INFORMÁTICA
noviembre del 2000

CAPA PERSISTENTE DE CLASES PARA EL ALMACENAMIENTO DE OBJETOS

Resumen / *Abstract*
En este artículo se describe una capa persistente de clases, que aisla a las clases persistentes del dominio de la forma en que se almacena la información. La capa persistente responde a un modelo de persistencia que describe cómo se diseña la base de datos a partir del uso de una metodología de análisis y diseño orientada a objetos.

In this article is described a persistent layer of classes, that isolates to the persistent classes of the domain in the way in that the information is stored. The persistent layer responds to a persistence model that describes how the database is designed starting from the use of an object-oriented analysis and design methodology.

Palabras claves / *Key words*
Capa persistente, capa de objetos, modelo orientado a objetos, modelo de persistencia
Persistent layer, object layer, object oriented model, persistent model

Anaisa Hernández González, *Ingeniera en Informática, Máster en Informática, Asistente, Centro de Estudios de Ingeniería de Sistemas (CEIS), Facultad de Ingeniería Industrial, Instituto Superior Politécnico José Antonio Echeverría (ISPJAE), Ciudad de La Habana*
e-mail:anaisa@ceis.ispjae.edu.cu
Sofía Álvarez Cárdenas*, Ingeniera Hidráulica, Doctora en Ciencias Técnicas, Profesora Titular, CEIS, ISPJAE, Ciudad de La Habana*

INTRODUCCIÓN

Llevar los conceptos del mundo real a una base de datos (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 modelo relacional (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.

Diseñar la BD a partir de un modelo orientado a objetos (MOO) no puede hacerse aplicando los métodos tradicionales porque no se trabaja con los mismos conceptos, por lo tanto, se requieren de nuevos métodos y herramientas de desarrollo.[1] Además, los métodos y herramientas, utilizados para el MR, presentan limitaciones en el diseño ya que se refieren solo a los datos y sus relaciones, sobre los que no tienen en cuenta todas las restricciones a ellos asociados.[2]

Gran parte de las metodologías desarrolladas que siguen el enfoque orientado a objetos (OO), eluden el tema del diseño de la base de datos. Incluso hay algunas que reconocen que no abordan este problema.[3] Esta situación es lógica si se tiene en cuenta el estado de los gestores de objeto, y que el desarrollo de métodos para un nuevo enfoque va precedido del desarrollo de lenguajes que lo soporten.

La tendencia más explotada cuando se trabaja utilizando el enfoque orientado a objetos, es convertir las clases a tablas o almacenar la información utilizando las potencialidades de los len-

guajes de programación orientados a objetos para guardar los valores de los atributos de los objetos. En cualquiera de las dos variantes, se pierden conceptos importantes asociados al enfoque, por ejemplo, el concepto de objeto y la posibilidad de que persistan juntos, datos y comportamientos, respectivamente.

Para atenuar en parte estas limitaciones, que imponen el no uso de un gestor de objetos como medio de almacenamiento, trabajos presentados en los últimos años proponen crear una capa de objetos que sirva de interfaz entre las clases del dominio y el medio de almacenamiento, de forma tal que los objetos se sigan viendo como objetos aunque se almacenen en tablas.

En este trabajo se aborda una propuesta de definición de esta capa, a partir de la aplicación de un modelo de persistencia en el que se definen los pasos a seguir para diseñar la BD a partir de un modelo orientado a objetos.

CAPA PERSISTENTE O DE OBJETOS

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 base de datos relacional - BDR o incluso jerárquica) y las aplicaciones que utilizan esta.[4-10] 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.

La definición más completa de esta capa se ha encontrado en los trabajos de Jesse Liberty[9] y Ambler Scott.[6]

La metodología de Liberty incluye un modelo de persistencia para el almacenamiento de la información en ficheros y en una base de datos relacional. 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 creación, recuperación, actualización y eliminación (CRAE). Esta propuesta es similar a lo que brindan los lenguajes de alto nivel para garantizar la persistencia, 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 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 la referencia 6 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.

El método para el diseño de la BD que propone Ambler, consiste en un modelo de persistencia para la notación Unified Modeling Language (UML) que se refiere a la parte estática,[11] y se basa en definir varios diagramas de clases cuyos títulos, 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 objetos, que no se tienen en cuenta.

MODELO DE PERSISTENCIA

La persistencia es la capacidad de un objeto para existir fuera de un programa, proceso, función o hilo de control;[12] de manera que se conserva su estado y su comportamiento.

Para garantizar la persistencia, los sistemas de gestión de base de datos relacionales (SGBDR) disponen de las operaciones:

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.[13]

El modelo de persistencia utilizado 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.

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.

Identificar cuáles son las clases persistentes y cuáles son las temporales. Existen algunas reglas que el diseñador puede utilizar cuando se identifican las clases persistentes.[14,15]

2. Refinar las clases.

Revisar si existe algún comportamiento de interés que no se haya tomado en cuenta y que sea trascendental en la solución del problema, para incluirlo. Crear nuevas relaciones padre-hijo, a partir de clases con comportamiento y atributos comunes.

3. Clasificar las clases y los atributos.

Clasificar las clases, de acuerdo con los atributos que la integran, en: simples, complejas y compuestas. Para los atributos, se adopta la clasificación propuesta en la referencia 16, que los identifica como estáticos, dinámicos y derivados.

4. Realizar el diagrama de clases (DC).

El DC toma como referencia la notación del diagrama de estructura de clases de UML,[17,18] aunque se añaden restricciones

de integridad estática (asociadas al dominio de los atributos) y dinámica (relacionadas con las características de las relaciones de herencia y las asociaciones entre clases), utilizando estereotipos y notas. Se toma la definición de OO-Method de dimensión estática-dinámica.[16]

Existe un conjunto de pasos para la construcción del DC y para la definición de las cardinalidades de las asociaciones, de acuerdo con la clasificación de las clases y el tipo de asociación que existe entre ellas.[19]

5. Realizar el diagrama de transición de estado (DTE).

El DTE se utiliza para describir el comportamiento dinámico de los objetos. Se realiza para aquellas clases que posean atributos dinámicos pues, en función de estos, es que están los posibles estados por los que transita un objeto. Hay recomendaciones para:

- Formato de transiciones.
- Definición de estados agregados sobre la base de los eventos que afectan a un mismo atributo y el acoplamiento entre estados. Además, se sugieren los estudios de George Miller.[20]
- Reglas para el balance entre los niveles de diagramas, en cuanto a las transiciones de entrada y salida, que se describen en la tesis de maestría.[2]

A partir de la clasificación de los atributos dinámicos en cardinales, característicos de un estado y pertenecientes a una situación,[16] se brinda recomendaciones para identificar estados, como parte del proceso de construcción del DTE.

6. Obtener las restricciones estáticas y las fórmulas dinámicas.

Se obtienen los comportamientos estático y dinámico de los objetos en función del diseño de la BD, a partir de los diagramas de clases y transición de estado construidos. Del DC se pueden obtener todas las características estáticas, hay un formato para expresar otras restricciones del dominio de los atributos, para las cuales la especificación de que: su valor es único o no para un objeto, puede o no tomar valor nulo, tipo de dato, valor por defecto, rango y formula de derivación (se describe textualmente utilizando formato predefinido), es insuficiente.

Para la vista dinámica se pueden obtener disparadores y precondiciones del DTE. En el caso de los disparadores, no todos se obtienen de este diagrama porque no siempre un cambio de estado tiene como antecedente un cambio en la BD, por lo que hay un formato de especificación del mecanismo de disparo.

En el caso de las fórmulas de evaluación de los atributos dinámicos, su especificación (al igual que en OO-Method) se obtiene siguiendo las trazas de las actividades que se ejecutan para cada estado y que dependen de su clasificación.

Existen algunas construcciones que pueden utilizarse para describir métodos de clases, actividades de estados y acciones de transiciones, partiendo de las recomendaciones descritas,[21-15] en, y que complementan el lenguaje de especificación de UML.[18]

7. Convertir las clases al medio de almacenamiento.

Aquí se tiene en cuenta hacia qué medio se guardarán los objetos, especificando cómo las características estáticas y dinámicas se implementan, analizando tres soluciones:[2,14]

1. Si se tiene un SGBD de objetos (SGBDO), hay que garantizar que las definiciones obtenidas se correspondan con el modelo Object Database Management Group (ODMG).

El modelo de persistencia satisface al modelo de objetos de ODMG. En estos casos, no es necesario convertir las definiciones porque la tendencia de estos gestores es a que se basen en ODMG.

2. Si lo que se tiene es un lenguaje de programación OO (LPOO) que permite el trabajo con una gestor relacional o un SGBD objeto-relacional (SGBDOR), la solución es llevar las clases definidas a tablas.

Están definidas las reglas de conversión de clases simples, compuestas y complejas, de la herencia, de la agregación y de las asociaciones entre clases.[19,26] En esta conversión se mantienen algunas características de los objetos 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.

En el caso de las restricciones estáticas y fórmulas dinámicas, se describe cómo definirlas como métodos de las clases (si se posee un LPOO) o utilizando procedimientos almacenados y disparadores (si el trabajo es con un gestor que no permite el trabajo con un lenguaje de alto nivel OO). Se incluye una valoración sobre la conveniencia o no de utilizar esta segunda variante.

3. Si lo que se tiene es un LPOO que no impida el trabajo con una base de datos relacional, pero que permite salvar registros, entonces se deben utilizar las clases que brinda el lenguaje, para almacenar los atributos de las clases persistentes en ficheros. Puede que el lenguaje posibilite la relación con una BDR, pero se escoge como forma de almacenamiento la organización en ficheros.

Las clases persistentes sufren modificaciones relacionadas con la adición de métodos y la herencia de la clase que el lenguaje define para guardar los valores de los atributos de un objeto.

CAPA PERSISTENTE

Cuando se trabaja con un gestor de objetos, 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 se utilizan como medio de almacenamiento los ficheros o un LPOO que permita relacionarse con gestores relacionales u objeto-relacionales, 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.[6,9] Se 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 1).

Teniendo en cuenta lo anterior, a continuación se describe la propuesta de las autoras de este trabajo, sobre cómo modificar la estructura de las clases definidas para un aplicación, para garantizar que se implementen los comportamientos estático y dinámico, antes descritos, y la creación de nuevas para el trabajo con la información almacenada.

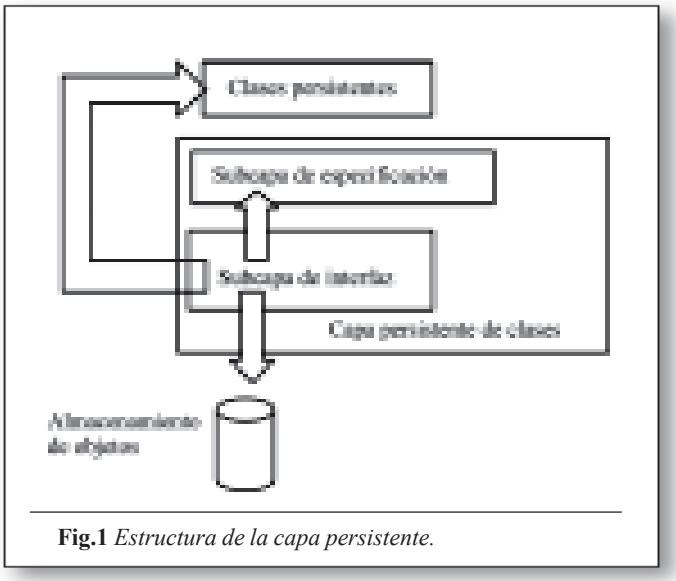

Fig.1 *Estructura de la capa persistente.*

Modificación de la estructura de las clases persistentes

Al aplicar el modelo de persistencia, se obtienen restricciones al dominio de los atributos y las fórmulas dinámicas asociadas a las clases, por lo que se hace necesario 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 de lo que indique, como evento que lo provoca, la especificación del mecanismo de disparo), 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.

Subcapa de especificación

En la fase de diseño de la BD, se han 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 rutas 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 puede 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 corresponde 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 con 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 tenga 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.

Subcapa de interfaz

Ambler Scott[6] 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 dos de ellas.

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 el 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 afecta a los métodos. 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, pero la 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	Lista que contiene las tablas asociadas a la clase base y fue generada a partir de las conversiones de clases a tablas.

• Métodos para salvar, borrar, recuperar y modificar objetos, que se redefinen en cada clase hija de acuerdo con 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 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.

CONCLUSIONES

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 se le da nombre a estas dos subcapas.

Los aspectos más importantes de esta capa son:

• La modificación de la estructura de las clases persistentes del dominio para recoger las restricciones estáticas y las fórmulas dinámicas obtenidas.

• La nuevas clases que surgen, como parte de la subcapa de especificación, que reflejan las características de las relaciones padre-hijo y de las cardinalidades y dimensiones de las asociaciones entre las clases.

• Las variantes de la implementación de la subcapa de interfaz, si se utiliza la implementación a ficheros o hacía un gesto relacional u objeto relacional. En el primer caso se describen los cambios que se realizan a las clases del dominio, y en el segundo, la estructura de las nuevas clases.

• La relación de las nuevas clases con las clases persistentes del dominio.

Las complejidades que pueda tener el acceso a la información almacenada, se reducen cuando se tiene definida una capa persistente de clases, aunque su implementación requiere más esfuerzo de los programadores. Pero, las ventajas que ofrece cuando ocurre un cambio en el esquema o cuando se cambia de gestor o versión de un mismo gestor, con respecto a las clases del dominio que no se verían afectadas, puede resultar un aspecto interesante que motive su definición.

REFERENCIAS

1. **HERNÁNDEZ, A.:** "Diseño de la base de datos para entornos objeto/relacional", SoST´99 Simposio en Tecnología de Software, 28 Jornadas Argentinas de Informática e Investigación Operativa (JAIIO), agosto, 1999.
2. **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.
3. **MEDINA , H.:** *"*Fusión: meodología orientada a objetos para desarrollo de software*", Revista Soluciones Avanzadas*, p. 5(38), octubre, 1996.
4. **AMBLER, S.:** "Design a Robust Persistence Layer", *Software Development*, pp. 6(4) 73-75, April, 1998.
5. **_____:** "Persistence Layer Requirement", *Software Develpoment*, pp. 6(1) 70-71, January 1998.
6. **_____:** "The Design of Robust Persistence Layer for Relational Database", http://www.amy-soft.com/ persistencelayer. pdf, October, 21, 2000.
7. **KELLER, W.; C. MITTERBAUER AND K. WAGNER:** *"*Object-Oriented Data Integration", in *Object Database in Practice,* pp. 3-20, Prentice-Hall, Inc. Hewlett-Packard Company, 1998.
8. **LARMAN, C.:** *Applying UML and Patterns: an Introduction to Object-Oriented Analysis and Design*, Prentice Hall PTR, 1998.
9. **LIBERTY, J.:** "Begining Object-Oriented Analysis and Design with C++", Wrox Pres, Canadá, 1998.
10. **WAYNA, M.; J. CHRISTIANSEN; CH. HIELD AND K. SIMUNICK:** *Modeling Battefield: Sensor Environment, Object Database in Practice,* pp. 210-215, Prentice-Hall, Inc. Hewlett-Packard Company, , 1998.
11. **AMBLER, S.:** "Persistence Modeling in the UML", *Software Development Magazine*, http://www.sdma-gazine.com/ articles/1999/ 0008a. htm, August 1999.
12. **DUCHESNEAU, D.:** "Object-Oriented Glossary", *DataBase Advisor*, p. 86, April, 1992.
13. **AMBLER, S.:** "Mapping Objects to Relational Databases", http://www.Ambysoft.com/mapping Objects.pdf, October 21, 2000.
14. **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 Aplícada a la Educación Superior, Cuba, diciembre, 1997.
15. **HERNÁNDEZ, A. Y M. DELGADO:** "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.
16. **PASTOR, O.; V. PELECHANO; B. BONET Y I. RAMOS:** "OO-Method 2.0: una metodología de análisis y diseño orientado a objetos", Reporte de investigación DSIC,UPV, España, 1996.
17. **AMBLER, S.:** "How the UML Models fit Together", *Software Development Magazine*, http://www.sdmaga-zine.com/ articles/2000/003z/ focus.ambler.htm, March 2000.
18. **RUMBAUGH, J.; I. JACOBSON AND G. BOOCH:** *The Unified Modeling Language: Reference Manual,* Adison-Wesley Longman, Inc, Canadá,1999.
19. **HERNÁNDEZ, A.; R. MATO Y D. TENORIO:** "Base de datos: coexistencia de los modelos relacional y orientado a objetos", Instituto Técnico Militar José Martí. septiembre, 2000.
20. **COOLING, J. E.:** *Real-Time Software Systems: An Introduction to Structure and Object-Oriented Design*, International Jhonson Publishing Inc, EUA. 1997.
21. **DELGADO, M.:** "Automatización del Diagrama de transición de estado para la metodología ADOOSI*",* Tesis para optar por el título de máster en Informática, CEIS, CREPIAI, ISPJAE,Cuba, noviembre, 1997.
22. **AHO, A.; R. SLETHI Y J. ULLMAN:** *Compiladores. Principios, técnicas y herramientas*, Addison-Wesley Iberoamericana, SA, 1990.
23. **ÁLVAREZ, S.; I. ANACHE; A. HERNÁNDEZ Y V. PITA:** *MetVisualE 2.0: metodología de análisis y diseño estructurado para medios ambientes visuales*, CEIS, ISPJAE, mayo, 2000.
24. **FERTUCK, L.:** *Systems Analysis and Design with CASE Tools*, pp. 460-504, Wm. Brown Publishers,1992.
25. **URMAN, S.:** *Oracle8: PL?SQL Programmning*, Osborne McGraw-Hill. 1998.
26. **HERNÁNDEZ, A.; I. ANACHE Y S. ÁLVAREZ:** "Diseño orientado a objetos y bases de datos relacionales", Memorias del Evento Internacional Compac'96, Cuba, noviembre, 1996.

EDICIONES
IMPRENTA
CUJAE