UNICIENCIA 22
pp. 87-92
2008

# MODELOS, LENGUAJES Y ABSTRACCIÓN

***José Aurelio Sánchez***
Escuela de Informática, Universidad Nacional, Apdo. 86-3000, Heredia, Costa Rica
jsanchez@una.ac.cr
Centro de Formación en Tecnologías de Información (CENFOTEC),
jsanchez@cenfotec.com

## RESUMEN

Todo proceso de construcción (ya sea de edificaciones, de máquinas, de software, etc.) requiere previamente construir un *modelo* del artefacto a fabricar. Este modelo tendrá como base o punto de partida un paradigma o metamodelo elegido: cuáles son los elementos básicos de modelaje, qué conceptos del mundo permiten representar o modelar, qué tipos de interrelaciones pueden establecerse entre ellos, etc. También deberá existir un *lenguaje* que nos permita expresar o representar el modelo, manipularlo de diversas formas y sacar conclusiones o pruebas de las características y desempeño del artefacto a construir. Necesariamente, el modelo que hagamos será solo una *abstracción* del artefacto real a construir, por ello, tanto el paradigma que elijamos como el lenguaje que utilicemos deben tener la capacidad de expresar dicha abstracción, concentrándose en elementos esenciales y relegando detalles accidentales. Además, muy deseablemente, ambos, el paradigma y el lenguaje, deberán permitir hacer construcciones con mayores niveles de abstracción a partir de otras más elementales. En este artículo se caracteriza la interrelación que existe entre estos tres conceptos, modelo-lenguaje-abstracción, también, se ubican en este enfoque varios modelos históricamente utilizados y se da una perspectiva más general para visualizar futuras propuestas.

**Palabras claves:** Modelaje de software, abstracción, paradigma, orientación a objetos, modelo relacional.

## ABSTRACT

Every construction process (whatever buildings, machines, software, etc.) requires first to make a *model* of the artifact that is going to be built. This model should be based on a paradigm or meta-model, which defines the basic modeling elements: which real world concepts can be represented, which relationships can be established among them, and son on. There also should be a *language* to represent, manipulate and think about that model. Usually this model should be redefined at various levels of *abstraction*. So both, the paradigm an the language, must have abstraction capacity. In this paper I characterize the relationships that exist between these concepts: model, language and abstraction. I also analyze some historical models, like the relational model for databases, the imperative programming model and the object oriented model. Finally, I remark the need to teach that model-driven approach to students, and even go further to higher level models, like component models o business models.

**Keywords:** Software modeling, abstraction, paradigm, object oriented, relational model.

## INTRODUCCIÓN

Usualmente, los desarrolladores de software utilizan herramientas (IDE[1], CASE[2], etc.) para la construcción de sistemas. Con frecuencia, a la ya inherente dificultad del proceso de construcción de software se une la dificultad del uso, y más aún la del aprendizaje, de estas herramientas. Consideramos que muchas de estas dificultades obedecen a que

[1] IDE: Integrated Development Environment.
[2] CASE: Computer Aided Software Engineering.

Recibido 7/4/2006 / Aceptado 25/5/2006
Revisado por Ing. Laura Núñez González

se intentan aprender y utilizar dichas herramientas sin procurar primero identificar y comprender el paradigma o metamodelo que soportan o apoyan, y que se usará en el modelado y construcción del sistema dado. De tal manera, la atención se dirige a las facilidades (*features*) específicas de la herramienta, sin antes tener una visión global de ella: su modelo básico, la capacidad de representación de éste, los tipos de manipulaciones que el modelo permite y el lenguaje en que ellas se expresan, las abstracciones que el modelo soporta y cómo construirlas, etc.

Al construir un sistema debemos, al igual que se hace en otras áreas del quehacer humano, elegir el paradigma, los paradigmas o metamodelos que vamos a usar y luego modelar el sistema con el mayor nivel de abstracción posible, utilizando el lenguaje o los lenguajes, gráficos o de prosa, atinentes. Es importante recalcar aquí que el paradigma que elijamos moldeará, en gran parte, nuestro pensamiento respecto del problema a resolver. Una vez construido, el modelo podrá ir siendo expresado en niveles inferiores de abstracción, agregando cada vez más detalles, en un proceso de refinamiento por pasos. Al pasar de un nivel de abstracción a otro podemos, ya sea permanecer en el mismo paradigma o metamodelo original, o expresar, todo o parte del modelo, en otro paradigma o metamodelo. Por ejemplo, podríamos pasar de un modelo orientado a objetos en UML a dos modelos, uno de datos, representado con el modelo relacional, y otro de programación, representado en un lenguaje de programación como C++, Java o C#. Por supuesto que habrá que hacer los "mapeos" entre unos y otros modelos.

En este artículo se presentan varios modelos, no sólo por su interés particular sino como ejemplos de la interrelación que existe entre el modelo, el lenguaje y la abstracción en cada uno de ellos.

Primero se presenta el modelo relacional, luego el modelo de programación procedural estructurado y después el modelo de orientación a objetos.

Al final, a manera de conclusión, se reflexiona sobre esta visión basada en modelos, se plantean recomendaciones y se señalan aspectos abiertos en esta temática.

## 1. El modelo relacional y el lenguaje SQL

En el año 1970 Ted Codd [COD1970] inventó el "Modelo Relacional", como una herramienta general para modelar y operar datos y sus relaciones utilizando conceptos matemáticos, como el de relación, álgebra y cálculo relacional. Posteriormente, Chris Date y otros ampliaron y desarrollaron dicho modelo. Este modelo venía a ser una alternativa a los modelos jerárquico y de red prevalecientes en el momento.

El modelo relacional sirve para modelar datos, lo cual es una parte muy importante de una aplicación o sistema. Aunque, claro está, los datos sólo representan un aspecto de una aplicación, pues faltaría modelar los aspectos dinámicos o reglas de negocios y los aspectos de interacción con el usuario, entre otros.

Una base de datos esencialmente es un "conjunto de datos relacionados entre sí" que son de interés en una situación particular. Desde la perspectiva relacional:

> *una base de datos está formada por un conjunto de tablas (relaciones matemáticas), las cuales contienen registros (tuplas) formados por columnas (atributos en determinados dominios). Usualmente cada registro se identifica de manera única por los datos almacenados en algunas de sus columnas, que constituyen la llave primaria de la tabla. Las relaciones entre tablas se representan por medio de llaves foráneas, es decir por columnas en un registro que almacena la llave primaria del registro relacionado en la otra tabla. Los registros, las tablas y sus relaciones pueden operarse de diversas maneras para obtener los datos que respondan a diversas consultas* [ELM2000].

En un modelo relacional el desarrollador representa cada concepto del mundo real por medio de una tabla o relación matemática, y cada característica de ese concepto por medio de una columna o atributo. Cada instancia particular del concepto corresponderá a un registro en dicha tabla. Las relaciones entre conceptos se modelan por medio de llaves foráneas entre tablas.

El modelo relacional, como fue concebido

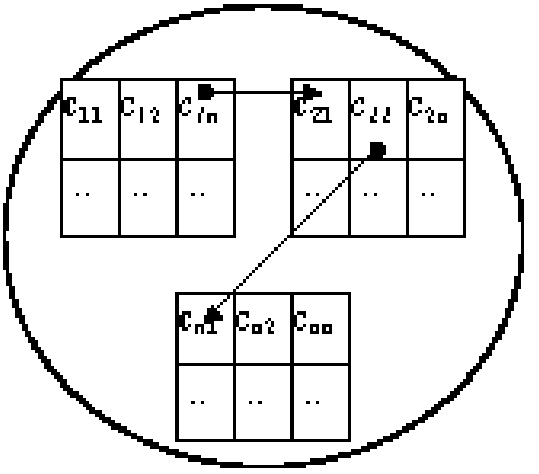

**Figura 1. Modelo relacional**

originalmente, utiliza el álgebra y cálculo relacional como lenguajes para ejecutar diversas operaciones sobre las tablas, para obtener otras tablas resultantes. La operación de selección genera una tabla con las mismas columnas que la tabla original, pero incluyendo únicamente los registros que cumplan una condición dada. La operación de proyección genera una tabla con las columnas elegidas de una tabla indicada. La combinación de la selección y la proyección permite seleccionar algunos registros de una tabla y sólo algunas columnas de dichos registros. La operación de reunión genera una tabla con registros "más anchos", que incorporan columnas de dos tablas relacionadas por llaves foráneas, cada registro con su registro relacionado correspondiente.

Las operaciones del álgebra relacional, y otros conceptos, como funciones y alias, fueron incorporados en los lenguajes de consultas para bases de datos como SQL (Structured Query Language). Este lenguaje tiene enunciados que permiten la definición de la base de datos DDL (Data Definition Language) y enunciados que permiten aplicar las operaciones del álgebra relacional para manipular los datos DML (Data Manipulation Language).

El lenguaje SQL tiene la gran ventaja que permite especificar las manipulaciones de datos (consultas, modificaciones, etc.) de manera declarativa y con un alto nivel de abstracción, dejando los detalles de su implementación al sistema administrador o motor de bases de datos (DBMS). Además, la posibilidad de declarar vistas, que son consultas preestablecidas, cuyo resultado es visto como una nueva tabla que permite subir aún más el nivel de abstracción.

Por otra parte, el modelo relacional es limitado por su bajo nivel de abstracción en aspectos de modelado propiamente dichos y por no capturar la parte dinámica o de comportamiento. Las tablas, atributos y llaves foráneas son elementos de bajo nivel de abstracción y no permiten representar aspectos dinámicos del sistema.

La introducción del modelo entidad-relación ER [CHE1976] mejoró el nivel de abstracción del modelo relacional original. La aparición posterior del modelo orientado a objetos mejoró no sólo el nivel de abstracción sino que superó la limitante en cuanto al modelado de los aspectos dinámicos o de comportamiento de los sistemas.

## 2. El modelo procedural estructurado y los lenguajes de alto nivel

El modelo procedural o imperativo tiene su origen a fines de los años 50 e inicios de los 60, con el advenimiento de los llamados lenguajes de alto nivel, como Fortran, Cobol y Algol. Luego, a principios de los 70, evolucionó al modelo estructurado con trabajos como los de Dijstra, Wirth y Hoare, introduciendo lenguajes como Algol-W, Pascal y C.

El modelo procedural estructurado podría expresarse de la siguiente manera:

> *Un programa está estructurado como un conjunto de funciones o subrutinas, donde unas invocan a otras, siguiendo, usualmente, una descomposición jerárquica en la cual funciones más generales invocan a otras más específicas y así sucesivamente. La función más general o función principal representa el programa como un todo. Cada función tiene sus datos locales y todas podrían compartir datos globales. Cuando una función invoca a otra pasa como parámetros, y recibe como resultado, estructuras de datos (registros, arreglos, etc.) con la información respectiva*

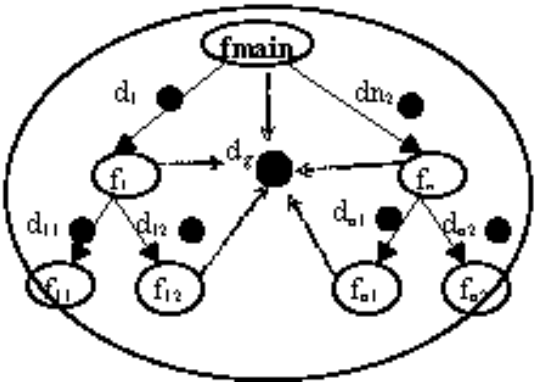


**Figura 2. Modelo procedural**

[LOU2004].

Con este enfoque estructurado el desarrollador piensa y modela los programas y sistemas como jerarquías de funciones y de datos. Por ejemplo, podría utilizar la técnica de DFD (Diagramas de Flujo de Datos) como herramienta de modelaje. Luego, utilizando lenguajes estructurados como C, Ada y Pascal, implementa dicho modelo (cada función y tipo de datos) y lo manipula, indicando cuál función invoca a cuál otra y cómo los datos fluyen entre funciones.

En este modelo, la abstracción puede darse tanto en el nivel de funciones como en el nivel de datos. Una función más general abstrae un proceso, ocultando los detalles de las funciones más específicas que invoca (más la lógica propia que coordina tales invocaciones). Por su parte, una estructura de datos usualmente abstrae un concepto, ocultando los datos detallados que la caracterizan.

Los lenguajes procedurales o imperativos, donde el lenguaje C constituye uno de sus más conocidos representantes, deben tener las construcciones sintácticas para expresar todos estos elementos que caracterizan el modelo: funciones, función principal (*main*), paso de parámetros por valor y/o por referencia, estructuras de datos (*struct*), definición de nuevos tipos de datos (*typedef*), etc.

Una de las debilidades de este modelo es que los elementos a manipular, utilizando el lenguaje, son de muy bajo nivel (funciones y datos). Esto genera dos problemas. Desde el punto de vista de implementación se requiere mucho trabajo aún para realizar tareas sencillas. Desde el punto de vista de modelado limita o restringe el pensamiento del desarrollador, pues, como se mencionó al inicio, normalmente nuestro pensamiento está moldeado por la visión, paradigma o modelo del "mundo" que tengamos.

Otra debilidad de este modelo es que representa de manera independiente las funciones y los datos, por lo tanto, es el desarrollador quien debe mantener la asociación entre cada función y los datos que ésta manipula. Un avance en la solución de esta debilidad lo representó, en su momento, la introducción de los tipos de datos abstractos. La idea era mantener juntos el tipo de datos, definidos por el desarrollador, y las funciones que manipulan dicho tipo de datos. Así se podía hablar del tipo pila, cola, etc. Para lograr mantenerlos juntos se utilizan, ya fueran recursos que el lenguaje ofrecía, como los paquetes de Ada, los módulos de Modula y las unidades de Pascal, o la disciplina del desarrollador, como en el caso de los módulos [.h+.c] de C. Los tipos de datos abstractos pueden verse como un paso en la evolución hacia la programación orientada a objetos.

## 3. El modelo orientado a objetos y el lenguaje UML

El modelo orientado a objetos cobra fuerza en los años 80 con lenguajes como SmallTalk y C++, que tomaban el concepto de clase del lenguaje Simula67 para sustituir los mecanismos de tipos de datos abstractos (ADT) de lenguajes como Ada, Modula y Pascal. Luego, en el año 1995, surgió Java como un fenómeno de crecimiento muy rápido asociado también a la Internet y en el 2000 Microsoft introdujo, con .Net, lenguajes con total soporte a la orientación a objetos como C# y VB.Net.

En el modelo orientado a objetos,

*un programa o sistema se concibe como un conjunto de objetos, cada uno con estado (datos) y comportamiento (funciones) propios, que interactúan unos con otros enviándose mensajes o peticiones para completar las diversas tareas que conforman el*

*programa o sistema. Los objetos se crean a partir de clases que definen la estructura y funcionalidad de todos los objetos derivados de ellas. Este modo de comunicación entre objetos simula la forma en que las personas, máquinas, etc. colaboran en el mundo real para completar los procesos* [LOU2004].

En el paradigma de orientación a objetos, el desarrollador, antes de pensar en las tareas o procesos específicos que conlleva el sistema, debe identificar y modelar cada concepto involucrado

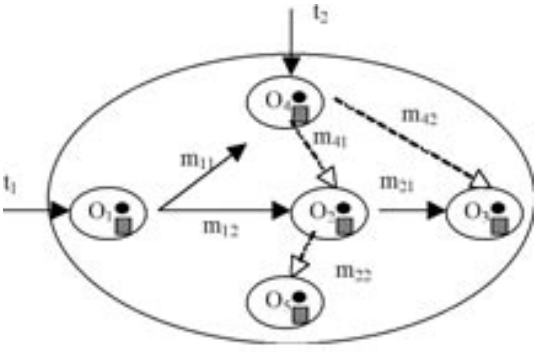


**Figura 3. Modelo orientado a objetos**

en el dominio del problema y sus relaciones con los demás conceptos. Los conceptos los modela como clases y las relaciones según su tipo las puede modelar como asociaciones, agregaciones, composiciones, herencias, etc. Posteriormente, una vez modelada esta infraestructura o red de conceptos interconectados podrá modelar, en un nivel más alto de abstracción, las tareas o procesos. Para ello identificará cuáles conceptos participan en cada tarea y la secuencia de cómo ellos colaboran para completarla, indicando las responsabilidades de cada uno. Una vez modeladas todas las tareas podrá hacerse una "suma" de las responsabilidades totales de cada concepto, identificando así cuáles comportamientos deberá exhibir.

Desde esta perspectiva orientada a objetos, se piensa en los sistemas como redes de objetos interactuantes. Este modelo es mucho más amplio y le da mayor libertad de pensamiento y de modelado al desarrollador. Le permite trabajar en un nivel de abstracción más alto, alejándose del computador y sus características físicas y acercándose al mundo real.

En orientación a objetos, la abstracción se manifiesta, pues cada clase es una abstracción de un concepto del mundo real. Además, como se mencionó antes, una vez formada esta infraestructura o red de conceptos puede pasarse a otro nivel de abstracción, para modelar cada tarea o proceso del sistema como una colaboración entre objetos.

Respecto del lenguaje, en la actualidad el más usado para expresar modelos orientados a objetos es UML (Unified Modeling Language). Este es un lenguaje gráfico, en el que pueden representarse los conceptos (clases), las relaciones entre ellos, las colaboraciones entre objetos, etc. UML nació en el año 1995, fruto del acuerdo entre varios autores (Booch, Rumbaugh y Jacobson) que tenían diversas propuestas y que decidieron unificarlas. Por su parte, en el nivel de implementación existen muchos lenguajes de programación orientados a objetos, tales como C++, Java, etc.

De un modelo orientado a objetos, expresado en UML, puede derivarse, al menos parcialmente, un modelo de implementación orientado a objetos y un modelo relacional de datos.

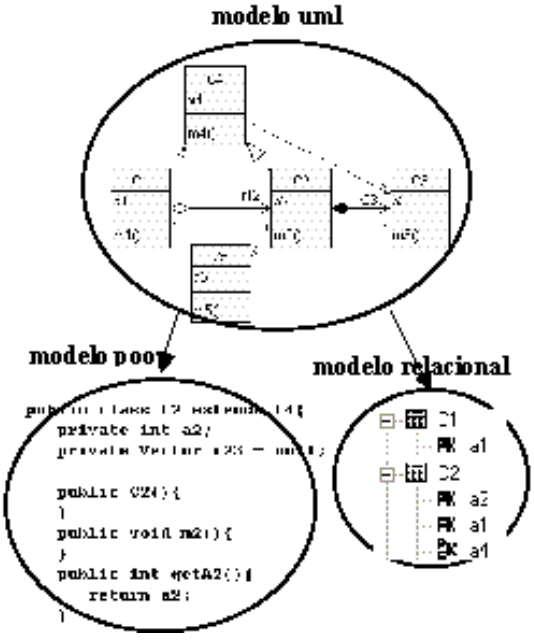


**Figura 4. Transformación entre modelos**

El modelo orientado a objetos tiene grandes virtudes, pero hay también algunas consideraciones. Por ejemplo, prescindió del concepto de módulo que poseía el modelo procedural. La introducción del modelo de componentes y servicios ha ayudado a solventar esto. Por otro lado, al tener el modelo de orientación a objetos un alto grado de granularidad (muchas clases) no resulta fácil aplicar características generales (como *loggin*), pues habría que incluirlas en cada clase. En respuesta a esto se ha estado desarrollando el concepto de orientación a aspectos.

## CONCLUSIÓN

Definitivamente, el tener una visión de modelo en el desarrollo de software, esto es, estar claro de que antes de construir hay que modelar y que se debe comprender el metamodelo o paradigma que estamos usando, es hoy una exigencia para los ingenieros de software, como lo es en otras áreas del quehacer humano. Con frecuencia en la formación universitaria se enseñan muchos lenguajes, técnicas y tecnologías y tal vez se subestima la importancia del modelado y de la comprensión de los metamodelos con que se trabaja. De igual modo, en el ejercicio profesional, y posiblemente debido a ese desenfoque en la formación, es frecuente que los desarrolladores intenten aprender y aplicar herramientas, lenguajes y técnicas sin dedicar primero el espacio requerido a comprender el modelo que subyace a ellos.

Ahora es claro que, detrás de tecnologías tan usuales como las de bases de datos, los lenguajes procedurales y el diseño y programación orientados a objetos, existen modelos en los que éstas se fundamentan: modelo relacional, modelo procedural y modelo orientado a objetos, respectivamente. Asimismo, es evidente que junto a cada modelo hay uno o varios lenguajes asociados: SQL, lenguajes estructurados, UML, lenguajes de programación orientada a objetos, etc. También resulta clara la importancia que la abstracción tiene en el proceso de desarrollo de software y cómo se expresa en cada modelo. Particular interés tiene el hecho de que un buen dominio de varios modelos nos permitirá combinarlos adecuadamente, al ir bajando de mayores a menores niveles de abstracción, utilizando en cada nivel el modelo o los modelos adecuados.

Se recomienda que en la formación de los desarrolladores de software se enfatice este enfoque de modelos y que se dedique el tiempo requerido para reflexionar acerca de las capacidades y limitaciones de cada uno.

A pesar de que las tecnologías de software cambian día con día y año con año, no sucede lo mismo con la aparición de nuevos modelos o paradigmas que impacten de manera significativa esta área, como lo han hecho los modelos: relacional, estructurado y orientado a objetos. Habrá que estar atentos a las nuevas propuestas que surjan y evaluar el impacto que puedan tener.

## REFERENCIAS

[CHE1976] Chen, Peter. "The Entity-Relationship Model - Toward a Unified View of Data". *ACM Transactions of Database Systems (TODS)*, Vol. 1, Nº 1, 1976, pp. 9-36.

[COD1970] Codd, E.F. "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM*, Vol. 13, Nº 6, June 1970, pp. 377-387.

[ELM2000] Elmasri, Ramez y B. Navathe. *Sistemas de Bases de Datos*. 2da. edición. Addison Wesley. México. 2000.

[LOU2004] Louden, Kenneth. *Lenguajes de Programación, Principios y Práctica*. Internation Thomson Editores. México. 2004.