemas asociados a la tercera FN tampoco deben existir en la definición de las clases porque cuando una clase requiere conocer atributos de otra clase, define una referencia (un atributo del tipo de esa clase) y le envía mensajes a la clase solicitando la información que requiere.*

Como se aprecia, no es necesario en este módulo desarrollar los pasos incluidos dentro del proceso de normalización aún cuando se implemente en un medio relacional porque el proceso de obtención de las clases asociadas al problema a resolver, permiten obtener el modelo conceptual. En las clases están incluidas consideraciones físicas, como, por ejemplo, los atributos calculables.

Las llaves en el MR están asociadas a atributos de una tupla de una relación que caracterizan unívocamente a esa tupla. En ocasiones es difícil determinar un conjunto mínimo y suficiente de atributos para conformar un identificador. Por otra parte, no hay nada que impida a una aplicación modificar los valores de una columna identificada como llave, aunque hay SGBD que apoyan a esta restricción de integridad [102]. Por esto se recomienda asociar a las clases simples y complejas un atributo que constituya una llave de autoincremento (por ejemplo, SQL Server permite este tipo de atributos), sin significado semántico en la aplicación. Esto no excluye la posibilidad de definir las llaves tradicionales, en cuyo caso en el DC hay que identificar los atributos llaves indicando si es <<llave

primaria>> o <<llave extranjera>>. La llave de las clases compuestas, por defecto, es la suma de las llaves de las clases que la integran.

Para definir las reglas que permitan convertir los objetos, a tablas se tomaron en consideración las recomendaciones descritas en [13][50][82][91][98][114][136][154]; definiéndose en esta tesis las siguientes reglas:

1. *Cada clase con dominios simples se convierte en una tabla.*
2. *En el caso de la jerarquía de clase, se pueden aplicar tres posibilidades. Estas posibilidades son:*
   a. *Definir una única tabla con los atributos del padre y los atributos de cada hija.*
   b. *Definir una tabla para cada clase hija hoja del árbol de jerarquía, que incluya los atributos de la clase padre.*
   c. *Definir una tabla para cada clase hija y para cada clase padre. Para recuperar la información de una clase hay que consultar la clase hija y sus ancestros.*

      Esta tesis recomienda la última variante porque está más preparada para la evolución del esquema ya que en las variantes a y b las tablas creadas se modifican si ocurre algún cambio al no implementar el concepto de herencia. Además las tablas tendrían atributos que tienen sentido solo para algunos hijos, lo que implica permitir el valor nulo y por lo tanto los programadores tendrían que definir los procedimientos para validar estas restricciones de dominio. Las propuestas a y b son superiores en la rapidez en la recuperación de la información.
3. *Para clases complejas la conversión depende de las cardinalidades máximas en cada extremo. Las cardinalidades mínimas introducen restricciones de integridad. Las reglas que se aplican en estos casos son:*

m: n	*Se convierte en una tabla que tiene como llave a las llaves de las clases que conforman la relación. Otra variante es transformarla en dos relaciones bidireccionales en cuyo caso se añade un atributo de referencia a cada tabla. Esta variante hace más complejas las búsquedas y actualizaciones.*
1: m o m: 1	*Se puede convertir en una nueva tabla o puede ser añadida una llave extranjera al extremo m, que se corresponde con la llave de la clase del extremo 1.*
1:1	*Se puede convertir en una nueva relación o se adiciona un nuevo campo a alguna de las tablas que es la llave de la otra.*
c: m o m: c	*En el caso de que en uno de los extremos exista una cantidad mayor que uno de elementos con los que se relaciona la clase del extremo m, se puede definir una nueva tabla que tenga, como atributos llave, la llave del extremo m y tantos atributos del tipo de la llave del extremo constante como indique le valor de la constante. También se pueden adicionar a la clase del extremo m, tantos campos del tipo de la llave del extremo constante como indique el valor de la constante.*

c:c 1 *c y c1 son valores constantes conocidos. Se puede definir una nueva tabla que tendrá c+c1 llaves, en la que habrá c campos del tipo de la llave del primer extremo, y c1 campos del tipo de la llave del otro extremo. También se puede adicionar a la clase de uno de los extremos, tantos campos del tipo de la llave del otro extremo como indique el valor de la constante.*

En el caso de las relaciones n-arias (n>2), se crea una tabla con las llaves de las clases que intervienen y las cardinalidades definen restricciones de integridad.

A pesar de que no es necesario crear nuevas tablas que reflejen la relación en casi ningún caso, esta tesis recomienda hacerlo porque el esquema puede evolucionar y no sería en estos casos necesario hacer cambios a la estructura de la BD, solo a las restricciones de integridad estática asociadas a la cardinalidad. Además hay menos complejidad en las búsquedas y en la actualización de la información, y se conserva la propiedad de encapsulamiento, clave en el enfoque orientado a objetos, ya que un objeto no debe conocer cosas de otro objeto si no es necesario.

4. *Para las clases compuestas se crea una nueva tabla con el nombre de la clase compuesta y que tiene como llave a las llaves de las clases componentes. Las cardinalidades se definen como restricciones de integridad estática. En esta conversión no se tienen en cuanta las cardinalidades, pues se trata de llevar al esquema de la BD el objeto del mundo real tal como se aprecia, aunque esto introduce validaciones relativas a su cardinalidad. Además hay que tener en cuenta su dimensión con respecto cada componente.*

En el caso de que la implementación sea hacia un SGBDOR, que permita tener campos que referencien a otro objeto, no es necesario que se cambie el tipo de este campo para el tipo de la llave de la clase a la que pertenece el objeto referenciado.

La implementación de las características asociadas al comportamiento estático dependen de las potencialidades del gestor que se utilice. Por ejemplo, **SQL Server** incluye el predicado **check** que permite especificar restricciones de dominio de los atributos [52]. En el caso de no existir predicados para indicarlas, se deben utilizar disparadores que se activen cuando se intenta modificar o dar valor a los atributos involucrados en la restricción.

En el caso del comportamiento dinámico se pueden implementar como procedimientos almacenados las validaciones asociadas a las fórmulas de evaluación, las acciones que se ejecutan como parte de los mecanismos de disparo y las actividades asociadas a los estados.

En [18] se plantea que no es conveniente usar procedimientos almacenados y disparadores porque se escriben en un código, que puede cambiar en las versiones del gestor y entre vendedores, y se incrementa el acoplamiento en la BD porque se accede directamente a las tablas, lo que reduce la flexibilidad de la BD. No obstante, cuando no se posee un lenguaje de alto nivel que permita implementar estos algoritmos, el uso de los procedimientos almacenados y disparadores es una variante factible y rápida.

En el epígrafe 2.5.2 se describe la capa de persistencia en la que se definen estos procedimientos en un lenguaje OO. Esta variante permite localizar en un único lugar el código que se necesita reescribir si hay cambios, y está más cerca del enfoque

porque se conserva el concepto de clases para todo aquello que no sean datos almacenados.

2.5 Capa persistente.

Cuando se trabaja con una BDO, internamente se implementan los conceptos del enfoque OO y las capacidades de las BD, por lo que conceptos como los relativos a la herencia, por ejemplo, son tratados automáticamente sin que intervengan los programadores.. En las variantes de utilizar como medio de almacenamiento los ficheros o un LPOO que permita relacionarse con una BDR o una BDOR, será responsabilidad de diseñadores y programadores definir las clases que permitan manejar los conceptos de objeto y otros relacionados con las relaciones entre clases, que no están incluidos dentro de los modelos relacional y objeto/relacional.

La posibilidad de especificar clases permite la definición de una capa de clases persistente que encapsule el trabajo con los datos almacenados. Tradicionalmente esta capa solo contiene las definiciones necesarias para implementar la interfaz entre el medio de almacenamiento y las clases del dominio [17][119]. El método DIBAO propone incluir una subcapa de especificación que describa las características de las relaciones entre los objetos en correspondencia con lo definido en el DC. Además modifica la estructura de las clases persistentes adicionando métodos que recogen las restricciones de integridad y las fórmulas dinámicas antes descritas (figura 2.4) [94].

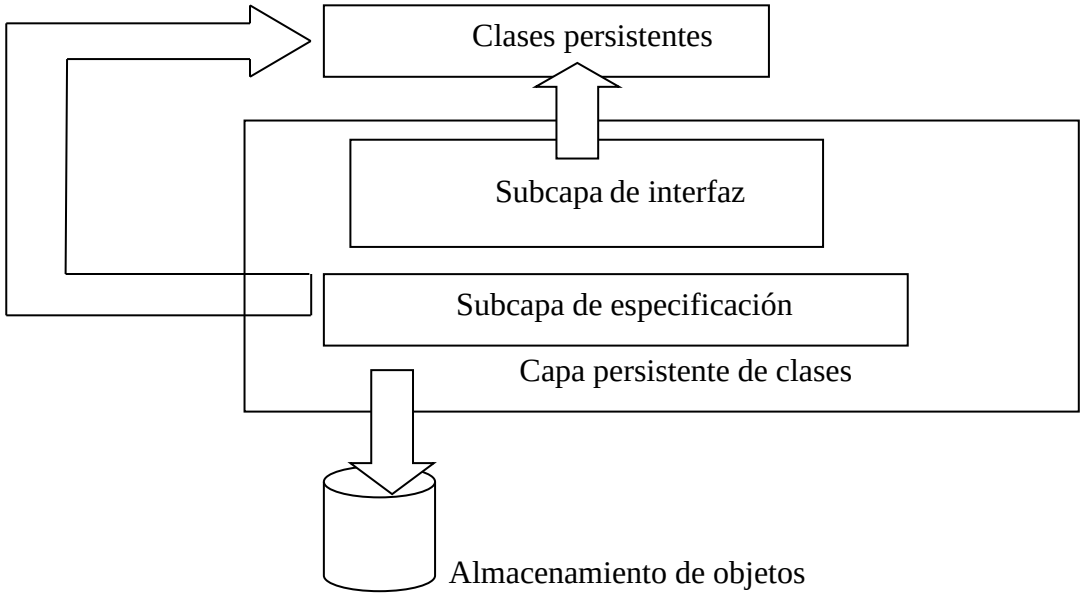


Figura 2.4 Estructura de la capa persistente.

A continuación se describe la propuesta de la autora, sobre cómo modificar la estructura de las clases definidas para una aplicación, para garantizar que se implementen los comportamientos estático y dinámico, y la creación de nuevas clases para el trabajo con la información almacenada.

2.5.1 Modificación de la estructura de las clases persistentes.

Al aplicar el modelo de persistencia del método DIBAO, se obtienen restricciones de dominio y fórmulas dinámicas, por lo que se necesita añadir a las clases persistentes:

- Para cada atributo, independientemente de su clasificación, un método que verifique sus restricciones de dominio.
- Un método para cada uno de los algoritmos que le asignan un valor en el caso de que sea un atributo dinámico.
- Un método que implemente la fórmula de derivación para los atributos derivados.
- Para el caso de los disparadores, se definen nuevos métodos (invocados por los métodos de creación, eliminación o actualización de un objeto, en dependencia del evento que lo active), que verificarán la condición, en caso de existir, e implementarán las acciones.

A través de la herencia, redefiniendo métodos o incluyendo nuevos métodos o clases según corresponda, se pueden implementar cambios en el esquema.

2.5.2 Subcapa de especificación.

En la fase de diseño de la BD, el método DIBAO ha adicionado características a las clases y sus relaciones que, aunque están relacionadas con el comportamiento de los objetos en el mundo real (por ejemplo, la dimensión entre compuesta y componente), no son propiedades semánticas como el color de los ojos o del cabello. La definición de nuevos atributos tiene aparejada la creación de métodos que los actualicen y garanticen las restricciones a ellos asociadas. En su implementación se puede seguir dos formas diferentes:

1. Modificar las clases definidas.
2. Crear nuevas clases que contengan las nuevas restricciones.

Optar por la primera variante hace más sencillos los procesos de validación de restricciones y de almacenamiento/recuperación de la información, al tener todo en una misma estructura, pero esta es también su principal desventaja porque un cambio en el dominio de estos atributos implica un cambio en la clase al modificarse el algoritmo de validación. Por ejemplo, si la dimensión de la relación entre las clases *Reservación* y *Habitación ocupada* cambia de dinámica a estática, no se pueden permitir modificaciones de ese atributo. Si la opción adoptada es la segunda, un cambio también cambiaría el algoritmo, pero solo se afectarían estas clases nuevas.

Obviamente, la definición de una nueva relación padre/hijo o una asociación, implica en cualquier variante modificaciones, pero aquí es donde se aprovecha una de las principales ventajas del enfoque OO en el mantenimiento de un sistema, que a través de la herencia permite añadir nuevas especificaciones sin cambiar las anteriores.

Teniendo en cuenta lo anterior, la subcapa de especificación opta por la segunda variante. Como los nuevos atributos y métodos que pudieran aparecer son

características del objeto que deben persistir junto al resto de las propiedades, se recomienda en esta subcapa, extender las estructura de las clases a través de la herencia, de forma tal que se incluya:

- Definir la clase THerencia con la siguiente especificación:

♦ Herencia de la clase que el lenguaje defina como persistente.

♦ Atributos:

Atributo	Tipo	Descripción
Total	Lógico	Falso- indica que al menos un objeto del tipo de la clase padre no se corresponden con alguno de sus hijos.
Solapamiento	Lógico	Falso- indica que un objeto del tipo de la clase padre solo se corresponde con un objeto en alguno de sus hijos.
Hija	Lógico	Verdadero- indica que tiene ancestros en la jerarquía del dominio.
Atributo de especialización	Clase	Clase que contiene el atributo que provoca la especialización y el valor que toma para esta hija en particular.
Árbol de herencia	Clase base del árbol de jerarquía del lenguaje	Colección de clases hijas en el árbol de jerarquía.

♦ Redefinir los métodos de salvar y restaurar de acuerdo a sus nuevos atributos.

♦ Crear los métodos que permitan crear y destruir una instancia de este tipo y le dan valor y consultan a sus atributos.

- Para todas las clases o un ancestro en su árbol de jerarquía, heredar de la clase THerencia (definida por el programador en el lenguaje que utilice). Esta clase recoge las características de una relación padre/hija lo que convertiría a todas las clases por defecto en posibles padres.

- Definir una nueva clase para cada una de las clases que tengan atributos que referencien a otros objetos. En todos los casos, por cada clase referenciada, se definen cuatro atributos que indican las cardinalidades máxima y mínima en cada extremo. En el caso de las clases compuestas hay que adicionar por cada componente un atributo de tipo lógico que indique la dimensión (Falso–Dinámica, Verdadero- Estática). Esto implica que hay que definir métodos que verifiquen que se cumplan las restricciones asociadas a los valores de estos atributos.

2.5.3 Subcapa de interfaz.

Ambler Scott en [17] describe diferentes formas de implementar una capa de clases persistente para el trabajo con la información almacenada en una BDR. En este trabajo se utilizan el código embebido y nuevas clases.

En la variante de implementación hacia ficheros, se incluye en la especificación de las clases, los métodos para salvar y recuperar datos. En este caso, la capa de clases está embebida dentro de las clases del dominio, lo que permite escribir código rápidamente (útil para aplicaciones pequeñas y prototipos), pero que implica un fuerte acoplamiento entre las clases del dominio y el medio de almacenamiento, por lo que un cambio en estos últimos afecta a los métodos de la clase. La forma en que los lenguajes de programación implementan esta forma de almacenamiento, obliga a utilizar esta variante a pesar de sus problemas.

Cuando se trabaja con un LPOO, yla información se almacena en un BDR o BDOR, es posible encapsular el trabajo con el medio de almacenamiento en una estructura intermedia que sea la única afectada si cambia el gestor o la estructura de las tablas en que se almacena la información de un objeto. Evidentemente esta solución hace más compleja la construcción de aplicaciones, pero crea una disciplina que mejora la calidad del producto final, facilita el mantenimiento y permite el desarrollo de aplicaciones más complejas.

La idea consiste en:

- Definir la clase TPersistenciaBD que incluya:

♦ Atributos:

Atributo	**Tipo**	**Descripción**
TablaBase	Clase base del árbol de jerarquía del lenguaje	Nombre de la tabla a la que directamente se transforma la clase.
Lista de tablas asociadas	Clase base del árbol de jerarquía del lenguaje	Listado que contiene las tablas asociadas a la clase base y fue generado a partir de las conversiones descritas en el epígrafe 2.343

♦ Métodos para salvar, borrar, recuperar y modificar objetos, que se redefinen en cada clase hija de acuerdo a las tablas asociadas a cada clase. Su implementación mezcla sentencias del lenguaje de programación y del lenguaje de consulta. Estos métodos son los únicos que se afectarían por un cambio en el esquema de la BD o el gestor utilizado.

- Incluir en las clases persistentes una referencia a la clase asociada de la capa persistente porque, los métodos que necesitan de los valores de los atributos del objeto, solo tendrán acceso a ellos a través de esta capa de clases.
- Definir una nueva clase para cada clase persistente que herede de TPersistenciaBD.

2.6 Comprobación de la calidad del diseño de la BD.

La autora de esta tesis plantea que la calidad de la definición de las clases está subordinada al cumplimiento de las directivas que se establecen en la metodología para lograr un buen diseño de las clases, y de ella depende en gran medida la calidad del diseño de la BD [93].

Durante el proceso de análisis, la metodología ADOOSI brinda una serie de recomendaciones que permiten analizar el flujo de control entre las clases. Una definición de clases adecuada tiene en cuenta lo siguiente [4]:

1. Una clase no debe relacionarse con más de 7±2 clases para evitar excesivo centralismo de control.
2. Un subordinado no debe tener más de un jefe, por lo que una clase debe tener un sólo jefe aunque se reconocen algunas excepciones.
3. *Una clase debe tener acceso solo a los atributos que necesite para llevar a cabo sus responsabilidades y solo a ellos.*
4. *Una clase debe clasificarse en una sola categoría (manejadora de datos o estados, pozos o fuentes de datos, interfaz, auxiliares o de ayuda y controladoras).*

En la evaluación de la calidad del proceso de análisis y del refinamiento asociado al diseño, se analizan varios atributos de calidad de acuerdo a la apreciación que se haga de un conjunto de factores y métricas de calidad. De los atributos definidos en [75], en esta tesis se concluye que los que hay que tener en cuenta en la valoración del nivel de aceptación de las clases son:

1. Completitud de las clases*: Es el grado de especificación total (sin omisión) de los requerimientos para la definición de la clase.*
2. Consistencia de las clases*: Es el grado en la que la definición de la clase es única y no contradictoria respecto a los requerimientos de una entidad, capaz de cumplir las responsabilidades asignadas y tener los atributos necesarios para ello, sin que exista ninguna contradicción en la forma de lograrlo.*
3. Cohesión*: Es el grado de cohesión simple entre las clases y objetos, es decir, que los atributos de una clase trabajan justamente a la vez para proveer un buen comportamiento.*
4. Encapsulamiento*: Es el grado en que la información apropiada está adecuadamente encapsulada dentro del tipo correcto de entidad, para disminuir el acoplamiento y aumentar la cohesión.*
5. Visibilidad*: Es el grado en que una clase puede invocar métodos de otra. Se dice que una clase B es visible a una clase A, cuando A invoca métodos de la clase B.*
6. Consistencia de las clases abstractas y concretas*: Es el grado en que la definición de clase abstracta y concreta es única y no contradictoria respecto a los requerimientos de una asociación de analogía o a las asociaciones de generalización/especialización.*
7. Complejidad de cada clase: *Es el grado de comprensión de la conducta de la abstracción, en la que se tienen en cuenta los métodos que invoca, la cantidad de métodos y atributos, y la profundidad y amplitud del árbol de herencia.*
8. Tamaño de la clase: *Es la magnitud de la clase. El tamaño de la clase indica la cuantía de la colaboración. Las clases grandes son más complejas y difíciles de mantener. Las*

clases pequeñas tienden a ser más reusables ya que proporcionan un conjunto de servicios cohesivos.

9. Completitud de las responsabilidades y atributos de una clase*: Es el grado en que se ha logrado definir de forma clara y concisa todo lo que realmente debe hacer una clase, así como la especificación total (sin omisión) de los atributos necesarios para ello.*

10. Tamaño de la responsabilidad de una clase*: Es la cuantía de una responsabilidad dada por el número de mensajes que envía, cantidad de líneas de código y cantidad de sentencias.*

11. Complejidad de la responsabilidad de una clase*: Es el grado de comprensión de la conducta de la abstracción.*

La evaluación de estos atributos se realiza a través de métricas de calidad [74], pero no es objetivo de este trabajo adentrarse en este campo. Lo importante es conocer cuáles aspectos son necesarios para verificar la calidad de las definiciones obtenidas, tomando en cuenta el dominio del análisis y las particularidades del enfoque OO.

Que el esquema de la BD cumpla la cualidad de la compleción (representación de todas las características del dominio de la aplicación), es algo que se verifica desde los primeros pasos del desarrollo. En la encuesta incluida en [4] los usuarios de ADOOSI evaluaron con un alto grado de aceptación las posibilidades de la metodología para la especificación de los requerimientos. La inclusión de los diagramas de caso de uso y la definición del modelo conceptual en la etapa de análisis en la versión actual [7], contribuyen a esto.

Con respecto a la estructura de las tablas, tampoco es necesario definir nuevas métricas porque se tienen en cuenta cuando se evalúan los atributos relativos al refinamiento de clases, responsabilidades, atributos y colaboraciones entre clases [75]. Estos atributos van más allá de la forma en que se describe el comportamiento a las clases asociadas.

Una encuesta aplicada a expertos en diseño de la BD (anexo 3), refleja cómo el método DIBAO afecta a la calidad del software, con respecto a la BD obtenida. Se tomó como referencia los factores de calidad para este tipo de productos definidos por las normas ISO. Todos los factores fueron evaluados entre 4 y 5, excepto el de facilidad de uso para expertos, con experiencia en el diseño de BD (13 años como promedio), pero ninguna con el MOO. Esta situación es lógica si se considera que no es fácil desarrollar un software usando la tecnología objeto [16], sobre todo si se trabaja con otros enfoques.

2.7 Experiencias de su utilización.

La propuesta de método de diseño de la BD que se describe en este trabajo se incluye dentro del curso de Diseño de Sistemas Informáticos, que reciben los estudiantes de 4to año de la carrera de Ingeniería Informática, desde el curso 1999-2000. Además desde ese mismo período forma parte de los cursos de la metodología ADOOSI que se han impartido a empresas del país. La metodología ADOOSI 5.o se adoptó en todas las unidades del MININT y con ella el método DIBAO.

La versión actual del método se ha aplicado en proyectos que han sido fundamentalmente sistemas de gestión implementados usando Borland Delphi para el trabajo con BDR (16) y de propósito general (7) para lo cual se ha utilizado la persistencia hacia ficheros.

Una de las aplicaciones más complejas en la que se está utilizando este método es el *Software para la información, documentación y mejoramiento genético del germoplasma de papa",* desarrollado por los departamentos de Matemática y Genética y Mejoramiento del INCA (Instituto Nacional de Ciencias Agrícolas). En el anexo 4 se muestran fragmentos de los resultados de la aplicación del método DIBAO, que reflejan la complejidad del sistema. En este caso la variante seleccionada ha sido la del almacenamiento de la información en un gestor relacional, aunque las clases persistentes definen un método para salvar su información hacia un fichero ASCII pues es un requerimiento del sistema.

Actualmente también se incluye este método de diseño dentro del curso de análisis y diseño que reciben los estudiantes de la carrera de Ingeniería de Sistemas de la Universidad Peruana de Ciencias Aplicadas (Lima, Perú) y en la carrera de Sistemas Automatizados de Dirección del Instituto Técnico Militar José Martí (Cuba).

2.8 Conclusiones.

Estática y dinámica son dos perspectivas importantes que se han de tener en cuenta cuando se diseña la BD. Por un lado se busca definir estructura y por el otro cambios permitidos sobre esa estructura. Obviar alguna de estas vistas en el proceso de diseño puede llevar a una BD inconsistente.

En esta propuesta de método de diseño de la BD se incluyeron los conceptos de las definiciones estática y dinámica referidos por [49][118][136]. Se usó como referencia a estos autores porque recogen en sus criterios tópicos abordados por otros también de reconocimiento en este campo.

La idea de acercar ADOOSI a UML no es casual. UML es desde finales de 1997 la notación estándar para la OO aprobada por el OMG (Object Management Group), por lo que su utilización en la metodología es lógica. Además posee una alta capacidad expresiva.

Partir, como premisa para el diseño de la BD, de la clasificación de sus atributos, permite minimizar este proceso al centrarse el diseñador en lo que puede afectar al atributo y lo que debe saber de cada uno.

El método de diseño sistematiza los pasos a seguir en el diseño así como recomendaciones para llevarlos a cabo. El método DIBAO describe cómo reflejar las características estáticas y dinámicas utilizando los diagramas de clase y transición de estado y formatos de especificación de algunas restricciones de integridad y fórmulas dinámicas. Esto lo hace superior a otras propuestas que, si mencionaban algo, por lo general estaba relacionado solamente con las reglas de conversión de clases a tablas, y el uso de diagramas para especificar la estructura de las clases.

La conversión de clases a tablas no desecha por completo todas las características de este enfoque OO ya que:

- Encapsula el trabajo con las tablas con la creación de clases persistentes que actúan de interfaz en su relación con otras clases que requieren de la información almacenada.
- Recomienda la creación de nuevas clases con vistas a futuros cambios en el esquema.
- Representa la agregación como una tabla, independientemente de la cardinalidad, para conservar el objeto del mundo real con su significado.

- Recomienda siempre como tendencia la implementación del código utilizando las potencialidades del enfoque OO.

  El modelo de persistencia satisface las características definidas por ODMG y, además, si se le añade el comportamiento dinámico, entonces se puede decir que también cumple lo referente al marco formal definido en O^3.

  El modelo de persistencia tiene en cuenta las variantes, en cuanto a forma de almacenar la información, con que se pueden enfrentar los diseñadores. En esto es superior a lo planteado por otros autores.

  La capa de persistencia incluye dos subcapas que permiten la definición de nuevas clases que responden al modelo de persistencia y asilan a las clases del dominio del medio de almacenamiento. En la bibliografía solo se asocia a esta capa lo relacionado con la interfaz. Además en esta tesis se le da nombre a estas dos subcapas.

  En el caso de las clases persistentes, se propone adicionar nuevos procedimientos asociados a las validaciones de las restricciones estáticas y fórmulas dinámicas.

  Finalmente se mencionan algunos aspectos que influyen en la calidad del diseño de la BD, sobre todo con la calidad de la calidad de las clases obtenidas, pues los criterios asociados al proceso de diseño en sí, se han fundamentado en el lugar donde se explicaron.

CAPÍTULO 3
Herramientas CASE

CAPÍTULO 3 HERRAMIENTAS CASE

La utilización de herramientas CASE (Computer-Aided Software Engineering) ha sido uno de los aportes más importante en el campo de la Ingeniería de software. Aunque el concepto fue abordado desde hace tres décadas, son los productos que se encuentran hoy en el mercado los que han provocado una revolución por el aumento que propician en la eficiencia, productividad y calidad del producto final.

Las herramientas CASE son una tecnología para automatizar el desarrollo y mantenimiento del software, combinando herramientas de software y metodologías. Estas herramientas deben constituir un conjunto integrado que automatice todas las partes del ciclo de vida y por tanto ahorren trabajo [125].

Las facilidades que brindan para revisar especificaciones de un sistema, diagnosticar errores cometidos, desarrollar prototipos al generar parte o completamente una aplicación a partir de especificaciones, y el soporte para el mantenimiento como resultado de haber guardado las especificaciones del sistema en un depósito central de datos [64][85][159]; son razones suficientes que justifican la importancia del desarrollo de estas herramientas acompañadas de las metodologías y métodos.

A partir de especificaciones de diseño, las herramientas generadoras de código pueden generar un esquema o un programa completo [125].

En este capítulo se describe la propuesta de CASE que permite el diseño y la generación del esquema de la BD, de acuerdo al método DIBAO descrito en el capítulo anterior.

3.1 Concepción general.

El término CASE integrado se refiere a la idea de utilizar un diccionario de datos centralizado alrededor del cual giran herramientas que permiten automatizar el ciclo de vida completo. Las herramientas que se presentan, están insertadas dentro de la concepción de un CASE integrado más abarcador que se desarrolla para la metodología ADOOSI, y agrupa a las herramientas que se desarrollan para el diseño y generación de la BD.

En la actualidad se trata de dos herramientas, que no dependen una de otra, pero que guardan relación con las herramientas predecesoras. En el próximo subepígrafe se explicará la forma en que intercambian información. En el epígrafe 3.1.2 se describen los módulos en los que se agrupan los pasos definidos para el diseño de la BD.

La propuesta en un futuro debe brindar una sola herramienta que permita generar hacia cualquier medio de almacenamiento. No obstante no ha sido hasta ahora una prioridad los SGBDO por la falta de productos maduros y la escasez de conocimiento sobre ellos en el país, pero la inclusión del estándar ODMG en algunos de estos gestores, sugiere que es una variante a considerar en un futuro cercano.

Podría pensarse que, como la propuesta presentada usa la notación UML y existen CASE como Rational Rose (de amplia divulgación) que se basan en esta metodología, no se justifica la realización de estas herramientas. Hay dos razones importantes que dan respuesta a esta interrogante. En primer lugar, Rational Rose genera la estructura de la BD a partir del DC y, en el modelo de persistencia propuesto por la autora, se han incluido nuevas especificaciones que no están consideradas en la generación que realice Rational Rose. En segundo lugar. Rational Rose no tiene en cuenta la perspectiva dinámica en el diseño de la BD.

Por otra parte, la estructura de los ficheros que genera esta herramienta no se conoce, lo que limita las posibilidades reales de utilizar, como fuente de entrada del CASE, los resultados obtenidos con el uso de este software. No obstante, es una variante acertada tratar de descifrar las salidas de Rational Rose para utilizarlas en las herramientas propuestas.

En [22] se describen dos formas de modelación de los objetos, para su almacenamiento en un medio persistente. En Rational Rose se implementa la variante de transformar el modelo de objetos a instrucciones de SQL que convierten en tablas a las clases. El CASE Rose DataModeler [147] es uno de los últimos productos de Rational, y se anuncia como una herramienta que permite capturar restricciones, disparadores índices y que puede trabajar con una SGBDR y un SGBDO.

Aunque muchas personas usan las herramientas CASE como herramientas de documentación, los productos que se describen permiten, además, generar información automáticamente, ayudan en la realización de diagramas, validan inconsistencias, trabajan con rapidez, y presentan una interfaz agradable.

3.1.1 Integración con otras herramientas.

Antes de que un diseñador proceda a diseñar la BD, es necesario que un analista haya estudiado el dominio del análisis para interpretar los requerimientos del usuario y traducirlo en las clases, con los atributos y comportamiento, que son de interés para esa aplicación. El método de diseño propuesto parte de este conocimiento, aunque no descarta la posibilidad de adicionar o eliminar algún comportamiento.

Las herramientas, permiten al diseñador introducir toda esta información manualmente o utilizar la información generada por otra herramienta automatizada. En particular, identifica los ficheros generados por las herramientas CASE: Analyze, Designer y GenProo; utilizadas por los usuarios de la metodología ADOOSI.

Analyze [149] automatiza parte de la etapa de estudio preliminar y el análisis de esta metodología. Designer [81][83] toma estas especificaciones y facilita el desarrollo de casi todas las fases de la etapa de diseño, a excepción del diseño de la BD. Estas dos herramientas se corresponden con versiones anteriores de la metodología y no incluyen ningún tipo de generación de código. GenProo [124] permite obtener prototipos del sistema generando automáticamente parte de la interfaz y el código asociado a la interfaz de las clases del dominio. En todos los casos se genera la documentación asociada a las fases que automatizan y se almacena la información en ficheros con un formato definido, una extensión y marcas de chequeo que los identifican como resultados de estos softwares (todos usan la persistencia a fichero).

En la versión actual de la metodología, se recomienda usar Rational Rose porque es una herramienta automatizada que brinda altas prestaciones de servicio e incluye los diagramas que utiliza ADOOSI 5.0. Este CASE, al no soportar una metodología en particular, no realiza algunos chequeos asociados a una lógica de desarrollo del software. En la actualidad se trabaja en la concepción y desarrollo de otras herramientas que respondan a la metodología y aborden aspectos en los cuáles hay algunas deficiencias.

El CASE integrado toma la información contenida en los diccionarios de datos, y automatiza el método de diseño de la BD. La forma de integración utilizada en este caso es la denominada por Roger Pressman [144] como intercambio dinámico de datos. La aplicación

cuenta con un traductor que convierte la información del diccionario de datos ya que requiere añadir nuevas especificaciones imprescindibles como parte del diseño (Figura 3.1).

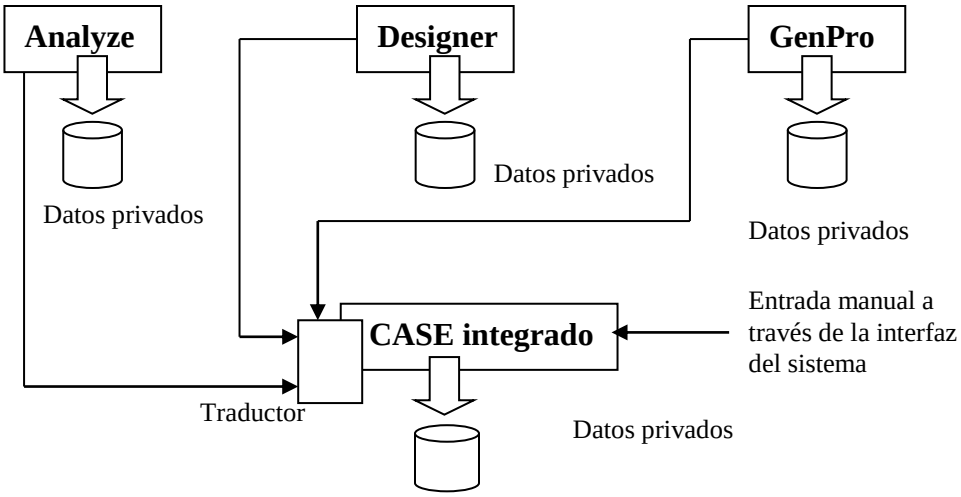


Figura 3.1 Intercambio de información entre herramientas.

Como el CASE integrado está diseñado para un método de diseño en particular (DIBAO), soporta las tres filosofías descritas en [174]. Es restrictivo pues hay un conjunto de acciones que están restringidas (por ejemplo, no permite que clases controladores sean persistentes); guía al usuario ya que hay acciones que, aunque no se restringen, se le alerta de las posibles consecuencias (por ejemplo, cuando se intenta eliminar una clase, dejando al usuario la libertad de decidir continuar el proceso, implementándose las acciones en cascada que genera), y es flexible porque el usuario puede realizar algunas acciones con completa libertad sin seguir estrictamente el orden de los pasos definidos por el método (por ejemplo, el refinamiento de la clases y la clasificación de clases y atributos, puede ser hecho en cualquier orden.

3.1.2 Módulos.

En esta tesis se propone desarrollar los pasos del diseño y la implementación de la capa persistente, dividiendo el proceso en 4 módulos (Figura 3.2). Se define otro módulo para la generación de las clases no persistentes

Estos módulos se ejecutan secuencialmente, aunque se permiten retrocesos. La ejecución secuencial implica que cada módulo utiliza la información obtenida por el módulo predecesor y genera automáticamente información. Un cambio en la información precedente, es interpretada como un cambio en la especificación por lo que se vuelve a generar.

En el proceso de automatización no se siguió la variante de implementar un módulo completo antes de pasar al siguiente, sino que, en dependencia del objetivo de cada ciclo (descritas en el epígrafe 3.2), se automatizó algo de cada módulo. Es decir, se siguió un desarrollo incremental, en el que cada ciclo constituye un ciclo de desarrollo completo.

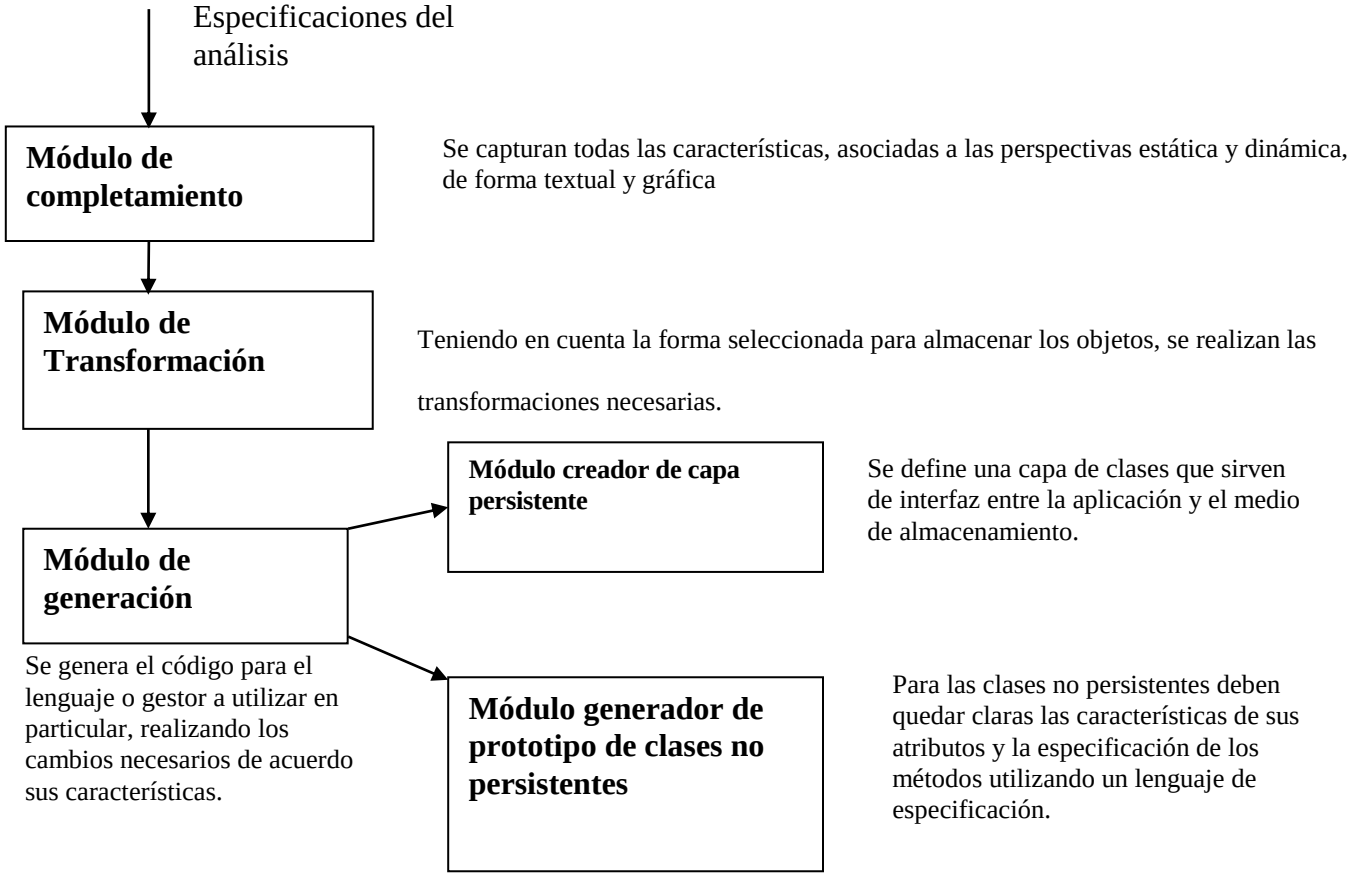


Figura 3.2 Módulos del proceso de diseño y generación.

3.2 Herramientas automatizadas.

Desarrollar una herramienta CASE que incluye parte de las etapas de diseño y desarrollo, generando el esquema de la BD y el comportamiento dinámico de sus objetos para todas las alternativas, no es un proceso sencillo. Podría realizarse todo en un solo paso, pero el desarrollo incremental permite obtener prototipos reales del sistema que permite ir obteniendo ventajas tempranas.

En la automatización del método DIBAO se definieron los siguientes ciclos:

1er Ciclo: Debido a la debilidad de los SGBDO presentes en el mercado, el incremento en el país del desarrollo de aplicaciones que utilizaban la metodología ADOOSI, las características de las aplicaciones que se desarrollaban utilizando este enfoque de programación que no requerían grandes prestaciones en cuanto a la gestión de los datos, las limitaciones de los LPOO para el trabajo con SGBD y, por lo tanto, el uso de la persistencia a ficheros como la mejor variante para el almacenamiento de la información; se decidió desarrollar como primera herramienta aquella que manejara este tipo de persistencia. Como resultado se obtuvo en Marzo de 1998 Gecodel 1.0, como parte de la tesis de ingeniería “Generación de código para Borland Delphi a partir de especificaciones de diseño orientado a objeto” [24].

2do Ciclo: Desarrollar una herramienta que permita la generación de la base de datos para el trabajo con un SGBDR, dado las posibilidades de Borland Delphi para el trabajo con estos gestores. Como resultado se ha obtenido Gebase 1.0, que fue esbozado en la tesis de ingeniería “Generación de la base de datos” [2].

3er Ciclo: Integración de todas las herramientas en un único producto. Se trabaja también en una tesis de maestría en el desarrollo de un lenguaje de especificación más formal que permita una generación más completa y fidedigna de los métodos asociados a las clases persistentes.

En los subepígrafes siguientes se describen las herramientas mencionadas y se hace referencia a la utilización de un algoritmo para el trazado automático del diagrama de clases a partir de las especificaciones de las clases.

3.2.1 Persistencia de los datos hacia ficheros.

GECODEL es un sistema que aborda la generación automática, para Borland Delphi, de las units con la estructura de clases y el tratamiento de la persistencia a ficheros.

El dominio del CASE comienza en la última etapa del diseño que es la definición de clases para el desarrollo. A partir de las especificaciones del archivo de diseño, el diseñador puede completar la información añadiendo elementos propios del lenguaje, para posteriormente generar las units con el código respectivo a la definición de las clases en Object Pascal. A partir del contexto en que se mueve el sistema se automatizaron las siguientes funciones:

- Obtener la información del Designer.
- Crear nuevos elementos para la generación de código.
- Almacenar y cargar ficheros con formato propio.
- Generar el código en las units respectivas, con los procedimientos necesarios para la persistencia a ficheros.
- Editar las units con el código generado, para lo cual se brinda un editor integrado con las estructuras básicas usadas en el lenguaje Object Pascal.

En la figura 3.3 se presenta un fragmento de una unit generada por el sistema. Los puntos suspensivos se corresponden con detalles de otras clases que no aportan algo nuevo a la comprensión del ejemplo. En el anexo 5 se muestran algunas pantallas de este software.

```
{*******************************}
{ Unit Personas generada por Gecodel 1.0}
{*******************************}
Unit Personas;
Interface
Uses …..
Const .....
Type .....
  { Clase: Persona }
  Persona = class (TPersistent)
    protected
      nombre : string;
  edad : integer;
      direccion : string;
      .....
    public
      .....
      procedure Actualizar_direccion
(Direc:string);
      procedure ReadData
(Reader:TReader);virtual;
  procedure WriteData (Writer:TWriter);virtual;
end;
  .....
Implementation ......
{----------Persona----------}
......

procedure Persona.Actualizar_direccion
                              (Direc:string);
begin
end;

begin
 inherited ReadData(Reader);
 with Reader do
 begin
   nombre:=ReadString;
   edad:=ReadInteger;
dirección:=ReadString;
 end;
end;
procedure Persona.WriteData
      (Writer:TWriter);virtual
begin
inherited WriteData(Writer);
 with Reader do
 begin
   WriteString(nombre);
   WriteInteger(edad);
   WriteString(dirección);
 end;
end;
.....
inicialization
.....
classes.Registerclass(Persona);
.....
end.
```

Figura 3.3 Fragmento de unit generada por Geco**d**el.

Los costos de un proyecto son afectados fundamentalmente por la fuerza de trabajo que interviene en su realización y los medios técnicos y materiales consumidos. La determinación de los costos de fuerza de trabajo y medios técnicos es directamente proporcional al tiempo de desarrollo.

Se estima que **Gecodel** genera el 20% del código aproximadamente por lo que disminuye el tiempo de desarrollo de la etapa de codificación. Esto, por transitividad, implica un descenso en forma proporcional del costo total del proyecto. La disminución en el tiempo de desarrollo se refleja principalmente en el hecho de que el sistema permite:

- Obtener las units con parte del código fuente, proceso que se realiza automáticamente sin la intervención del usuario. Si cambia alguna especificación, solo es necesario generar nuevamente el código, lo cual no implica un esfuerzo adicional del usuario.

- Obtener la documentación del proyecto, con las definiciones de las clases contenidas y todas sus características.

3.2.2 Persistencia a gestores relacionales.

GeBase es una herramienta que, a partir de algunas especificaciones correspondientes al análisis de un sistema informático, permite automatizar parte de la etapa de diseño través de la adición de información y la definición de los parámetros necesarios para generar la base de datos. Se encarga de:

- Establecer la relación con el fichero resultante del Analyze [149] y MetProo [124].
- Capturar las especificaciones necesarias visualmente y por pantallas de edición.
- Convertir las clases persistentes a tablas a partir de las reglas definidas en el método DIBAO para esta alternativa de almacenamiento.
- Generar el DC a partir de las características de la clases, utilizando un algoritmo eficiente de trazado automático [152].
- Chequear la calidad y consistencia de la información.
- Generar la documentación asociada a los procesos que automatiza.

Una herramienta CASE que genere la base de datos debe ser capaz de obtener y almacenar las especificaciones necesarias y crear un modelo que contenga las características de la información generada. Para este tipo de productos esta tesis define un conjunto de criterios que ayudan a evaluar su calidad, entre los que se encuentran:

- *Tiempo de generación:* **Gebase** la realiza instantáneamente después de activarse esta opción.
- *Complejidad de la presentación:* las propiedades y los campos de las tablas resultantes son mostradas de forma similar al formato utilizado por los gestores relacionales, en forma de tabla.
- *Número de tablas mostradas a la vez:* los sistemas de gestión, por la extensión que puede alcanzar una tabla de este tipo, solo muestran una cada vez y así está implementado en Gebase.
- *Posibilidad de modificación de la estructura de la tabla antes de generar:* el sistema parte del principio de que si se utiliza un generador, lo más eficiente es cambiar la especificación y volver a generar.

Es muy común que las metodologías incluyan herramientas gráficas. Debido a esto se han definido un conjunto de criterios que ayudan a evaluar su automatización [176], entre ellos se encuentran:

- ¿cuán rápido pueden los dibujos ser cambiados y redibujados?,
- ¿cuán rápido pueden los dibujos ser salvados y recuperados?,
- ¿cuántos diagramas diferentes pueden ser vistos en la pantalla al mismo tiempo?,
- si el documento soporta niveles de diagramas, ¿el usuario puede moverse de un nivel a otro?, ¿es fácil mostrar dos niveles al mismo tiempo?, y
- control del usuario sobre el tamaño, posición y color de los elementos que aparecen.

Estos aspectos son muy importantes para que el dibujo cumpla su cometido de describir con más facilidad los detalles. Para ello pueden utilizarse algoritmos que decidan, por ejemplo, cuántos niveles de representación se requieren (que se traduce en cantidad de vistas para un gráfico), cómo se organiza la información en dependencia del tipo de gráfico que sea, entre otros aspectos [83].

Algunas veces la forma de representar, lejos de ayudar a la comprensión, la hace muy difícil debido a la gran cantidad de líneas y puntos que la conforman. Aunque se han desarrollado

algoritmos para el trazado de diagramas para algunos diagramas como es el Diagrama Entidad Relación, en otros diagramas no se ha logrado esto [151].

Gebase, para el trazado automático del DC, usa el método del Escalador de colinas estocástico para la ubicación espacial de los elementos. El algoritmo utilizado fue el resultado de la tesis de doctorado que se describe en [1] y se implementa en una biblioteca de clases. **Gebase** utiliza la biblioteca en dos momentos, para generar el diagrama a partir de las clases contenidas en el diccionario de datos, y después para redibujar el diagrama ya que el diseñador puede modificarlo.

El algoritmo se adapta a los requerimientos del CASE porque permite cualquier tipo de nodos, las líneas (para unir los elementos) pueden ser rectas o quebradas, y tiene en cuenta los siguientes criterios estéticos: minimizar la cantidad de cruces entre arcos, minimizar la longitud total entre arcos, balancear la ubicación de los nodos respecto a determinados ejes, minimizar el solapamiento entre nodos y minimizar el área total ocupada. Este último criterio se utiliza además para introducir la semántica relacionada con la necesidad de que las clases compuestas puedan estar ubicadas espacialmente cerca de sus componentes.

A pesar de que no siempre se obtiene un dibujo óptimo con el trazado automático, se logra una solución factible si se considera que la complejidad de los diagramas que se automatizan puede ser alta. En los ejemplos de pantallas de **Gebase**, que se incluyen en el Anexo 6, se muestra la calidad visual que se obtiene con la aplicación de este algoritmo. La legibilidad en la representación gráfica forma parte de los criterios de calidad a tener en cuenta para este tipo de software.

Gebase se estima que ahorra entre un 20%-25% del tiempo de desarrollo de un proyecto informático, de acuerdo al volumen de clases persistentes (mientras mayor es la cantidad con respecto al total de clases, se obtiene un ahorro mayor). Constituye además una ayuda para el diseñador pues ya no tendrá que cambiar el código cada vez que desee cambiar la estructura de la BD, solo tiene que cambiar la especificación y volver a generar.

3.2.3 Implementación de la capa de persistencia.

En **Gecodel** se implementó la capa de persistencia asociada al trabajo con ficheros ya que como parte del código generado, se encuentran las instrucciones que permiten que:

- Todas las clases o un ancestro hereden de la clase del lenguaje TPersistent.
- Se redefinen los métodos ReadData (cargar) y WriteData (salvar) de acuerdo al tipo de los atributos de la clases (tal como se aprecia en la figura 3.3).
- Se registren las clases persistentes.

Actualmente se trabaja en la generación de la capa persistente para el trabajo con una BDR o una BDOR. Al igual que los otros CASE, se trabaja para la generación en el lenguaje Object Pascal, por las facilidades de Borland Delphi para el desarrollo rápido de aplicaciones, utilizar un LPOO, el amplio uso en el país, y las potencialidades para el trabajo con gestores relacionales.

Para el caso de que se implemente en un gestor relacional u objeto/relacional en el que no se posea un lenguaje de alto nivel que incluye los conceptos del MOO, no es posible generar la capa persistente. En su lugar se generan los procedimientos almacenados y disparadores que recojan las restricciones estáticas y fórmulas dinámicas definidas.

3.3 Conclusiones.

Las herramientas **Gebase** y **Gecodel** están basadas en un método de diseño en particular (DIBAO), lo que adiciona valor a los productos ya que brindan recomendaciones, chequean contradicciones y generan automáticamente información. Todo esto mejora la productividad al disminuir el tiempo de desarrollo. A este fin contribuye también el hecho de que se cambia el esquema, no es necesario modificar directamente la implementación, solo tiene que cambiar la especificación y volver a generar.

En este capítulo se presentó la propuesta de automatización que implementa el método propuesto en el capítulo 2. Los resultados forman parte de una CASE integrado mayor que se viene desarrollando desde hace varios años con vistas a dar un soporte automatizado a la metodología ADOOSI.

No se descartó el uso de herramientas comerciales que utilicen el estándar UML, como fuente de entrada de información para el CASE. Se trabaja en la actualidad en la interpretación de un fichero texto al que es posible exportar la información de un proyecto creado con Rational Rose. De esta forma Gebase se convertiría en una extensión que cubre un espacio en el que Rational Rose no es suficiente.

Se describen además las herramientas automatizadas que ya se poseen y los trabajos que se están desarrollando, siendo importante en un futuro cercano permitir la generación hacia SGBDO como Poet y Objectivity/DB que, como se comentaba en el capítulo 1, han logrado un grado alto de madurez.

La utilización del algoritmo del Escalador de colinas estocástico, mejora la calidad visual del DC y, por lo tanto, permite que sea entendida mejor la información en él representada. Además del uso de este algoritmo, se han definido un conjunto de criterios que contribuyen a valorar la calidad de los productos obtenidos.

CONCLUSIONES

Las principales conclusiones del estudio realizado sobre el tema son:

1. Del estudio bibliográfico realizado se puede deducir que no existe una tendencia a la formalización del MOO. No obstante los trabajos de ODMG en la definición de un estándar para SGBDO y el modelo formal O^3, son marcos formales adecuados para sustentar el método de diseño de la BD a partir del MOO (DIBAO) que se propone en esta tesis.

2. Del estudio comparativo realizado de varias propuestas metodológicas, tomando como referencia algunas magnitudes fijadas para determinar el grado de completitud y calidad de sus modelos de persistencia, se concluye que la tendencia más explotada por las metodologías y métodos de diseño de la BD, es la conversión de las clases a tablas. Sus deficiencias en cuanto a la definición de los pasos para el diseño de la BD, la escasez de guías y criterios que ayuden a obtener los comportamientos estático y dinámico, y la forma insuficiente en que son tratados estos comportamientos; hacen necesario un modelo de persistencia más completo.

3. La utilización de la notación UML es acertada si se tiene en cuenta su aceptación como estándar desde 1997, su amplia difusión en los últimos años y su alta capacidad expresiva.

En cuanto a la propuesta de diseño de la BD descrita, se puede concluir que:

4. El modelo de persistencia propuesto tiene en cuenta todas las posibles alternativas actuales (ficheros, SGBDO, y SGBDR o SGBDOR utilizando o no un LPOO), en cuanto a formas de almacenar la información, que se pueden encontrar en el desarrollo de un software, a diferencia de otras propuestas que profundizan en solo un aspecto (Ambler Scott) o lo tratan de forma insuficiente (el resto de las descritas).

5. Todos los conceptos definidos dentro del marco formal del modelo O^3, así como del modelo de objetos de ODMG se obtienen como parte de las vistas estática y dinámica de las clases del dominio.

6. Como parte del modelo de persistencia se han sistematizado los pasos a seguir en el diseño de la BD, y se han definido recomendaciones que permiten obtener las especificaciones estáticas y dinámicas, utilizando los diagramas de clases y transición de estado y formatos para especificar algunas restricciones y fórmulas dinámicas, que no se derivan de estos diagramas. Además se describe cómo obtener de estos diagramas y especificaciones textuales información útil sobre los objetos, relacionada con su persistencia.

7. Aún cuando no siempre es posible almacenar los objetos del mundo real tal como se aprecian, la propuesta de conversión de las clases a tablas trata de conservar las principales características de este enfoque, por las ventajas que ofrece cuando hay cambios, y por la forma directa en que se representa el dominio de análisis.

8. Se ha definido una capa persistente tanto para el tratamiento de la persistencia en fichero, como su implementación con un SGBDR o SGBDOR a partir de un LPOO. En ambos casos se proponen los cambios a la estructura de las clases persistentes y la estructura de las nuevas clases que se requieren para reflejar las características de las relaciones entre las clases, que se describe en el DC, y para encapsular el trabajo con el medio de almacenamiento; correspondientes a las subcapas de especificación e interfaz, respectivamente. La subcapa de especificación no se ha descrito en trabajos anteriores y la subcapa de interfaz se define para el almacenamiento en ficheros, BDR o BDOR; lo que

hace superior esta capa persistente a otras propuestas. Ambas capas responden al modelo de persistencia del método DIBAO.

9. Se incluye la concepción de la automatización del método DIBAO, así como las características de los CASE obtenidos. **Gebase** y **Gecodel** presentan una interfaz agradable, generan información y documentación y realizan validaciones asociadas a la lógica del método; lo que ayuda al diseñador y disminuyen el tiempo de desarrollo. Además, un cambio en el esquema implica modificar la especificación y volver a generar, lo que resulta menos costoso que modificar la implementación.

10. Para garantizar la calidad de las herramientas obtenidas, se analizaron criterios asociados a las características de este tipo de productos (por ejemplo, las filosofías de trabajo que soportan, la rapidez en la generación). Atención especial se dedicó a la legibilidad del diagrama de clases, para lo cual se usó la solución flexible y eficiente que brinda un algoritmo basado en el Escalador de Colinas Estocástico.

Los resultados de esta tesis ya han tenido aplicaciones concretas, en particular cuando se diseña para almacenar los datos en ficheros y en SGBDR, a partir del uso de Borland Delphi.

RECOMENDACIONES

Las principales recomendaciones que se derivan de esta tesis son:

1. Llevar a una mayor escala los resultados de esta tesis, continuando su transmisión dentro de los cursos que reciben los futuros ingenieros en Informática e incrementado las acciones para capacitar al personal que trabaja en este campo.

2. Profundizar en el estudio de las características asociadas al comportamiento dinámico con vistas a definir atributos que contribuyan a validar la calidad del proceso de diseño de estas características.

3. Integrar las herramientas existentes dándoles un nivel comercializable.

4. Continuar en la búsqueda de las vías que permitan llegar a obtener la tecnología sobre la que se desarrolla Rational Rose, para permitir que el CASE integrado importe ficheros generados por esta herramienta.

5. Identificar y estudiar problemas que no tengan solución o sean difíciles de resolver con los modelos tradicionales, y en los que el uso de un SGBDO sea el adecuado.

GLOSARIO DE TÉRMINOS

Agregación: Es una forma especial de asociación que especifica una relación todo-parte entre la agregada (todo) y los componentes (parte).

Agrupación: La clase agrupada es una clase que agrupa a objetos de una clase base en una jerarquía. Se identifican en los lenguajes como Listas, Colecciones y Registros.

Base de datos correcta: Es el grado en que la base de datos está libre de errores o defectos, de acuerdo a los datos que se desea almacenar asociados al dominio del análisis.

Base de datos fácilmente mantenible: Es el grado en el que se puede cambiar el esquema de la base de datos y los programas provocando el mínimo de afectaciones en los datos almacenados.

Base de datos fácil de usar: Es el grado en que es fácil de usar la base de datos sin esfuerzo, para lo cual el sistema de gestión debe brindar capacidades que permitan realizar búsquedas y actualizaciones de forma rápida.

Base de datos fiable: Es la probabilidad de que la base de datos pueda ser usada sin que falle o se pierda información durante un período de tiempo determinado en unas condiciones de operación dadas. Expresa el grado de confiabilidad que se tenga sobre los datos.

Base de datos flexible: Es el grado en que el esquema de la base de datos se puede acomodar a los cambios.

Clases del dominio: clases que reflejan conceptos del mundo real que son abstraídos durante el desarrollo de una aplicación en particular.

Completitud de la base de datos: Es el grado en que se ha logrado definir de forma clara y concisa todos los datos del dominio del análisis que interesan de cuerdo a los objetivos de la aplicación a construir.

Composición: Es una forma especial de agregación en el que hay una relación fuerte entre el todo y las partes, de manera que la vida de las partes depende del todo.

Consistencia de la base de datos: Es el grado en el que la base de datos es única y no contradictoria respecto a los datos que se desean almacenar teniendo en cuenta los objetivos de la aplicación.

Estereotipo: Es una extensión del vocabulario de UML que permite crear nuevos artefactos o elementos a los ya definidos en la notación.

Integridad de los datos: Lograr que los datos estén de acuerdo a la realidad objetiva y no se produzcan inconsistencias, para lo cual se asocian reglas de integridad que deben validarse para mantener esta cualidad.

Llave primaria: Conjunto no vacío de atributos que identifican unívoca y mínimamente una tupla.

Llave extranjera: una llave primaria de una tabla que es embebida en otra (o la misma tabla).

Metaclase: Es una clase tal que sus instancias son clases.

Método: Conjunto de técnicas que se emplean para abordar una determinada fase del ciclo de vida de un sistema.

Metodología: Conjunto de métodos junto a unas reglas que permiten pasar de una fase a la siguiente en el ciclo de vida.

REFERENCIAS BIBLIOGRÁFÍCAS

1. Abiteboul, S.; Hull, R. And Vianu, V. “Foundation of Database”. Addison-Wesley Publishing Company, 1995.
2. Acosta, K. y Jiménez, K. “Generación de la base de datos (Gebase)”. Trabajo para optar por el título de Ingeniería Informática. CEIS, ISPJAE, Cuba. Junio 1999
3. Aho, A.; Slethi, R. y Ullman,J. “Compiladores. Principios, técnicas y herramientas”. Addison-Wesley Iberoamericana, S.A. 1990.
4. Alvarez, S: “Metodología de análisis y diseño orientado a objetos de sistemas informáticos”. Tesis para optar por el título de doctor en ciencias técnicas. ISPJAE. Julio 1995.
5. Alvarez, S.; Anache, I. y Hernández, A. “ADOOSI Visual: metodología de análisis y diseño orientado a objetos para medios ambientes visuales”. Revista Ingeniería Industrial. XX(2) 65-67, 1999.
6. Alvarez, S.; Anache, I.; Hernández, A. y Pita, V. MetVisualE 2.0: metodología de análisis y diseño estructurado para medios ambientes visuales”. Manual para uso docente en soporte electrónico. .CEIS, ISPJAE. Mayo 2000.
7. Alvarez,S. y Hernández, A. “Metodología ADOOSI-UML, versión 5.0”. Manual para uso docente en soporte electrónico. CEIS, ISPJAE. Cuba. 2001.
8. Ambler,S. “Schooling normalization”. Computing Canada. 22(17 22, August 15, 1996.
9. Ambler, S. “Design a robust persistence layer” (Part 4 of 4). Software Development. 6(4) 73-75, April 1998.
10. Ambler, S. “Persistence layer requirement”. Software develpoment. 6(1) 70-71, January 1998.
11. Ambler, S. “Persistence modeling in the UML”. Software Development Magazine. http://www.sdmagazine.com/articles/1999/ 0008a.htm. August 1999.
12. Ambler, S. “The OMG should extend the existing UML class diagram definition to help you develop real world, mission-critical applications using object and relational technologies”. Software Development Magazine. http://www.sdmagazine.com/articles/1999/0008q.htm. August 1999.
13. Ambler, S. “Mapping objects to relational databases. What you need to know and why”. http://www-4:ibm.com/software/developer/library/mapping-to-rdb/index.html. Page 1-9, 8/01/2000.
14. Ambler, S. “How the UML models fit together”. Software Development Magazine. http://www.sdmagazine.com/articles/2000/003z/ focus.ambler.htm. March 2000.
15. Ambler, S. “Crosing the object-data divide”. Software Development Magazine. http://www.sdmagazine.com/articles/2000/ 0003/0003j/0003j.htm. March 2000.
16. Ambler, SW. “The Object-oriented techniques to develop software”. http://www-106.ibm.com/developworks/library/using-oo/. June 2000.
17. Ambler, S. “The Design of Robust Persistence Layer for Relational Database”. http://www.amysoft.com/persistencelayer.pdf. October 21, 2000.

18. Ambler, S. “Mapping objects to relational databases”. http://www.Ambysoft.com/mappingObjects.pdf. October 21, 2000.
19. Ambler, S. “The Object Primer 2nd Edition: the application developer’s guide to objet-oriented”. http://www.ambysoft.com/theObjectPrimer.html. 2000.
20. Andrade, L. F.; Gouveia, J.C.; Xardoné, P.J. and Câmara, J.A. “Architectural concerns in automating code generation”. OBLOG Software S.A. http://www.oblog.com/news/concerns.html. 1999.
21. Anónimo. “Object relational mapping strategies”. http://www.obectmatter.com/vbf/docs/maptool/ormapping.html. 1999.
22. Anónimo. “Integración de la modelización de datos”. Objects by Design. http://www.objectsbydesign.com/tools/modelingtoolssp.htm/#Data%20Modeling. 1999.
23. Barden, D. ” Baloney Detection Kit”. The Journal conceptual modeling. Issue number 10. http://www.inconcept.com/JCM/August1999/Barden.html. August 1999.
24. Barrera, J. y Reyes, D. “Generación de código para Borland Delphi a partir de especificaciones del diseño orientado a objetos (Gecodel)”. Trabajo para optar por el título de Ingeniería en Sistemas .CEIS, ISPJAE. Cuba. Marzo 1998.
25. Barry, D. “ODMG 2.0: An overview”. Dr. Dobb’s Sourcebook. http://www.com/aerticles/1997/9717a/9717a.html. September/October. 1997.
26. Barry, D. “Trajectory Analysis: ODBMS Vendors”. http://www.barryassociates,com/trajectory/trajtoc.html. December 1997.
27. Barry, D. “ODMG 2.0: A standard for object storage” . Component Strategies. http://www.org/library/readingroom/Articles-Component Strategies-July’98.html. July 1998.
28. Barry, D. “ODBMS or ORDBMS”. Software Development. 6(10) 45-48, October 1998.
29. Becker, S. A. “Building a Better Data Model”. Issue of The Data Administration Newsletter, number 6. http://www.inconcept.com/JCM/ December1998/becker.html. December 1998.
30. Becker, S.A. “Conceptual Data Modeling in an Object-oriented Process (Part One). http://www.inconcept.com/JCM/February/2001/becker.html. February, 2001.
31. Bertino, E. and Martini, L. “Object-oriented Database Systems: concepts and architectures”. Addison- Wesley Publishing, Co. 1993.
32. Blaha, M. and Premarlini, W. “Observed idiosyncrais of Relational Database Design”. 2nd Working Conference on Reverse Engineering, Toronto. http://www.omtassociates.com/Media/2wcre.pdf. July 1995
33. Blaha, M. and Premarlini, W. “Object-Oriented concepts for database design”. http://www.omtassociates.com/Media/2writeppr.pdf. October, 1998.
34. Booch, G. “Object-Oriented design and analysis with applications”. Benjamin/Cummings Publishing Company. 2da edición. 1994.
35. Booch, G., Rumbaugh, J. and Jacobson, I. “The Unified Modeling Language user guide”. Addison-Wesley Longman, Inc. 1999.
36. Brooks, P.L. “Consulting the database crystal ball”. DBMS. 11(4) 57-60, April 1998.

37. Budd, Timothy. “An introduction to object-oriented programming”. Second Edition. Addison-Wesley Longman, Inc. 1997.
38. Burleson, D. “More than just pretend”. Database Programming & Design. 52-59, October 1995.
39. Buson, R. “Bases de datos relacionales frente a bases de datos orientadas a objetos”. PC World. Mayo 1993.
40. Bussert, J. “The rule of triggers”. MIDRAGE Systems. 11(8) 50, June 15, 1998.
41. Campbell, R. “What’s up with object-relational databases”. Databased Web Advisor. 15(8) 22-27, 1997.
42. Carsí J. A.; Canos, J.H. ; Ramos, I.; Penadés, M.C. y Pelechano, V. “Descripción del modelo de objetos de OASIS a través de la Transaction Frame Logic.” JIS’96, I Jornadas de Trabajo de Ingeniería de Software. Sevilla. Noviembre 1996.
43. Canós, J.H. “OASIS: un lenguaje único para Bases Datos Orientadas a Objetos”. Tesis doctoral. DSIC-UPV, España. Junio 1996.
44. Cattel,R. “ODMG 2.0”. Morgan Kaufman. http://www.odmg.org. 1997.
45. Cattel, R.G.G.; Barry, D.K.; Berler, M.; Eastman, J.; Jordan, D.; Rusell, C.; Shadow, O.; Stanienda, T. and Velez, F. The Object Data Standard: ODMG 3.0”. The Morgan Kaufman Series in Data Management Systems. http://www.odmg.org/. January 2000.
46. Chakravarthy, S.; Amuar, E.; Maugis, L. and Mishra, D. “Design of SENTINEL: An object-oriented DBMS with Event-Based Rules”. 1-23, 1994.
47. Chaudhri,A. and Loomis, M. “Object Database in practice”. Prentice-Hall, Inc. Hewlett-Packard Company. EUA. 1998.
48. Coleman, D.; Hayes, F. and Bear, S. “Introducing ObjectCarts or How to use statecharts in object-oriented Design”. IEEE transactions on Software Engineering. 8(1). January 1992.
49. Cooling, J.E. “Real-time software systems: An introduction to structure and object-oriented design”. International Jhonson Publishing Inc. EUA. 1997.
50. Date, C. J. “An introduction to Database systems”. Sixth Edition. Addison-Wesley Publishing Company., Inc. 1995.
51. Davis, J.R. “Object-relational database managers (ORDBMS)”. Distributed computing monitor .19(2) 3-29, February 1995.
52. Dalton, P. “Microsoft SQL Server black book”. The Corilis group, Inc. International Thomson Publishing Company. 1997.
53. Delgado, M. “Automatización del Diagrama de transición de estado para la metodología ADOOSI”. Tesis para optar por el título de maestra en Informática. CEIS_CREPIAI, ISPJAE.Cuba. Noviembre 1997.
54. Dennis, K. “Catálogo de productos Windows”. Windows source. October 1993.
55. Dey, D.; Barron, T. And Stoney, V. “A conceptual model for the logical design of temporal databases”. Decision Support Systems. 15 305-321, 1995.

56. Dragan, R. “Hybrid databases will incorporate the best feature of relational databases management systems and object oriented products”. PC Magazine. 11(17) 170, June 1998.

57. Duchesneau, D. “Object-oriented glossary”. DataBase Advisor. Page 86, April 1992.

58. Dudman, J. “Relative values”. Computer Weekly. 46, June 13, 1996.

59. Duhl, J. and Barry, D.K. “Object storage Fact Book 4.1, Object-relational mapping”. Mapping object to external DBMSs. Barry & Associates, Inc. http://www.object-relational.com/ormapping2.pdf. page 132, 2000.

60. Eisenberg, A. and Melton, J. “SQL:1999, formerly knows as SQL3”. SIGMOD Record.. http://www.cssinfo.com/ncits.html. 28(1) 131-138, March 1999.

61. Eisenberg, A. and Meltom, J. “SQL Standarization: The next steps”. SIGMOD Record. 29(1).63-67, March 2000.

62. Elmasri, R. and Navathe, S.B. “Fundamentals of Database Systems”. Third Edition. Addison-Wseley Longman, Inc. EUA. 2000.

63. Embley, D. “Object database Development: concepts and principles”. Addison Wesley Longman, Inc. EUA.1998.

64. Fertuck, L. “Systems Analysis and Design with CASE tools”. Wm. Brown publishers. 460-504, 1992.

65. Foley, J. “Open the gates to object”. InformationWeek. (579) 44-48, May 13, 1996.

66. Foley, J. “The race heats up”. InformationWeek. 593 14-15, August 19, 1996.

67. Fowler, M. “Why use the UML?”. Software Developer Magazine. http://www.sdmagazine.com/articles/2000/0003/0003z/ 0003z.htm. March, 2000.

68. Fowler, M. “Put your process on a diet”. Software Developer Magazine. http: //www.sdmagazine.com/articles/2000/0012/0012a/ 0012a.htm. December, 2000.

69. Francett, B. “Hybrid DBMSs offer best of both words.” Software Magazine. 15(8) 61-65, August 1995.

70. Frank, M. “Object-relational hybrids: a look at technology and products that blend object and relational”. DBMS. 8(8) 46-52, July 1995.

71. Frazer, D. “Object/relational groups up”. Database Programming & Design. 11 (1) 22-28, January 1998.

72. Frederiks, P.J..M.; Hofstede, ter A.H.M. and Lippe, E. “A Unifying Framework for Conceptual Data Modelling Concepts”. Information and Software Technology. http://www.icis.qut.au/~authur/articles/ HowTo.ps.Z. 39(1).15-25, January 1997.

73. Frost, R.D. “An active entity model for database design”. Twenty Fifth Anual Regionak Conference Northeast Decision Sciences Institute Proceedings. 281-283, April 1996.

74. García, L. y Alvarez, S. “Bases para una metodología de evaluación de la calidad del software”. Revista Ingeniería Industrial Cuba. XX(3) 74-78, 1999.

75. García, L. “Metodología para la evaluación de la calidad del análisis y diseño orientado a objetos usando ADOOSI”. Tesis para optar por el título de doctor en ciencias técnicas. ISPJAE. Junio 2000.

76. Graham, I. “Métodos orientados a objetos”. 2da edición. Adisson-Wesley/Díaz de Santos. España. 1996.
77. Grimes, S. “Modeling object/relational databases”. DBMS. 11(4) 51-54, April 1998.
78. Grimes, S. “Object/relational reality check”. Dabase Programmnig & Design. 11(7) 26-32, July 1998.
79. Halpin, T. “UML Data Models from an ORM Perspective: Part One”. Issue of The Data Administration Newsletter, number 1. http://www.inconcept/JCM/April1998/Halpin.html.. April 1998.
80. Harvey, T. “Poet extends C++ development Data Based Advisor. 13(10) 78-80, November. 1995.
81. Hernández, A. “Una herramienta CASE para la automatización del diseño orientado a objetos”. Tesis presentada en opción al título de maestra en Informática Aplicada a la Ingeniería y la Arquitectura. CREPIAI, Cuba. Febrero, 1996.
82. Hernández, A. Anache, I. y Alvarez, S. “Diseño orientado a objetos y bases de datos relacionales”. Memorias del evento internacional Compac’96, Cuba. ISBN-959237-024-9. Noviembre 1996.
83. Hernández, A. “Automatización del diseño orientado a objetos”. Revista Ingeniería Industrial, Cuba. XVIII(3), 1996-1997.
84. Hernández, A. “Soluciones al diseño de la base de datos a partir de un desarrollo orientado a objetos” . IX Conferencia Científica de Ingeniería y Arquitectura. II Taller de Informática Aplicada a la Educación Superior. Cuba. Diciembre 1997.
85. Hernández, A. “CASE: Una solución a la baja productividad en el software”. Revista Ingeniería Industrial, Cuba. XIX(1) 35-39, 1998.
86. Hernández, A. “Estado del arte de las bases de datos orientadas a objeto.”. Revista Ingeniería Industrial, Cuba XIX(1).40-46, 1998.
87. Hernández, A. y otros. “Soluciones al diseño de la base de datos a partir de un desarrollo orientado a objetos”. Publicación electrónica del evento internacional Informática’98, Cuba. Febrero 1998.
88. Hernández, A.; Alvarez, S. y Pastor, O. “Generación de código para implementar la persistencia a partir de especificación de diseño orientado a objetos”. XXV Convención de la UPADI. 1er Congreso de Informática. Perú. Noviembre, 1998.
89. Hernández, A. ” Diseño de la base de datos para entornos objeto/relacional”. SoST´99 Simposio en Tecnología de Software. 28as Jornadas Argentinas de Informática e Investigación Operativa. (JAIIO). Agosto, 1999.
90. Hernández, A. y Delgado, M. “Modelación de los comportamientos estáticos y dinámicos en el diseño de la base de datos a partir de un modelo orientado a objetos“. VII Convención Internacional Informática 2000, VII Congreso Internacional de Nuevas tecnologías y Aplicaciones Informáticas, Cuba. Mayo 2000.
91. Hernández, A.; Mato, R. y Tenorio, D. “Base de Datos: coexistencia de los modelos relacional y orientado a objetos”. Instituto Técnico Militar José Martí. Septiembre 2000.
92. Hernández, A. Estado del arte del modelo híbrido de base de datos”. Revista Ingeniería Industrial, Cuba. XXI(2), 2000.

93. Hernández, A. “Calidad del diseño de la base de datos a partir de una modelación utilizando la tecnología objeto”. X Conferencia Científica de Ingeniería y Arquitectura. III Taller de Informática Aplicada a la Educación Superior. Cuba. Diciembre 2000.

94. Hernández, A. y Alvarez, S. “Capa persistente de clases para el almacenamiento de objetos”. Aprobado para publicar en la Revista Ingeniería Industrial, Cuba. 2001.

95. Hill, R. “Improve your database design techniques”. Data Based Advisor 14(11) 82-86, November 1996.

96. Hofstede, ter A.H.M. and H.A. Proper. “How to Formalize It?. Formalization Principles for Information Systems Development Methods”. Technical report #4/97, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia. http://www.icisqut.edu.au/~authur/ Work.ps.Z. June 1997.

97. Hubbers. W.G.M. and A.H.M. ter Hofstede. Exploring the Jungle of Object-Oriented Conceptual Data Modeling. In Chris McDonald, editor, Proceedings of the 9th Australasian Database Conference, ADC'98, Springer, .Perth, Australia. Australian Computer Science Communications. 20(2) 65-76,. February 1998.

98. Hughes, J.G. “Object-Oriented Databases”. Prentice Hall. 1991.

99. Java. http://java.sun.com/features/2000/06/javapro.html. Junio 2000.

100. Jacobson, I., Booch, G. and Rumbaugh, J. “The Unified Software Development Process”. Addison-Wesley Longman, Inc. 1999.

101. Jepson, B. “What’s inside the ODMG-93 standard”. DBMS. http://www.dbmsmag.com/9707d131.html. July, 1997.

102. Jordan, D. “Identifying ODMG objects”. SIGS Publications, USA. http://www.borland.be/borlandcpp/news/report/CR9503jordan.html. 1996.

103. Jungclaus, R.; Hartman, T.; Saake, G. and Sernadas, C. “Introduction to Troll-a language for object-oriented specification of information systems”. Second International IS_CORE Workshop, London. 77-138, 1991.

104. Jungclaus, R.; Wieringa, R.J.; Hartel, P.; Saake,G. and Hartman, T. “Combining Troll with OMT”. B. Wolfinger (ed.). Innovation bei Rehen- und Komminikationssystems, Springer-Verlag. .35-42, 1994.

105. Jungclaus, R.; Saake, G.; Hartman, T. and Sernadas, C. “Troll-a language for object-oriented specification of information systems”. ACM Transactions on Information systems. 14(2) 175-2111, 1995

106. Keller, W.; Mitterbauer, C. and Wagner, K. “Object-oriented data integration”. In Chaudhri,A. and Loomis, M. “Object Database in practice”. Prentice-Hall, Inc. Hewlett-Packard Company. 3-20, 1998.

107. Kim, W. “Object-oriented database definition and research directions”. IEEE transaction on knowledge and Data Engineering. 1990.

108. Kirkpatrick, K. “Caching on live objects”. Computer Shopper. 16(15) 598-601, 1997.

109. Korth, H. And Siberschatz, A. “Fundamentos de Bases de Datos”. 2da edición. McGraw-Hill/Iberoamericana de España, S.A. 1993.

110. Kovács, G. and Van Bommel, P. “Conceptual modelling based design of objetct-oriented Databases"” Infomation and Software Technology. 40 (1) 1-14, 1998.

111. Kramer, M. I. “Oracle 8 the object/relational database management systems”. Open Information Systems. 12(7) 3-17, July 1997.

112. Kramer, M. I. “Object/Relational databases extending the relational information model”. Open Information Systems. 22(10) 3-21, October 1997.

113. Krill, P. “Oracle ties file system to objects”. InfoWorld. 20(3) 8, September 21, 1998.

114. Kroenke, D. “Procesamiento de base de datos”. 5ta Edición. Prentice-Hall Hispanoamericana, S.A. 1996.

115. Ladányi, H. “SQL unleasshed”. Sams Publishing 1997.

116. Larman,C. “Applying UML and patterns: an introduction to object-oriented analysis and design”. Prentice Hall PTR. 1998.

117. Letelier, P. y Saavedra, O. “Clases, objetos y programación lógica concurrente”. Reporte de investigación, DSIC-UPV, España.1993.

118. Letelier, P. y otros. “Oasis versión 3.0: un enfoque formal para el modelado conceptual orientado a objetos”. Servicio de publicaciones, DSIC-UPV, España. 1998.

119. Liberty, J. “Begining Object-Oriented Analysis and Design with C++”. Wrox Pres., Canadá.1998.

120. Linthincum, D.S. “Question persist about the user value of universal databases”. Government Computer News. 16(26) 38-41, 1997.

121. Lunderraman, R. “Oracle programmning: a primer”. Addison Wesley Logman, Inc. USA. 1999.

122. Luque, I. y Gómez-Nieto, M. “Diseño y uso de bases de datos relacionales”. Ra-ma, España. 1997.

123. Martin, J. and Odell, J. “Object-oriented methods: a foundation”. Second edition. Prentice-Hall, Inc. EUA.1998.

124. Mato, R.M. “ MetProo and GenProo as learning tools. Proceedings of International Conference on Engineering and Computer Education (ICECE 2000). Brasil. 2000.

125. Mc Clure, C. “CASE, la automatización del software”. 1993.

126. Medina , H. “Fusion: meodología orientada a objetos para desarrollo de software”. Revista Soluciones Avanzadas. 5(38), Octubre 1996.

127. Oblog Software S.A. “The OBLOG Specification Language”. http://www.oblog.com/tech/spec.html. 1999.

128. Objectivity. Chapter 3 Data modeling page 19-28, http:// www. objectivity.com/PDF/T0-52/T0-52-0-chap3.pdf. Chapter 5 Objectivity/DB http://www.objectivity.com/PDF/T0-52/T0-52-0-chap5.pdf. page37-42, 2000.

129. Objectivity/DB. “Standards Development and Compliance”. http://www. objectivity.com/Standards.html/Products/. 2000.

130. ODMG. “Standard overview. ODMG 2.9-Book extract”. http://www.ddj.com/articles/1997/971/917ª/9717a.htm. March 1997.

131. ODMG. “Standard overview”. http://www.odmg-org/standard/standardoverview.htm. 2000.

132. OMT. “Object-oriented analysis and design methods. Chapter 7. Topics 7.6 OMT Analysis and design process”. http://www.is.cs.utwente.nl:8080/dmrg/ OODOC/oodoc/oo-7.6.html.

133. OMT. “Object-oriented analysis and design methods. Tools for OMT”. http://www.is.cs.utwente.nl:8080/dmrg/MEE/misop009/misop14.html97.

134. Oracle Corporation. http://www.oracle.es/08/obj.html. 1997.

135. Pastor, O. “Diseño y desarrollo de un entorno de ejecución automática de software en el modelo orientado a objetos”. Tesis doctoral. DSIC-UPV, España. Abril 1992.

136. Pastor, O. y García, R. “Uso de Oracle como base de datos relacional para implementar un diseño orientado a objetos”. Generalitat Valenciana, Conselerria de Senitat i Consum. 3 de Mayo de 1993.

137. Pastor, O. y Ramos, I. “OASIS versión 2 (2.2): A class definition language to model information systems using an object-oriented approach”. SPUPV-95-788. 1995.

138. Pastor, O., García, R. y Cuevas, J. “Implementation of an OO design in an Oracle 7 development enviroment”. Informe de investigación. Universidad Politécnica de Valencia. 1995.

139. Pastor, O., Insfrán, E. y Pelechano, V. “Uso de Oracle como base de datos relacional para la implentación de un diseño orientado a objetos”. ”. Informe de investigación. Universidad Politécnica de Valencia. 1995.

140. Pastor, O.; Pelechano, V.; Bonet, B. y Ramos, I. “OO-Method 2.0: una metodología de análisis y diseño orientado a objetos”. Reporte de investigación DSIC,UPV. España. 1996.

141. Pastor, O.; Insfrán, E.; Pelechano, V.; Ramos, I. and Meseguer, J. “OO-Method: An OO software production environment combining conventional and formal methods”. Conference on Advanced Information Systems Engineering (CaiSE)’97. 145-158. Barcelona, Spain. 1997.

142. Pérez, S.; Alvarez, S. y Pastor, O. “Lenguajes de especificación formal para la metodología ADOOSI”. Memorias del XXV Congreso de la UPADI, I Congreso de Informática. Perú. Noviembre 1998.

143. Poet Software. “Object-oriented tools bulletin, section 3: Object Toolsets and OO Environments” . Reprinted ComputerWire PLC. http://www.poet.como/about/report/download.meta-group99.pdf. 1999.

144. Pressman, R. “Software engineering practitioner’s approach”. Fourth edition. McGraw-Hill. 1997.

145. Ramos, I.; Canos, J.H.; Forradellas,J. and Oliver, J. “A conceptual schema specification system for Rapid Prototying”. Proceedings of the XI ASTED Conference on Applied Informatics, Insbruck. February. 1990.

146. Rational Software Corporation. “UML Document Set Version 1.0”. 13 de enero de 1997.

147. Rational Software Corporation. “Rational lanza Rose DataModeler”. http://www.abist-com.mx/Fabs/Rational/NotasTk/rosdatamodeler. 2001.

148. Rational Rose Corporation. “Lo nuevo de Rational Rose 2000”. http://www.abits.com/Fabs/Rational/notastk/nuevoratrose2000.htm. 2000.

149. Reyes, J. “Analyze: Herramienta CASE para el análisis orientado a objetos”. Revista Ingeniería Industrial, Cuba.. XVIII(3) 27-30, 1996-97.

150. Rosenberg, D. “UML applied: Nine tips to incorparating UML into your project”. Software Development Magazine. http://www.sdmagazine.com/articles/2000/0003/0003z/0003z.htm. March 2000.

151. Rosete, A. and Ochoa, A. “Genetic Graph Drawing”. Proceedings of 13th International Conference of Applications of Artificial Intelligence in Engineering, AIENG’98, Adey, R. A.; Rzevski, G. and Nolan, P. (Eds). Galway, Computational Mechanics Publising. 37-40, July 1998.

152. Rosete, A. “Una solución flexible y eficiente para el trazado de grafos basada en el Escalador de Colinas Estocástico”. Tesis para optar por el grado científico de Doctor en Ciencias técnicas. ISPJAE, Cuba. Junio 2000.

153. Rudge, J.F. and Moorley, A.J. “Transaction procccesing in the capital market”. In Chaudhri,A. and Loomis, M. “Object Database in practice”. Prentice-Hall, Inc. Hewlett-Packard Company. 216-233, 1998.

154. Rumbaugh, J.; Blaha, M.; Premarlini, W.; Eddy, F. y Lorense, W. “Modelado y diseño orientado a objetos. Metodología OMT.” Prentice-Hall Hispanoamericana. S.A. 1996.

155. Rumbaugh, J.; Jacobson, I. and Booch, G. “The unified modeling language: reference manual”. Adison-Wesley Longman, Inc. Canadá.1999.

156. Saltor, F. “Sobre la evolución reciente de las bases de datos”. Monografía de Base de Datos Avanzadas. Novática. Novática@.ati.es. 140 4-7, 1999

157. Sánchez, P. y Ramos, I. “Object Petri Nets: Modelo de implementación para OASIS. Informe técnico No. 11-DSIC-17/96, DSIC-UPV, España. 1996.

158. Sánchez, P.; Letelier, P.; Ramos, I. y Pastor, O. “Modelo conceptual con un lenguaje formal y orientado a objeto”. DSIC-UPV. España. 2000.

159. Senn, J.A. “Análisis y diseño de sistemas de información”. 2da edición. McGraw-Hill Interamericana de México. 1998.

160. Sernadas,C. and Fidiero, J. “Toward object-oriented conceptual modeling”. Data&Knowledge Engineering. 7, 479-508. 1991.

161. Schach, S.R. “Classical and object-oriented software engineering with UML and C++”. Fourth Edition. WCB/McGraw-Hill. 1999.

162. Shaw, M. “Companing Architectual Design Styles”. IEEE Software, 27-41. 1995.

163. Smith, t. “The objetct of VAR’S desires: vendors try to integrate DB, relational DB”. Computer Reseller News. (651) 59, October 2 1995.

164. Snell, M. “The best of both world: ORDBMS combines relational and object funcionalities (hybrid object/relational DBMS). LAN Times. 12(14), 53-54. July 24, 1995.

165. Soon-Kyeong, K. and Carrington, D.. “Formalizing the UML class diagram using Object-Z”. In Robert France and Bernhard Rumpe, editors, *2nd International Conference on UML: UML'99: The Unified Modeling Language: Beyond the Standard, LNCS.* Springer, Verlag. Berlin. .*1723 83*-98, October 1999.

166. Stevens, A. object-oriented database management systems”. Supplement to dr. Dobb’s Joirnal. 75-82, April 1993.

167. Tsichritzis, D.C. and Lochovsky, F.H. “Data models”. Prentice Hall. 1982.

168. Toval, J. A. “Formalización de un entorno de producción automática de prototipos orientados a objeto”. Tesis doctoral, , DSIC-UPV, España. 1994.

169. Tucker, A.; Bradley, W.; Cupper,R. y Garnik, D. “Fundamentos de informática: lógica, resolución de problemas, programas y computadoras”. McGraw-Hill Interamericana de España. 1994.

170. Tuijn,Ch. and Gyssens, M. “CGOOD, a categorial graph-oriented object data model”. Theoretical Computer Science 160. 217-239. 1996.

171. Urman,. S. “Oracle8: PL?SQL programmning”. Osborne McGraw-Hill. 1998.

172. VanDuyVenvoor, D. “Being objective about RDBMS”. Computing Canadá. 23(21) 53-54, 1997.

173. Versant Corporation, http://www.versant.com. 2000.

174. Vessey, I. “Evolution of vendors products: CASE tools as methodology companions”. Commnunications of ACM. April 1992.

175. Vowler, J. “The business case for object database”. Computer Weekly, 16, April 20, 1995.

176. Waterson, K. “Easy your database development woes”. Database Advisor. VII(4), April 1993.

177. Wayna, M.; Christiansen, J.; Hield, Ch. and Simunick, K. “Modeling battefield: sensor environment”. In Chaudhri,A. and Loomis, M. “Object Database in practice”. Prentice-Hall, Inc. Hewlett-Packard Company. 210-215, 1998.

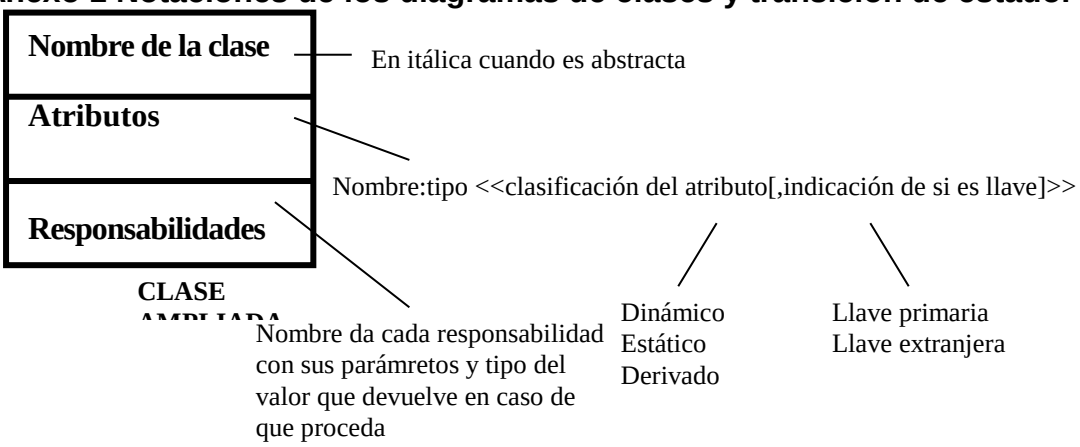
178. Wells,D. “Will Oracle 8 be universal”. Dabases programmning & Design. 10 14-19, October 1997.

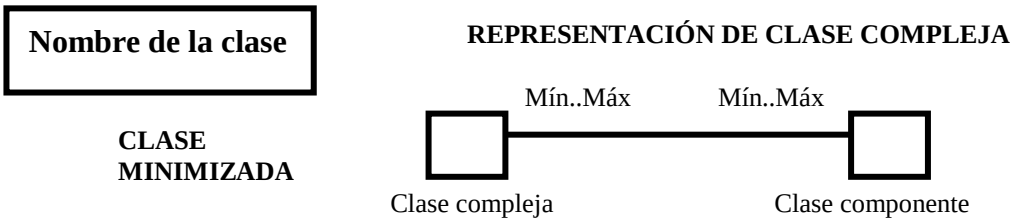
179. Wieringa, R,J.; “Algebraic Foundations for Dynamic Conceptual Model”. Centrale Huisdrukkerij Vrije Universiteit. 1990.

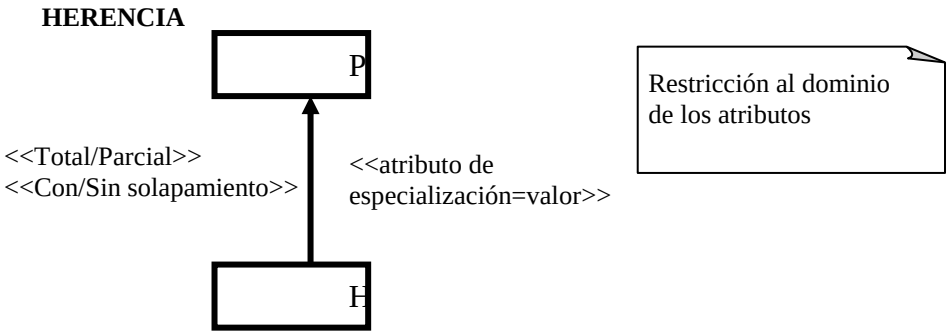
180. Wieringa, R,J.; De Jonge, W. and Spruit, P. “Using dynamic classes and role classes to Modelo Object Migration”. Theory an practice object systems. 1(1) 61-83, 1995.

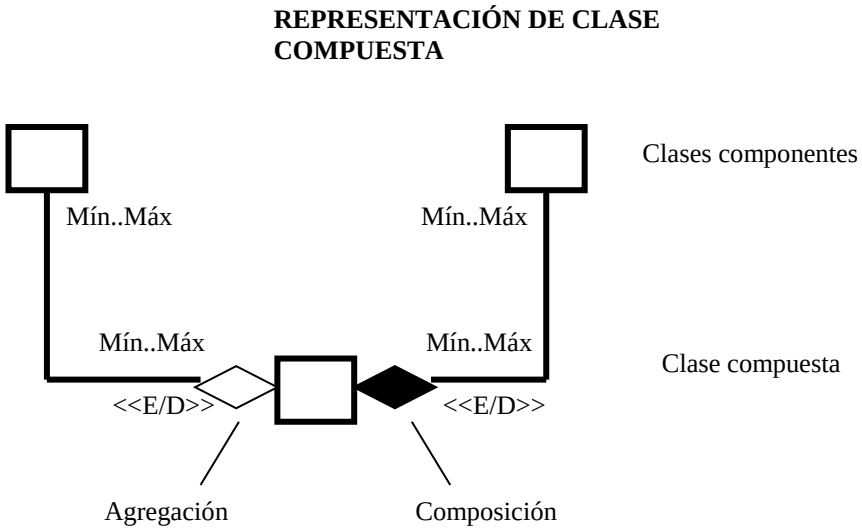
Anexos

Anexo 1 Notaciones de los diagramas de clases y transición de estado.







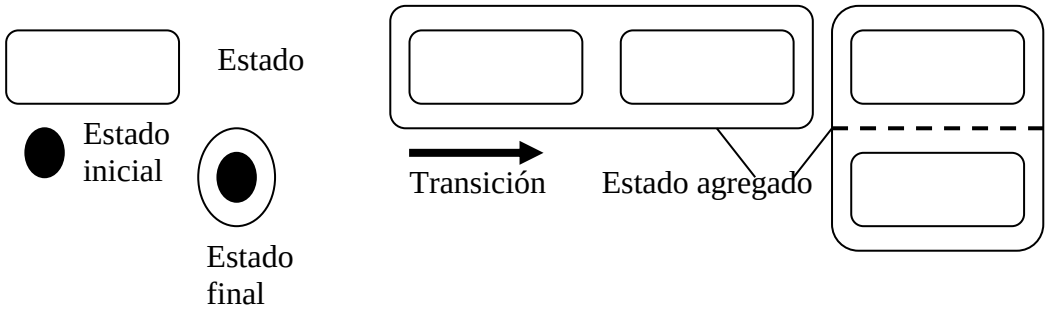


Notación del diagrama de clases.

Notas: << >> es el símbolo de estereotipo.

Mín..Máx simboliza las cardinalidades mínima y máxima respectivamente

<<E/D>> representa la dimensión.



Notación del diagrama de transición de estado.

Anexo 2 Fragmentos del caso de estudio:

Reservación de habitaciones en un hotel

En la automatización de un sistema de carpeta de un hotel se obtienen las siguientes especificaciones:

- Un huésped puede solicitar con antelación una habitación indicando el tipo de habitación que requiere, a partir de qué fecha y cuántos días.
- Por diferentes motivos una habitación y el empleado que la atiende pueden cambiar, e incluso antes de que el huésped comience a disfrutar de la reservación. Cuando lo que cambia es la habitación, hay que verificar si la nueva habitación asignada pertenece a otro grupo diferente al de la habitación precedente porque un empleado atiende a un solo grupo de habitaciones. En caso de que pertenezca a otro grupo, hay que buscar al empleado que la atiende.
- Una vez que el huésped está en el hotel puede solicitar prórroga, pero sólo lo podrá hacer el día que vence su reservación.
- Para un procesamiento estadístico posterior, se requiere almacenar el hecho de que una reservación cambie (por cambio de habitación, empleado o prórroga) o que se cancele.
- Si un huésped se va antes de que culmine su estancia, no es de interés que el sistema se entere.
- La fecha de inicio de una reservación no puede cambiarse.

Anexo 3 Resultados de la encuesta aplicada a expertos en diseño de la base de datos, sobre la influencia del método DIBAO en la calidad del software, de acuerdo a las normas ISO.

Encuestados:

	Cantidad
Total	**22**
Sin experiencia en el enfoque OO	**7**
Con experiencia en el enfoque OO	**15**

Características generales del encuestado:

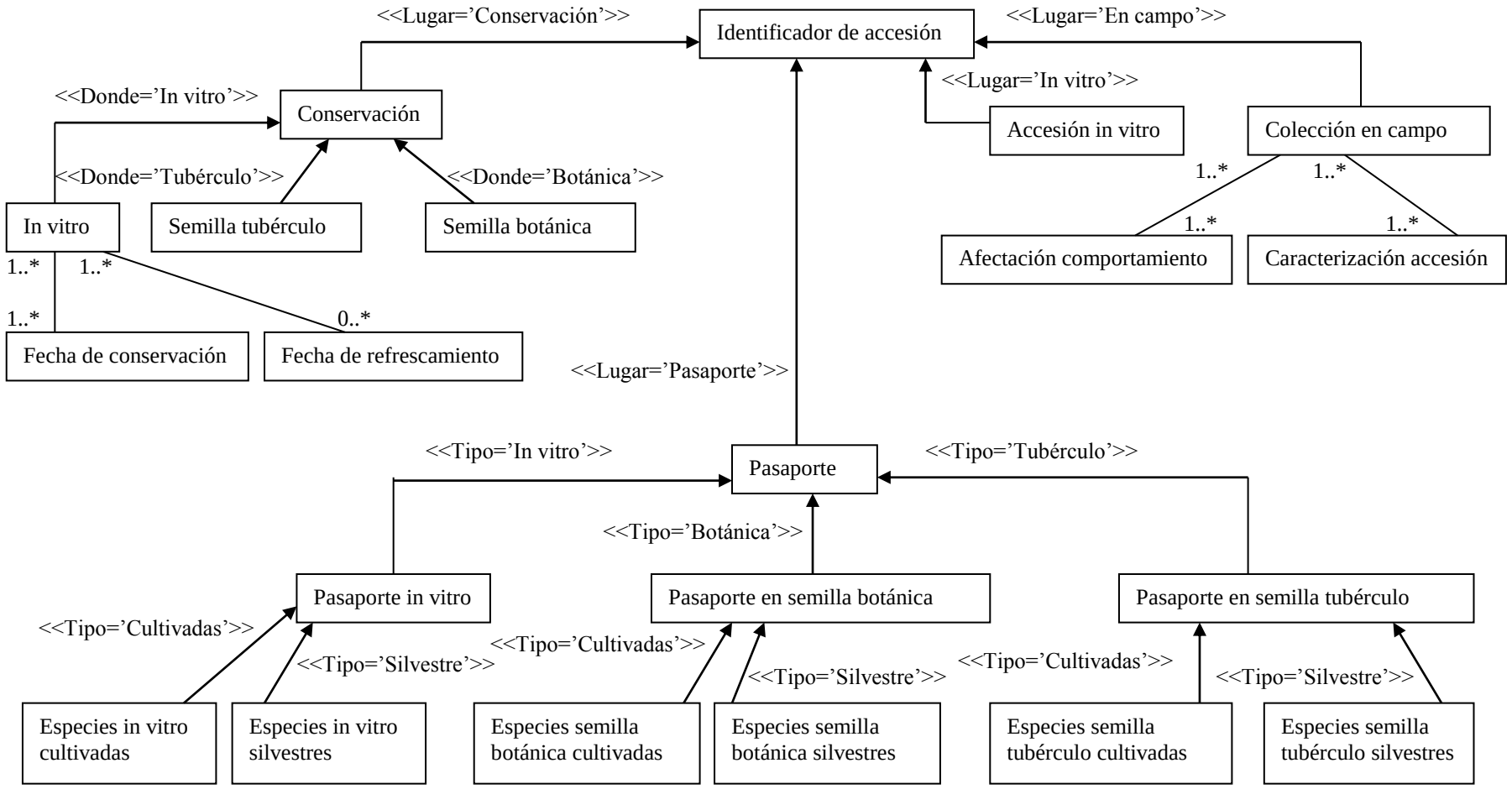
	Sin experiencia en el enfoque OO *Promedio de años*	Con experiencia en el enfoque OO Promedio de años
Años de experiencia en el trabajo con el enfoque orientado a objetos	**0**	**6**
Años de experiencia en el diseño de base de datos	**13**	**13**

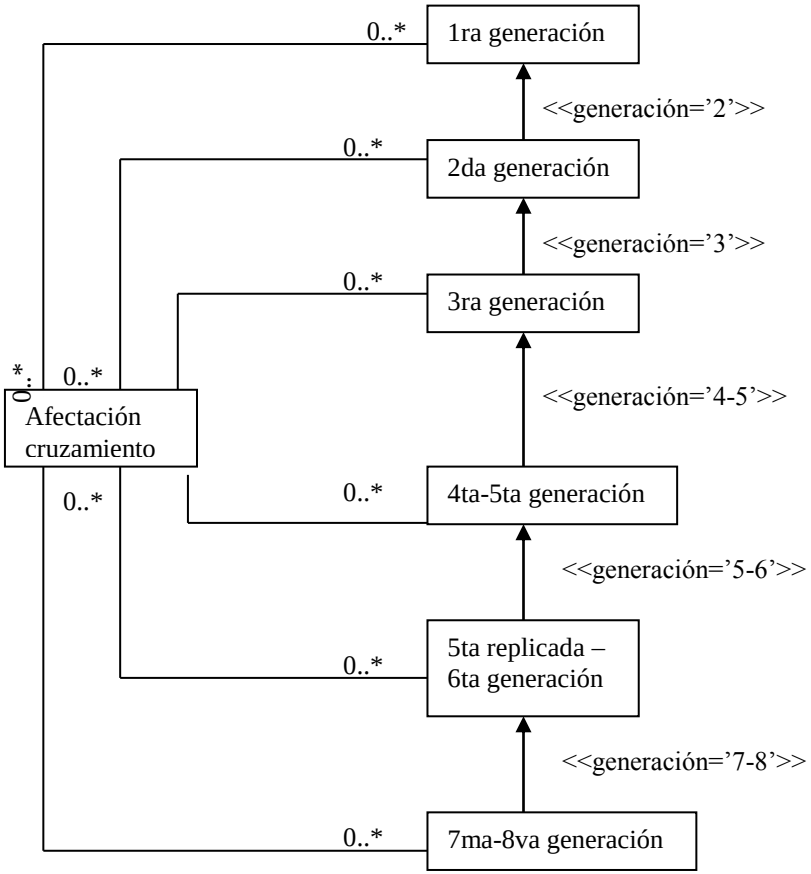
Evaluación de los factores de calidad que afectan al software (0-5):

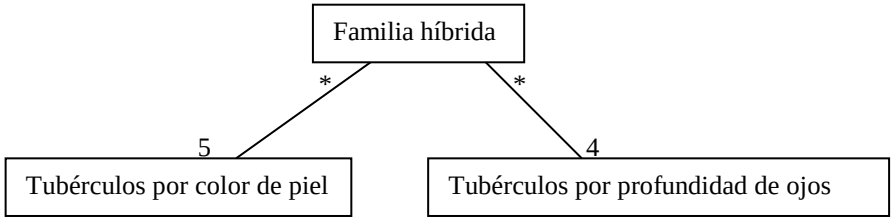
		Evaluación		
		Total	**Sin experiencia**	**Con experiencia**
1	Corrección: Grado en el que el diseño de la base de datos obtenido se corresponde con la funcionalidad.	**4.86**	**4.86**	**4.86**
2	Fiabilidad: Grado en el que el diseño de la base de datos obtenido está excepto de errores y puede ser usado durante un período de tiempo.	**4.23**	**4.13**	**4.43**
3	Eficiencia: grado en que el diseño de la base de datos obtenido garantiza una respuesta eficiente en tiempo.	**4.41**	**4.4**	**4.43**

		Evaluación		
		Total	**Sin experiencia**	**Con experiencia**
4	Integridad: Grado en que el diseño de la base de datos obtenido facilita la implementación de la seguridad de la base de datos.	**4.1**	**4.13**	**4**
5	Facilidad de uso: Esfuerzo requerido para trabajar con los pasos definidos por el método para el diseño de la base de datos.	**3.91**	**3.86**	**4**
6	Facilidad de mantenimiento: Esfuerzo requerido para localizar y arreglar un error cometido en el diseño de la base de datos, producto de un error en la definición inicial de clases.	**4.71**	**4.71**	**4.71**
7	Flexibilidad: Esfuerzo requerido para adaptar el diseño a cambios en el esquema.	**4.37**	**4.13**	**4.86**
8	Facilidad de prueba: Esfuerzo requerido para probar el diseño de la base de datos obtenido de acuerdo a la funcionalidad de las clases.	**4.14**	**4.14**	**4.14**
9	Portabilidad: Esfuerzo requerido para transferir, el diseño de la base de datos obtenido, de un Sistema de Gestión de Base de Datos a otro.	**4.19**	**4**	**4.57**
10	Reusabilidad: Grado en que es posible reusar las clases definidas.	**4.64**	**4.47**	**5**
11	Facilidad de auditoría: Grado en el que se sigue un estándar en el proceso de diseño de la base de datos.	**4.29**	**4.29**	**4.29**
12	Consistencia: Esfuerzo requerido para acoplar esta base de datos con otra.	**4.3**	**4.15**	**4.57**

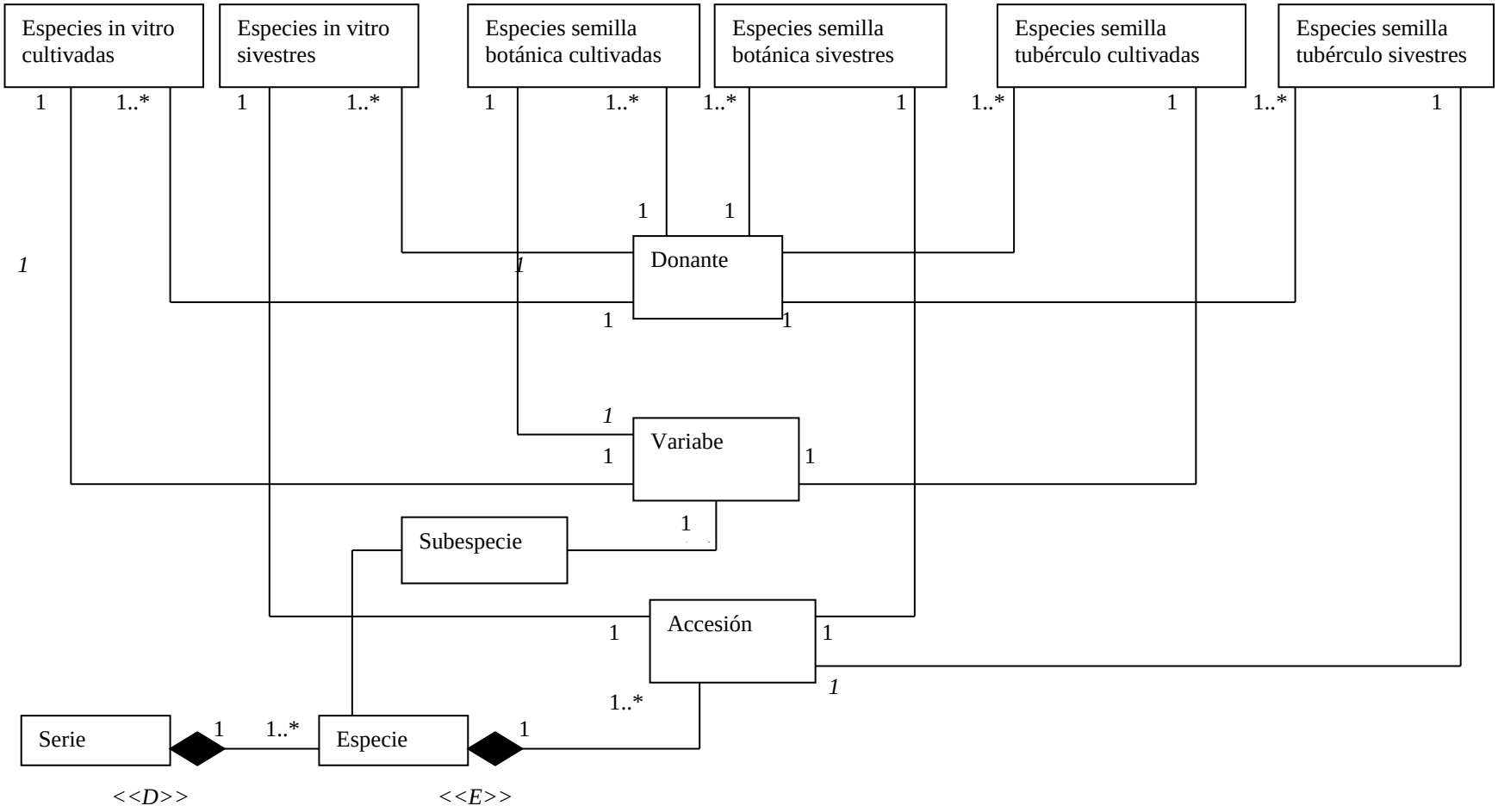
Anexo 4 Fragmentos de la documentación del "Software para la información, documentación y refinamiento genético del germoplasma de papa".





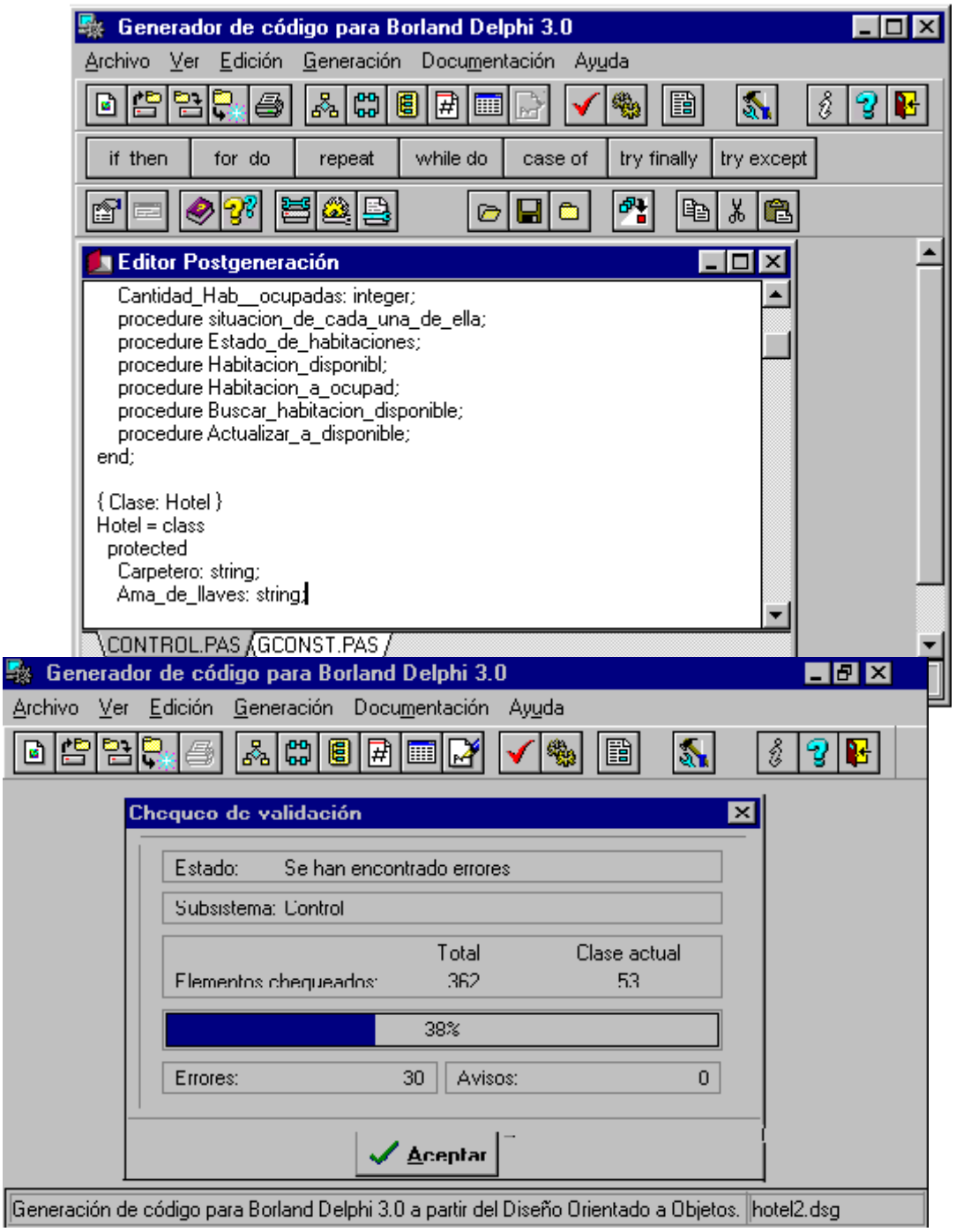


<<Total>>
<<Sin solapamiento>>
<<en todos los casos>>


Especies in vitro cultivadas
Especies in vitro sivestres
Especies semilla botánica cultivadas
Especies semilla botánica sivestres
Especies semilla tubérculo cultivadas
Especies semilla tubérculo sivestres
Donante
Variabe
Subespecie
Accesión
Serie
Especie
<<D>>
<<E>>

Anexo 5 Ejemplos de pantallas de Gecodel.

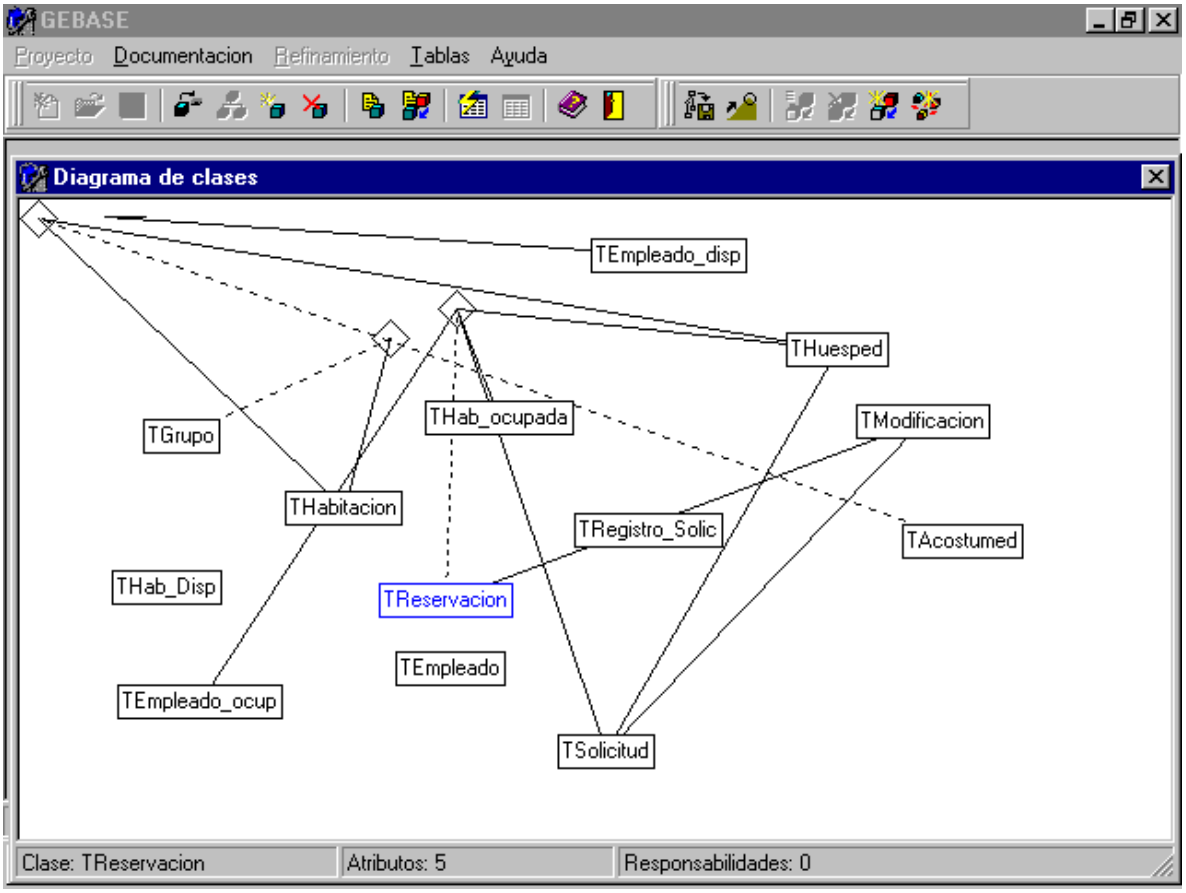
Edición postgeneración de las units.



Validación automática de contradicciones.

Anexo 6 Ejemplos de pantallas de Gebase.

Dibujo antes de aplicar algoritmo de trazado automático.



Dibujo después de aplicado el algoritmo para el trazado automático.