Computación y Sistemas Vol. 4 No.4 pp. 314 - 326
© 2001, CIC - IPN. *ISSN 1405-5546* Impreso en México

Especificación de Temporalidad en Entornos Automáticos de Producción de Software a Partir de Modelos Conceptuales Objetuales

Carlos Meneses[1], Oscar Pastor[2], Juan Carlos Molina[2] y Emilio Insfrán[2]
[1]Programa de Ingeniería de Sistemas y Computación (PISC)
Universidad Tecnológica de Pereira, La Julita Pereira, Colombia
[2]Departament de Sistemes Informàtics i Computació (DSIC)
Universitat Politècnica de València, Camí de Vera s/n 46022
València, España. Teléfono:+34-96 387 7734
e-mails: {einsfran, opastor, cmeneses, juamoud}@dsic.upv.es

Artículo recibido el 2 de diciembre, 1999 ; aceptado el 21 de febrero, 2001

Resumen

Para poder representar de manera adecuada los aspectos estáticos y dinámicos de un sistema acorde con las necesidades de modelización del mundo real, es importante incluir la expresividad temporal cuya especificación permita obtener un modelo funcionalmente equivalente al modelo conceptual del sistema estudiado.

En el contexto del proyecto IDEAS (Ingeniería de Ambientes de Software) financiado por la CYTED, y en particular en su tarea titulada "Métodos para especificación de transacciones en fase de modelado conceptual en ambientes OO", el grupo de investigación DSIC-UPV (Valencia, España) trabaja en la definición de un ambiente de producción automática de software basado en modelos conceptuales temporales y orientado a la especificación y diseño de transacciones, a partir de su aproximación metodológica OO-Method. El objetivo de este artículo es presentar como se incluyen adecuadamente en OO-Method las propiedades que permitan capturar la expresividad temporal de modelos conceptuales orientados a objetos, preservando la generación automática del correspondiente producto final software.

Palabras Clave: orientación a objetos, expresividad temporal, análisis de requisitos, generación automática de código, modelado conceptual.

1 Introducción

El ciclo de vida clásico para la producción de software ofrece un enfoque secuencial partiendo de los aspectos más abstractos a los más concretos (ingeniería del sistema, análisis, diseño, codificación, prueba y mantenimiento). Para las últimas etapas (a partir de la codificación) se cuenta actualmente con herramientas automáticas, que ayudan a asegurar la calidad del producto resultante. Lo mismo no acontece con las etapas previas, que son fuente de errores e ineficiencias, que afectan la calidad y los costes, perjudicando la productividad del producto desarrollado.

Cada vez es más notable la inversión en esfuerzos para la elaboración de herramientas basadas en métodos formales, lo cual realza una nueva disciplina (la Ingeniería de Requerimientos) que basa la especificación de requerimientos en modelos conceptuales, a partir de la información que es relevante para la aplicación que se quiere representar.

En el Modelado y Diseño Orientados a Objetos subyace una forma de pensar en la que los problemas se resuelven a partir de modelos basados en conceptos del mundo real. Este paradigma se considera, por su expresividad, el más adecuado para abordar el proceso de modelado conceptual. En este proceso, es importante representar todos los aspectos, no sólo estáticos y dinámicos, sino también su evolución temporal en el dominio de la aplicación, sin tener en cuenta la implementación.

La potencialidad de la información temporal se percibe cuando se verifica la posibilidad de consultar datos históricos, estados del sistema pasados y actuales que permiten analizar y planificar situaciones futuras para la toma de decisiones. La especificación de aspectos temporales en sistemas de información posibilita el manejo de datos temporizados y de tareas que pueden tener precondiciones temporales asocia-

das a su ejecución o presentar interacciones temporales con otras para el control de su ejecución; además, no se puede dejar de considerar la importancia que tiene la correcta representación de las condiciones de integridad temporal en la futura base de datos.

Para tratar el problema de cómo especificar correctamente esa información temporal, diversos modelos de datos temporales se han presentado (Edelweiss *et.al,* 1994; Özsoyoglu y Snodgrass, 1995; Tansel *et al.*, 1993), constituyendo la mayoría extensiones temporales de modelos ya existentes. Entre estas podemos destacar las basadas en modelos relacionales (Clifford y Crocker 1987; Gadia, 1988; Lorentzos y Johnson, 1988; Navathe y Ahmed, 1989; Sarda, 1990; Snodgrass, 1987; Tansel, 1986), E-R (Elmasri y Kouramajian, 1992; Elmasri *et al.*, 1993; Loucopoulos *et. al.*, 1991; Tauzovich, 1991; Lee y Elmasri, 1998) y las basadas en modelos orientados a objetos (Edelweiss *et al.*, 1993; Kafer y Schoning, 1992; Rose y Segev, 1991; Su y Chen, 1991; Wuu, 1993). Recientemente, esfuerzos importantes en el área del modelado temporal han sido los realizados en la definición del lenguaje de consulta temporal TSQL2 (Snodgrass, 1995), que pretende ser propuesta para manipulación de bases de datos relacionales temporales. En cualquier caso, no son frecuentes los trabajos que traten de forma unificada aspectos de especificación de propiedades temporales (espacio del problema) junto con aspectos de implementación (espacio de la solución).

En ese contexto, el objetivo de este artículo es claro: se trata de enriquecer un entorno de generación automática de código a partir de esquemas conceptuales objetuales, con las primitivas que permitan especificar adecuadamente la información temporal comentada anteriormente. Para ello, el método de producción de software elegido es OO-Method, siendo las ideas presentadas aplicables a cualquier método de producción de software objetual, basado en, por ejemplo, estándares de uso generalizado como UML. Ese enriquecimiento debe además incluir los mecanismos de conversión de las primitivas de especificación temporal introducidas en el modelo, en sus correspondientes representaciones software para preservar el carácter de generación de código a partir de modelos conceptuales característico de propuestas como OO-Method.

De acuerdo con lo anterior, la estructura del artículo es como sigue: la sección 2 presenta una breve introducción de la propuesta OO-Method y el lenguaje de especificación OASIS que es la base de OO-Method. La sección 3 permite entender los mecanismos de representación de las propiedades temporales en el modelado conceptual (espacio del problema). En la sección 4 se indica como es la implementación de la expresividad temporal en los entornos de desarrollo utilizados por los generadores de código implementados en OO-Method (espacio de la solución). En la sección final se presentan algunas conclusiones y el trabajo futuro.

2 OO-Method

OO-Method (Pastor, 1992; Pastor *et al.*, 1996; Pastor *et. al.*, 1997) es una metodología OO para la producción automática de software basada en técnicas de especificación formales y en el uso de modelos gráficos similares a los empleados por metodologías co~vencionales (OMT, Booch,...). OO-Method ofrece un marco riguroso para la especificación de sistemas de información que incluye una potente y sencilla notación gráfica para tratar con la fase de análisis, *OASIS* (Pastor y Ramos, 1995) como un lenguaje de especificación formal y OO que constituye el repositorio de alto nivel del sistema, y además, la definición de un preciso modelo de ejecución que guia la fase de implementación. En el proceso de construcción de software se definen dos fases principales:

1. *Modelado Conceptual.* Los modelos de análisis (objetos, dinámico y funcional) gráficos recogen las propiedades relevantes que definen el sistema a desarrollar sin tener en cuenta la implementación. Con esta descripción, se genera automáticamente una especificación formal del sistema OO en OASIS, que constituye un repositorio completo y formal de alto nivel del sistema.
2. Aplicación de un *Modelo de Ejecución.* La especificación formal es la fuente del modelo de ejecución, definido de forma precisa y que determina de manera automática las características del sistema dependientes de la implementación (interfase de usuario, control de acceso, activación de servicios, consulta y manipulación de información, etc.) y que no forman parte directa del espacio del problema.

2.1 OASIS: Un modelo Formal Orientado a Objetos

OO-Method se ha creado sobre las bases formales de OASIS, un lenguaje formal de especificación de sistemas de información OO. En una especificación OASIS, una clase es un patrón o conjunto de propiedades que comparten todos sus ejemplares. Este patrón tiene un nombre y permite la declaración de un mecanismo de identificación. La signatura de la clase incluye atributos, eventos y un conjunto de fórmulas de la lógica dinámica que nos permiten describir el resto de sus propiedades: *restricciones* de integridad estáticas y dinámicas, *evaluaciones* que especifican como cambian los atributos debido a la ocurrencia de eventos, *derivaciones* que especifican el valor de un atributo derivado en función de otros, *precondiciones* que establecen condiciones que deben satisfacerse para activar un servicio y *disparos* que provocan la activación espontánea de un servicio como consecuencia de la satisfacción de una condición.

Además, un objeto puede definirse como un *proceso observable*, por lo que la especificación de la clase se enriquece

con un álgebra de procesos básica, que permite declarar las vidas posibles de un objeto como términos de este álgebra, cuyos elementos son eventos y transacciones, combinados con operadores de alternativa y secuencia.

2.2 Modelo Conceptual

Se empieza la fase de análisis con la construcción de tres modelos (Objetos, Dinámico y Funcional) que recogen las propiedades relevantes (determinado por las distintas secciones de especificación en OASIS) de un sistema de información y describen la sociedad de objetos desde tres puntos de vista complementarios y dentro de un riguroso marco orientado a objetos.

El ***Modelo de Objetos*** es descrito gráficamente por un Diagrama de Configuración de Clases (DCC) que muestra la estructura y comportamiento de todas las clases identificadas en el dominio del problema así como sus relaciones. Una clase se representa gráficamente como una caja dividida en secciones donde se recoge información sobre sus atributos y servicios. Los servicios se declaran especificando su nombre y argumentos, distinguiendo entre las transacciones y los eventos de creación, borrado y los eventos compartidos (Pastor *et al.*, 1996). Para el tratamiento de la complejidad se pueden definir relaciones estructurales en términos de agregación (parte-de) y herencia (es-un).

La relación de agregación entre clases, incluye información sobre cardinalidades (mínimas y máximas) que determinan cuantos objetos componentes pueden estar relacionados con un objeto compuesto e inversamente. Además, una agregación debe ser indicada como inclusiva/referencial o estática/dinámica, cuya explicación detallada puede encontrase en Pastor *et al.* (1996).

El ***Modelo Dinámico*** especifica los aspectos relacionados con el control, vidas posibles, secuencia de servicios e interacción entre objetos. Se representa por tipos de diagramas: Diagramas de Transición de Estados (DTE) y Diagrama de Interacción de Objetos (DIO).

Los *Diagramas de Transición de Estados (DTE)* describen el comportamiento de objetos estableciendo *vidas posibles*. Una *vida válida* es una secuencia correcta de estados que caracterizan un comportamiento válido para todos los objetos de una clase. Los estados denotan situaciones en las que pueden encontrarse los objetos como consecuencia de la ocurrencia de servicios relevantes. Las transiciones representan los cambios de estado válidos. La utilización de condiciones permite eliminar el indeterminismo y crea restricciones en las transiciones de estado.

Los *Diagramas de Interacción de Objetos (DIO)* modelan gráficamente la interacción entre objetos y especifican dos tipos de interacciones básicas. Los *disparos*, que son servicios de una clase que se activan de forma automática cuando se satisface una condición en un objeto dicha clase y las *interacciones globales*, que son *transacciones* compuestas por servicios de clases diferentes.

El ***Modelo Funcional*** captura la semántica asociada a los cambios de estado de un objeto como consecuencia de la ocurrencia de un evento. El valor de cada atributo se puede modificar dependiendo de la acción ocurrida, de los argumentos del evento y/o del estado actual del objeto.

La especificación del cambio de estado viene determinado por la categorización de atributos (Pastor *et. al.*, 1997) entre un conjunto predefinido de *categorías*, las cuales indican qué información se necesita para determinar cómo cambia el valor del atributo ante la ocurrencia de determinados eventos.

2.3 El Modelo de Ejecución

El *modelo de ejecución* es un patrón abstracto de comportamiento, aplicable a cualquier modelo conceptual construido siguiendo la estrategia propuesta por OO-Method. La idea consiste en dar una visión del modelo que determinará directamente el patrón de programación a seguir cuando un desarrollador (o una herramienta CASE) aborde la fase de implementación y se definan las características del producto software final en términos del control de acceso a usuarios, activación de servicios, interfaz de usuarios, etc. Este modelo tiene tres fases esenciales:

- *Control de acceso*: en primer lugar, el objeto que desee conectarse al sistema deberá identificarse como miembro de la sociedad de objetos.
- *Vista del sistema*: una vez se ha conectado un usuario (u otro objeto del sistema), tendrá una visión clara de la sociedad de objetos en términos de qué clases de objetos puede ver, qué servicios puede activar y qué atributos puede consultar.
- *Activación de servicios*: por último, el objeto deberá ser capaz de activar cualquier servicio disponible y de realizar las observaciones pertinentes.

Cualquier ***petición de servicio*** se caracterizará por la siguiente secuencia de acciones: identificación del objeto servidor, introducción de argumentos del servicio a activar, validación de la transición entre estados, satisfacción de precondiciones, realización de las evaluaciones (modificación del estado del objeto), comprobación de las restricciones de integridad en el nuevo estado y comprobación de las relaciones de disparo. Estos pasos, claramente definidos en Pastor *et. al.* (1997), guían el proceso de implementación asegurando la equivalencia funcional entre la descripción del sistema recogida en el modelo conceptual y su esquema en un entorno de programación.

3 Espacio del problema: Temporalidad en los Modelos Conceptuales OO-Method

Marcas temporales. Se pueden tener cuatro formas distintas de marcas (*primitivas temporales*), para especificación temporal (Jensen, 1994):

- *Instante temporal*. Representa un punto en el tiempo. Por ejemplo, el instante en el que inicia un partido de fútbol. En este caso solo se registra un dato temporal.
- *Intervalo temporal*. Definido por dos instantes temporales, indicando el inicio y el fin del intervalo. Por ejemplo, el intervalo del último contrato de un empleado.
- *Elemento temporal*. Constituido por la unión finita de intervalos temporales. Por ejemplo, los diferentes intervalos de tiempo en los que un empleado ha sido contratado para la empresa.
- *Duración temporal*. Corresponde a un periodo de tiempo sin tener en cuenta los instantes en que inicia y termina. Puede ser de dos tipos, dependiendo del contexto en el que sea definido: fija y variable. Una duración fija es independiente del contexto de su definición (por ejemplo una hora corresponde siempre a 60 minutos) y una duración variable depende del contexto (por ejemplo un mes puede tener de 28 a 31 días) (Edelweiss, 1998).

En el modelo presentado en este documento, para agilizar y facilitar las consultas, se utiliza en los objetos históricos una especificación por *intervalo temporal* (que permite usar bases de datos relacionales que cumplan la 3FN –tercera forma normal -; y que además, proporciona información explícita y soporta la noción de *elemento temporal* usando un conjunto finito de instancias), excepto en la expresividad de las relaciones de agente en las que, por modelar cambios de estado que ocurren en un instante de tiempo, se utiliza el *instante temporal* (proporciona información implícita). El tipo de *marca temporal* empleada para la especificación de un *instante temporal* depende del entorno del sistema, de la decisión de implementación y de la *granularidad* (número interno secuencial, fecha y hora con sus posibles combinaciones de año, mes, día, horas, minutos, segundos, centésimas de segundo, etc.) ofrecida en el modelo de datos utilizado.

En cuanto al *significado de la marca temporal*, pueden emplearse:

- *Tiempo de transacción*. Tiempo en el que la información fue introducida en la base de datos que implementa la aplicación.
- *Tiempo de validez*. Tiempo en que la información es válida en la realidad modelada (corresponde al tiempo de existencia o vida de un objeto en el sistema).
- *Ambos*.

El *tiempo de validez* es usado en OO-Method, por ser relevante en la fase de modelo conceptual, en la que no deben considerarse aspectos relativos de la implementación física (excepto nuevamente en las relaciones de agente temporizadas que usan el tiempo de transacción para precisar el instante en el que fue activado el evento).

Niveles de Especificación. La especificación de la información temporal en un modelo orientado a objetos se puede hacer bajo cinco diferentes niveles: atributos, objetos, relaciones de agregación entre objetos, relaciones de herencia entre objetos y agentes que actúan como activadores de servicios. En una aplicación los valores tomados por los datos bajo estos niveles de especificación pueden cambiar con el tiempo. Cuando es necesario representar estos cambios y conservar la información histórica, se introduce una representación que indica un comportamiento temporal.

En un objeto, no todos sus atributos son temporizados, por lo cual se debe especificar en el modelado conceptual, cuáles lo son. De los objetos se puede requerir información temporal de cuando se creó, durante qué periodos existió (estuvo activo) para el sistema, cuando dejó de existir, etc. De las relaciones de agregación se pueden tener variaciones de su cardinalidad en el tiempo o variaciones en su valor; en el fondo, se trata de atributos objeto-valuados. Para las relaciones de herencia, algunos objetos pueden heredar propiedades durante diferentes intervalos en el tiempo. De los agentes que actúan como activadores de servicios debe ser posible conocer durante qué periodos históricos un agente ha activado un servicio, así como cuándo y cuáles servicios ejecutó un agente o cuándo y cuáles agentes ejecutaron un servicio.

La importancia de incluir la declaración de temporalidad radica en que gran parte de los requerimientos de los sistemas involucran el manejo de información temporal e histórica con lo cual se le permite al analista especificar, por ejemplo, que algunos atributos de una clase, una clase, una relación de agregación, de herencia o de agente, tengan asociado un *registro histórico* de todas sus variaciones en el tiempo. Veamos como se captura en OO-Method la expresividad temporal en cada uno de los cinco niveles de especificación determinados:

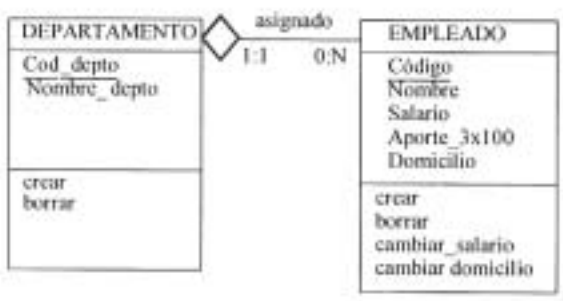

Figura 3.1. Ejemplo de Modelo de Objetos

3.1 Clases

Una *clase temporizada* es aquella en la que nos interesa conservar la información histórica de los distintos intervalos temporales en los que cada objeto de la clase en cuestión ha existido. La información asociada a *clases de objetos temporizados* indica una evolución temporal del objeto como un todo – cuando se creó, cuándo deja de existir, periodos de suspensión de actividad, etc.

Cuando un objeto de una *clase temporizada* sea eliminado, no desaparecerá de la base de datos utilizada, sino que se cerrará adecuadamente su intervalo de existencia. Para el ejemplo de la figura 3.1, si una clase *empleado* se declara temporizada, estaremos especificando que una persona puede aparecer como instancia de la clase empleado asociada a distintos intervalos de existencia.

Existencia	*Código*	*Nombre*	*Salario*	...
[3,10] ∪ [20,null]	e1	José Pérez	180 [3,6] ∪ 225 [7,10] ∪ 270 [20,null]	
[8,null]	e2	Manuel Gómez	175 [8,20] ∪ 185 [21,null]	
[7,35]	e3	María López	160 [7,20] ∪ 195 [21,35]	
...	...	...	...	

Tabla 3.1 Clase y atributo *salario* de la clase *empleado* temporizados

3.2 Atributos

Los *atributos temporizados*, son aquellos atributos que cambian de valor a lo largo de la vida del objeto, pero cuya secuencia de valores es necesario almacenar. Cada uno de los valores diferentes que pueden tener los *atributos temporizados* a lo largo de su evolución, tiene su marca temporal correspondiente. No todos los atributos de un objeto necesitan ser temporizados. Es importante que un modelo temporal permita la *coexistencia* de atributos temporizados y otros que no tengan asociada información temporal (no temporizados).

Cada intervalo de existencia asociado a un objeto de una clase temporizada puede, adicionalmente, tener relacionados uno o más intervalos de existencia asociados a los valores de los atributos que han sido declarados como temporizados. En el ejemplo anterior, si el atributo *salario* está vinculado a una dimensión temporal, un empleado en un intervalo de existencia dado puede haber tenido diferentes salarios (asumiendo que, como decíamos antes, salario es un atributo temporizado) en sus diferentes periodos de existencia como se ilustra en la Tabla 3.1.

3.3 Relaciones de Agregación

En las *relaciones de agregación temporizadas* se mantiene una historia de las agregaciones en el tiempo. En este caso, las cardinalidades pasan a ser restricciones referentes a un instante en el tiempo. La distinción de las *relaciones de agregación temporizadas* permite guardar información sobre los intervalos de tiempo en los que un objeto está compuesto (o forma parte) de otros. Por ejemplo, considerando como temporizada la relación de agregación entre las clases *departamento* y *empleado*, los objetos de la clase *departamento* están compuestos por 0 o N objetos de la clase *empleado* y *un* objeto *empleado* forma parte de *un* objeto departamento en un instante determinado. Una agregación en OO-Method puede ser vista como un atributo objeto-valuado, y tratada conceptualmente como un atributo temporizado más.

3.4 Relaciones de Herencia

Por medio de las *relaciones de herencia temporizadas*, es posible conservar un registro histórico de los intervalos de existencia asociados a los roles que asumen los objetos que forman parte de la clase especializada. El modelo temporal debe permitir la *coexistencia* de algunas especializaciones temporizadas y otras que no lo sean. Por ejemplo, los objetos de la clase *empleado* de acuerdo al cargo que desempeñan se clasifican como *operativos* o *administrativos* (pudiendo tener para cada especialidad atributos y servicios propios); podríamos estar interesados en mantener la información histórica de quienes y cuando han ocupado cargos administrativos, pero sin que se requiera la información histórica de los que han ocupado cargos operativos.

3.5 Relaciones de Agentes

De las *relaciones temporales para los agentes* que actúan como activadores de servicios, la especificación del modelo de objetos debe permitir conservar una historia de los instantes de tiempo en los que un agente hace que se active un servicio o transacción. Esta información histórica corresponde a un "log" que permitirá a los usuarios controlar la ejecución del sistema especificado. Por ejemplo, si se especifica como temporal la relación de agente entre la clase *empleado* y un servicio *cambio_salario*, cada vez que se ejecuta este servicio queda un registro histórico que indica el agente y el servicio involucrados, y si tuvo éxito la ejecución. Esto facilita conocer cuándo un agente ejecutó (o intentó ejecutar) un servicio, o cuáles servicios ha ejecutado, así como cuándo y cuáles agentes ejecutaron un servicio.

4 Espacio de la Solución. Tratamiento de las Propiedades Temporales en el Modelo de Ejecución

La **especificación de la expresividad temporal en el Modelo de Objetos** de OO-Method (ver Figura 3.2) se representa mediante una marca (reloj de arena) ubicado junto al elemento temporal. En el caso de clases, atributos y relaciones de agregación la marca está junto al nombre. Para relaciones de agente, la marca está sobre la línea punteada que especifica esta relación. En el caso de herencia, la marca está sobre la línea continua propia de la especialización.

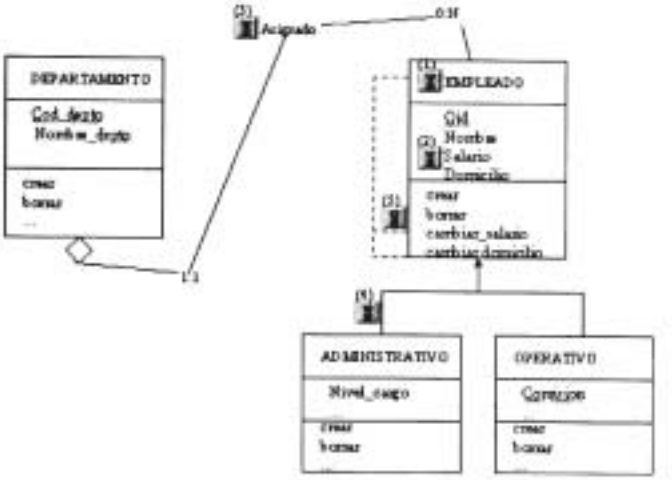

Fig 3.2 *Especificación de Temporalidad en OO Method*. (1) Clases. (2) Atributos. (3) Relaciones de Agregación. (4) Herencia (5) Agentes Activadores de Servicio

Para que la evolución temporal del sistema sea representada, el modelo de datos utilizado debe permitir la especificación para cada uno de los niveles anteriores durante la fase de modelado conceptual.

La principal característica de OO-Method es su capacidad de generar de forma automática la implementación de aplicaciones en entornos industriales de producción de software a partir del modelo de ejecución abstracto presentado anteriormente. Estas aplicaciones utilizan un Sistema de Gestión de Bases de Datos Relacionales (SGBDR) como repositorio de datos, mientras que los servicios declarados en las clases se representan como procedimientos almacenados, métodos de clases o estructuras de programación similares (dependiendo del entorno de desarrollo usado para implementarlos).

El que OO-Method utilice en sus entornos de producción SGBDRs como contenedores de datos, nos orienta hacia cómo representar apropiadamente y de forma automática esa expresividad temporal introducida ahora en los modelos conceptuales creados con OO-Method. Veamos a continuación cómo va a ser introducido el tratamiento de las características temporales en el modelo de ejecución y cómo va a afectar en OO-Method la generación automática de código.

La primera decisión que hay que tomar es si representar los datos de una clase temporizada en una única tabla con los datos actuales, o separarlos en tablas distintas. El hecho de utilizar una única tabla obliga a trabajar sobre los datos actuales a través de una vista en la que el atributo que representa al instante de modificación sea nulo y se tiene que modelar el manejo de conjunto de valores históricos para una instancia de un elemento temporal. Para evitar esta situación, vamos a elegir una representación basada en tablas diferentes para la información actual e histórica. Además, así también evitaremos problemas asociados a la normalización para el manejo de tablas y al tratamiento de la información faltante en las implementaciones relacionales actuales.

Los siguientes aspectos determinan cómo debe ser la representación software asociada en OO-Method a las características de manejo temporal introducidas en los modelos conceptuales del método:

4.1 Clases

Cada clase se representa como una tabla (llamada *tabla base*). Si la clase se especifica como *temporizada*, tendrá asociada a su tabla base una nueva "*tabla histórica*" que incluye las instancias eliminadas de la tabla base con la información de cierre del intervalo temporal de existencia de dichos objetos y las instancias actuales existentes en la tabla base con la información de apertura de existencia de los mismos.

La **tabla base** tiene la siguiente estructura:

(oid, {lista de atributos})

Donde, *oid* es el identificador del objeto y *lista de atributos*, contiene los demás atributos asociados a la tabla base.

La estructura de la **tabla histórica** es:

(oid, tiempo_inicio, tiempo_fin, {lista de atributos})

Donde *oid* es el identificador del objeto en la tabla base, *tiempo_inicio* referencia el instante en el que se abrió el intervalo temporal de existencia considerado, instanciado al crear el objeto en la tabla base. El atributo *tiempo_fin* representa el instante de cierre de dicho intervalo, y *lista de atributos* incluye el conjunto de atributos asociados al momento en el que se introduce esta tupla en la tabla histórica y que luego se modifican cuando se cierra el intervalo de existencia del objeto. Este instante será justamente el de ocurrencia del evento de destrucción declarado en la clase.

Código	*Nombre*	*Salario*
e1	José Pérez	270
e2	Manuel Gómez	185
...	...	...

Tabla 4.1. Tabla base de la clase Empleado

T_ini	*T_fin*	*Código*	*Nombre*	*Salario*
3	10	e1	José Pérez	225
7	35	e3	María López	195
8	Null	e2	Manuel Gómez	175
20	Null	e1	José Fernandez	270
...	...	...	...	...

Tabla 4.2. Tabla histórica de la clase Empleado

La llave primaria en la tabla histórica corresponde a *oid+tiempo_inicio*. Por cada llave foránea que referencia a una tabla histórica de otra clase temporizada, se agrega un atributo que contenga el "tiempo de inicio" de existencia de esa llave foránea para relacionar con la tabla histórica correspondiente.

La generación de una instancia en la tabla histórica, permite dos implementaciones:

- Que la implementación del servicio *new* de la clase implicada (en tabla base) sea la encargada de ejecutar el *insert* correspondiente en la tabla "histórica", y que la implementación del servicio *destroy* de la clase, sea la encargada de ejecutar la *actualización* correspondiente en la misma tabla "histórica".
- Que se defina un *disparo* ("trigger") asociado a la inserción y otro al borrado de tuplas de esa tabla base. Esta opción será la que se elegirá si el SGBDR permite la definición de disparos de la base de datos; de lo contrario, se debe usar la primera.

Los atributos derivados de una clase temporizada no aparecen en la estructura de la tabla base, pero sí en la estructura de la tabla histórica, con el fin de facilitar las consultas.

Para el ejemplo de la figura 3.1, las tablas bases resultantes tienen la siguiente estructura:

EMPLEADO:
(Código, nombre, salario, aporte_3x100, domicilio, fk_cod_depto)
DEPARTAMENTO:
(cod_depto, nombre_depto)

El atributo *fk_cod_depto* de *Empleado* es llave foránea y referencia a *Departamento*.

Con la clase *Departamento* no temporal, la tabla histórica para *Empleado* (temporal) es:

H_EMPLEADO:
(Código, tiempo_inicio, tiempo_fin, nombre, salario, aporte_3x100, domicilio, fk_cod_depto).

El atributo *fk_cod_depto* de la tabla H_Empleado no es llave foránea. Sólo indica cuál era el código del departamento para cada empleado sin garantizar que exista en la tabla base de Departamento.

Las tablas de la base de datos histórica poseen integridad referencial entre sí por medio de sus llaves foráneas, pero no se relacionan referenciando a las tablas base asociadas, pues no se garantiza la integridad. La razón es los objetos existen siempre en las tablas históricas pero no en las tablas base, de donde son eliminados al cerrarse su intervalo de existencia.

Si la clase *Departamento* se especifica como temporal, entonces se crea la tabla histórica correspondiente (*H_Departamento*) y se adiciona el atributo *tiempo_inicio_depto* en la tabla *H_Empleado*. Los atributos *fk_cod_depto* y *tiempo_inicio_depto* son llave foránea referenciando a la tabla *H_Departamento*.

4.2 Atributos

Cada atributo que se especifica como temporizado (espacio del problema), genera en la base de datos (espacio de la solución) la creación de una "*tabla histórica de atributo*" asociada a la tabla base de la clase en la que está definido el atributo y además, origina la creación de una "*tabla histórica de clase*" (si aún no existe) asociada a la misma tabla base de la clase.

La *tabla histórica de atributo* incluye instancias con la identificación del objeto, los valores históricos que ha tenido el atributo temporal para dicho objeto en la tabla base con el intervalo temporal de existencia de dichos valores, y los valores actuales existentes para cada objeto en la tabla base con la información de apertura de existencia de los mismos valores.

La estructura de la tabla base de la clase, no se ve afectada por la inclusión de atributos temporizados.

La estructura de la **tabla histórica de atributo** es:
(oid, inicio_oid, inicio_atributo, fin_atributo, atributo)

Donde *oid* es el identificador del objeto, *inicio_oid* es el instante a partir del cual se crea el objeto con ese oid, *inicio_atributo* referencia al instante en el que se abrió el intervalo temporal de existencia considerado (en el que se generó la tupla histórica para el atributo), instanciado al crear el objeto en la tabla base o al modificar el valor del atributo temporal en la tabla base, *fin_atributo* representa el instante de cierre de dicho intervalo, instanciado cada vez que cambia el valor del atributo o cuando se elimina el objeto en la tabla base y *atributo* corresponde al valor que toma el atributo, asociado al intervalo considerado.

En el ejemplo de la figura 3.1, si la relación de agregación "*asignado*" se especifica como temporizada, la tabla histórica debe incluir los intervalos (determinados por los atributos *inicio_depto_empleado* y *fin_depto_empleado*) en los que existió (y existe) la relación de asignación de cada empleado en cada uno de los departamentos de la empresa.

La generación de una instancia en la tabla histórica de agregación, permite dos implementaciones posibles:

- Que la implementación del servicio *new* de la clase en la que está definida la clave ajena y cada servicio que afecte a dicha clave ajena (de la relación de agregación definida como temporal) sea el encargado de ejecutar el *insert* correspondiente en la tabla "histórica de agregación", y que la implementación del servicio *destroy* de la clase y cada servicio que afecte a la clave ajena en cuestión, será la encargada de ejecutar la *actualización* (modificación correspondiente al cierre del intervalo para la relación de agregación) correspondiente en la misma tabla "histórica de agregación".

  Que se defina un *disparo* ("trigger") asociado a la creación del objeto y otro a la modificación de la clave ajena para abrir el intervalo temporal de la relación de agregación. Además, se debe definir un disparo (con lo cual se cierra dicho intervalo) para la modificación de la clave ajena en cuestión y el borrado del objeto en la tabla base. Esta opción será la que se elegirá, si el SGBDR permite la definición de disparos de la base de datos; de lo contrario, se debe usar la primera.

Cuando una relación de agregación se especifica como temporal, para garantizar la integridad referencial, se tratan automáticamente como temporales (aunque no hayan sido definidas como tales) las clases relacionadas.

Para el ejemplo, si se especifica la relación de agregación como temporal, entonces se tratan (aunque no hayan sido especificadas como tales) automáticamente las clases Empleado y Departamento como temporales, generando las tablas históricas correspondientes.

La clave primaria de la tabla histórica para una relación de agregación corresponde a los identificadores de las clases relacionadas y se le añade el tiempo de inicio del intervalo en el que es válida la relación de agregación. En el ejemplo anterior, si se especifica la relación de agregación como temporal, se crea la *tabla histórica*:

H_DEPTO_EMPLEADO:

(código, inicio_empleado, cod_depto, inicio_depto, inicio_depto_empleado, fin_depto_empleado)

Las claves ajenas *código+inicio_empleado* y *cod_depto+inicio_depto* referencian a la tablas históricas de *empleado* y *departamento* respectivamente.

4.4 Relaciones de Herencia

Para las relaciones de herencia temporizadas, la clase especializada se trata como temporal a nivel de modelado conceptual, y la tabla histórica correspondiente recogerá la información sobre los intervalos de tiempo en los que la relación de herencia existe, de la misma forma que presentamos anteriormente al estudiar la representación de clases temporizadas. Es decir, una relación de herencia que se especifique como temporal, trata automáticamente a la clase especializada como temporal (aunque no haya sido definida como tal) y se le da el mismo tratamiento de clase temporizada.

Cuando una relación de herencia se especifica como temporal, además de tratar la clase especializada como temporal, también se tratan como temporales (aunque no hayan sido definidas como tales) todas las clases que son ancestros en la línea jerárquica ascendente hasta la raíz del árbol de herencia. Esto se hace con el fin de conservar la información temporal del objeto como un todo y para mantener el esquema de la base de datos relacional normalizada.

Para el ejemplo anterior, si *Empleado* es una especialización de la superclase *Persona* y si se define como temporal la relación de herencia entre las clases *Empleado* y su especialización *Administrativo*, entonces se tratan automáticamente (aunque no hayan sido definidas como tales) las clases *Persona*, *Empleado* y *Administrativo*, y se generan las tablas históricas correspondientes.

4.5 Relaciones de Agente

La especificación de temporalidad (espacio del problema) en cada relación de *agente* que actúa como activador de un *servicio* origina en la base de datos (espacio de la solución) la creación de una *tabla histórica de agente* (o "*log*"), que contiene la información histórica por medio de la cual se puede conocer y controlar las actividades de ejecución de servicios. Como la ejecución de un servicio se considera un proceso atómico, la marca temporal usada es el instante temporal.

Para almacenar los datos históricos generados por las relaciones de agente temporales, se genera una tabla histórica ("log") por cada clase agente que tenga al menos una *relación temporal de agente*. El máximo número de tablas corresponde al número de clases agente. La *tabla histórica para una relación de agente* incluye instancias, cada una de las cuales contiene la identificación del objeto agente, así como el nombre de la clase y servicio ejecutado, el instante temporal de ejecución, información de identificación de los datos afectados y una indicación de si se pudo o no (y el por qué) ejecutar.

La estructura de las tablas base de las clases agente, no se ve afectada por la inclusión de relaciones de agente temporizadas.

La estructura de la **tabla histórica de agente** es:

En el ejemplo de la figura 3.1, si la relación de agregación "*asignado*" se especifica como temporizada, la tabla histórica debe incluir los intervalos (determinados por los atributos *inicio_depto_empleado* y *fin_depto_empleado*) en los que existió (y existe) la relación de asignación de cada empleado en cada uno de los departamentos de la empresa.

La generación de una instancia en la tabla histórica de agregación, permite dos implementaciones posibles:

- Que la implementación del servicio *new* de la clase en la que está definida la clave ajena y cada servicio que afecte a dicha clave ajena (de la relación de agregación definida como temporal) sea el encargado de ejecutar el *insert* correspondiente en la tabla "histórica de agregación", y que la implementación del servicio *destroy* de la clase y cada servicio que afecte a la clave ajena en cuestión, será la encargada de ejecutar la *actualización* (modificación correspondiente al cierre del intervalo para la relación de agregación) correspondiente en la misma tabla "histórica de agregación".

  Que se defina un *disparo* ("trigger") asociado a la creación del objeto y otro a la modificación de la clave ajena para abrir el intervalo temporal de la relación de agregación. Además, se debe definir un disparo (con lo cual se cierra dicho intervalo) para la modificación de la clave ajena en cuestión y el borrado del objeto en la tabla base. Esta opción será la que se elegirá, si el SGBDR permite la definición de disparos de la base de datos; de lo contrario, se debe usar la primera.

Cuando una relación de agregación se especifica como temporal, para garantizar la integridad referencial, se tratan automáticamente como temporales (aunque no hayan sido definidas como tales) las clases relacionadas.

Para el ejemplo, si se especifica la relación de agregación como temporal, entonces se tratan (aunque no hayan sido especificadas como tales) automáticamente las clases Empleado y Departamento como temporales, generando las tablas históricas correspondientes.

La clave primaria de la tabla histórica para una relación de agregación corresponde a los identificadores de las clases relacionadas y se le añade el tiempo de inicio del intervalo en el que es válida la relación de agregación. En el ejemplo anterior, si se especifica la relación de agregación como temporal, se crea la *tabla histórica*:

H_DEPTO_EMPLEADO:

(código, inicio_empleado, cod_depto, inicio_depto, inicio_depto_empleado, fin_depto_empleado)

Las claves ajenas *código+inicio_empleado* y *cod_depto+inicio_depto* referencian a la tablas históricas de *empleado* y *departamento* respectivamente.

4.4 Relaciones de Herencia

Para las relaciones de herencia temporizadas, la clase especializada se trata como temporal a nivel de modelado conceptual, y la tabla histórica correspondiente recogerá la información sobre los intervalos de tiempo en los que la relación de herencia existe, de la misma forma que presentamos anteriormente al estudiar la representación de clases temporizadas. Es decir, una relación de herencia que se especifique como temporal, trata automáticamente a la clase especializada como temporal (aunque no haya sido definida como tal) y se le da el mismo tratamiento de clase temporizada.

Cuando una relación de herencia se especifica como temporal, además de tratar la clase especializada como temporal, también se tratan como temporales (aunque no hayan sido definidas como tales) todas las clases que son ancestros en la línea jerárquica ascendente hasta la raíz del árbol de herencia. Esto se hace con el fin de conservar la información temporal del objeto como un todo y para mantener el esquema de la base de datos relacional normalizada.

Para el ejemplo anterior, si *Empleado* es una especialización de la superclase *Persona* y si se define como temporal la relación de herencia entre las clases *Empleado* y su especialización *Administrativo*, entonces se tratan automáticamente (aunque no hayan sido definidas como tales) las clases *Persona*, *Empleado* y *Administrativo*, y se generan las tablas históricas correspondientes.

4.5 Relaciones de Agente

La especificación de temporalidad (espacio del problema) en cada relación de *agente* que actúa como activador de un *servicio* origina en la base de datos (espacio de la solución) la creación de una *tabla histórica de agente* (o "*log*"), que contiene la información histórica por medio de la cual se puede conocer y controlar las actividades de ejecución de servicios. Como la ejecución de un servicio se considera un proceso atómico, la marca temporal usada es el instante temporal.

Para almacenar los datos históricos generados por las relaciones de agente temporales, se genera una tabla histórica ("log") por cada clase agente que tenga al menos una *relación temporal de agente*. El máximo número de tablas corresponde al número de clases agente. La *tabla histórica para una relación de agente* incluye instancias, cada una de las cuales contiene la identificación del objeto agente, así como el nombre de la clase y servicio ejecutado, el instante temporal de ejecución, información de identificación de los datos afectados y una indicación de si se pudo o no (y el por qué) ejecutar.

La estructura de las tablas base de las clases agente, no se ve afectada por la inclusión de relaciones de agente temporizadas.

La estructura de la **tabla histórica de agente** es:

(oid, inicio_oid, instante, nombre_clase, nombre_servicio, argumentos, resultado, mensaje)

donde *Oid* es el identificador del objeto agente, *inicio_oid* es el instante en que el oid toma el valor asignado, *instante* referencia al momento en el que se ejecuta el servicio en cuestión, *nombre_clase* corresponde al nombre de la clase que contiene el servicio ejecutado, *nombre_servicio* es el nombre del servicio ejecutado, *argumentos* corresponde a un "string" con información que permite identificar los datos que son parámetros afectados durante la ejecución del servicio, *resultado* indica si se pudo o no ejecutar el servicio y *mensaje* corresponde a un mensaje de error en caso de fallo en la ejecución del servicio.

Agente	*Inicio_agente*	*Instante*	*Clase*	*Servicio*	Argumentos	Resultado	Mensaje
e1	3	20	Empleado	cambiar_salario	Emp=1234	ok	
e2	8	26	Departamento	Crear	Cod_dpto=d5	fail	error 132
...		...	...	...	...	...	...

Tabla 4.5 Tabla histórica para log de agente

El instante en el que un agente activa la ejecución de un servicio (cuya relación es especificada como temporal), se implementa haciendo que el servicio ejecutado por el agente sea el encargado de ejecutar el *insert* correspondiente a la(s) tabla(s) histórica(s) de agente afectadas.

Especificar como temporal una relación de agente que activa un servicio, implica que la clase agente se trate como temporal (aunque no haya sido especificado como tal), lo cual permite mantener la integridad de la clave ajena (oid del agente incluida en la tabla de log) que referencia a la tabla histórica de la clase. Para el ejemplo de la figura 3.1, si se especifica como temporal la relación de agente entre la clase *Empleado* y el servicio *crear* de la clase *Departamento*, se crean la tabla histórica de empleado (H_EMPLEADO) y la tabla de log (H_LOG_EMPLEADO).

La clave primaria de la tabla histórica para una relación de agente corresponde al identificador de la clase agente (*oid+inicio_oid*) y se le añade el instante de tiempo en el que se ejecuta el servicio. Se crean además referencias de clave ajena desde cada tabla de log, hacia su respectiva tabla histórica de la clase. La llave *oid+inicio_oid* en la tabla de log es foránea y referencia a la tabla histórica de la clase que es agente temporal.

Para una relación de agente temporal con un servicio, además de tratarse la clase agente como temporal, se deben tratar automáticamente como temporales las relaciones de agente entre las subclases (de todos los niveles inferiores en una relación jerárquica de herencia) y el servicio en cuestión, y por lo tanto, también se tratan como temporales (aunque no hayan sido especificadas como tales) las mismas subclases para conservar la integridad referencial de la base de datos histórica. Es decir, si existe una relación de agente temporizada que activa un servicio, cualquier objeto agente que se identifique con la llave primaria de una de sus subclases, puede activar el mismo servicio y dicho evento debe ser registrado históricamente en un log.

Para el mismo ejemplo, si se especifica como temporal la relación de agente entre la clase *empleado* y el servicio *crear* de la clase *departamento*, además de crearse la tabla histórica H_EMPLEADO y la tabla de log H_LOG_EMPLEADO, se crean las tablas históricas de las subclases de *empleado* (*administrativo, operativo,...*), y sus tablas de log: (H_LOG_ADMINISTRATIVO y H_LOG_OPERATIVO).

5 Conclusiones y Trabajo Futuro

No es habitual abordar conjuntamente el problema de especificar adecuadamente propiedades temporales (introduciendo las primitivas necesarias) y el problema de representarlas adecuadamente en el producto software resultante de un proceso de producción de software de calidad. La contribución de este trabajo es justamente el haber abordado esos dos problemas en un marco unificado, y dentro de un método de producción automática de software (OO-Method) basado en un modelo orientado a objetos con una sólida base formal.

La versión del modelo de objetos presentada, aumenta la expresividad de OO-Method añadiendo la especificación de este tipo de propiedades temporales al definir las clases del sistema. En concreto, cada clase (simple o compleja), relaciones (de agregación, herencia y agentes) entre clases, y los atributos variables podrán ser declarados como temporizados en la línea de lo expuesto anteriormente.

La especificación de temporalidad de *agente* que actúa como activador de un *servicio*, enriquece la semántica del sistema ya que permite tener una *tabla histórica de agente* (o *log)* que permite conocer y controlar las actividades de ejecución de servicios.

En este trabajo en definitiva se presenta una versión de OO-Method que incluye la especificación de propiedades temporales en la fase de modelado conceptual, con su correspondiente representación software de acuerdo con el paradigma de programación automática asociado a OO-Method. Con esto, se complementan trabajos realizados en el ámbito de la incorporación de propiedades temporales a modelos conceptuales, con trabajos que, basados en el modelo orientado a objetos como paradigma de modelado, proporcionan entornos de generación automática de código en tales ambientes software con todas su implicaciones.

Añadir la expresividad temporal a la herramienta CASE, ha permitido lograr como resultado una base de datos relacional histórica con toda la información que el analista del sistema determine indispensable para conservar a partir del modelo de objetos de la aplicación.

A partir de la inclusión de la expresividad temporal, OO-Method se presenta como una metodología para la producción automática de software que permite especificar un rango mucho más amplio de sistemas de información, incluyendo aquellos en donde los requerimientos implican un manejo de información temporal, con datos históricos y control de ejecución. De hecho, el método extendido con la expresividad temporal está siendo utilizado exitosamente en la resolución con OO-Method de Sistemas de Información reales, en el ámbito de diversos proyectos de I+D realizados con empresas locales dedicadas a la producción de software.

Este esfuerzo realizado tiene como proyección futura el poder usar lógica temporal en precondiciones y restricciones, planteando la especificación de propiedades temporales en el ámbito de los modelos dinámico y funcional. En el caso del modelo dinámico, ello permitiría especificar precondiciones temporizadas (es decir, que son válidas sólo en determinados periodos de la vida del objeto). Aplicado al modelo funcional, se podrían especificar análogamente evaluaciones temporizadas.

La expresividad temporal en OO-Method presentada en este trabajo se puede proyectar hacia el desarrollo de un lenguaje de consulta temporal orientado al manejo de la información histórica registrada en la base de datos relacional temporal. Se espera a partir de este lenguaje de consulta temporal, no solo tener acceso a la información para consultas históricas, sino que también se debe buscar un mecanismo que permita establecer consultas donde intervengan datos actuales con información histórica.

El uso de un lenguaje de consulta temporal permitirá conocer todos los estados pasados y presentes de una aplicación, a partir de los cuales se pueden proyectar estados futuros que facilitarán la gestión de los administradores para la toma eficiente de decisiones disminuyendo el nivel de incertidumbre presentado por la incapacidad para conocer información histórica.

6 Referencias

Antunes, D.C.; Heuser C.A.; Edelweiss N. *Modelagem temporal de transacoes em banco de datos*. Actas de las I Jornadas Iberoamericanas de Ingeniería de Requisitos y Ambientes de Software, IDEAS 98, Torres (Brasil), Abril 1998.

Bergamaschi, S.; Sartori, C. *Chrono: A conceptual Design Framework for Temporal Entities*. Italy, 1998.

Clifford, J.; Crocker, A. *The Historical relational data model (HRDM) and algebra based on lifespans*. Proceedings of the 3rd International Conference on Data Engineering, Feb. 1987, Los Angeles, California. p.528-537..

Edelweiss, N.; Oliveira, J. Palazzo M.; Pernici, B. *An Object-Oriented Temporal Model*. Proceedings of the 5th International Conference on Advanced Information System Engineering - CAISE'93, June 8-11, 1993, Paris. Heidelberg: Springer-Verlag, 1993. p.397-415. (Lecture Notes in Computer Science 685).

Edelweiss, N.; Oliveira, J. Palazzo M. *Modelagem de Aspectos Temporais de Sistemas de Informação*. Recife: Universidade Federal de Pernambuco, 1994. Livro texto da 9a Escola de Computação, 1994, Recife..

Edelweiss, N. *Bancos de Dados Temporais: Teoria e Prática*. Report UFMG. P 225-282, Joao Paulo Kitajima 1998

Elmasri, R.; Kouramajian, V. *A temporal query language based on conceptual entities and roles*. Proceedings of the 11th International Conference on the Entity Relationship Approach, 1992, Karlsruhe, Germany. Berlin: Springer Verlag, 1992. p.375-388. (Lecture Notes in Computer Science, v.645).

Elmasri, R.; Wuu, G. T. J.; Kouramajian, V. *A temporal model and query language for EER Databases*. In: TANSEL, A. et al. (Eds.). Temporal databases: theory, design and implementation. Redwood City: The Benjamin/ Cummings Publishing, 1993. p. 212-229.

Gadia, S. *A homogeneous relational model and query language for temporal databases*. ACM Transactions on Database Systems, New York, v.13, n.4, p.418-448, Dec. 1988.

Heuser, C.; Antunes, D. *Conceptual Modeling of Transactions Combined with Temporal ER Diagrams*. Report Interno UFRGS – Instituto de Informática, Porto Alegre RS (Brasil), 1997.

Jensen , C. S. (Ed). *A consensus glossary of temporal database concepts*. ACM SIGMOD Record, New York, v.23, n.1, p. 52-64, Mar 94.

Kafer, W.; Schoning, H. *Realizing a temporal complex-object data model*. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 2-5, 1992, San Diego. p.266-275.

Lee, J. Y.; Elmasri, R. A. *An EER-Based Conceptual Model and Query Language for Time-Series Data.* U.S.A. 1998

Lorentzos, N.A.; Johnson, R.G. *Extending relational algebra to manipulate temporal data. Information Systems*, v.13, n.3, p.289-296, 1988.

Loucopoulos, P.; Theodoulidis, C.; Wangler, B. *The entity relationship time model and conceptual rule language.* Proceedings of the 10th International Conference on the Entity Relationship Approach, 1991, San Mateo, California.

Navathe, B.; Ahmed, R. *A Temporal relational model and a query language.* Information Sciences, v.49, p. 147-175, 1889.

Özsoyoglu, G.; Snodgrass, R. T. *Temporal and real-time databases: a survey.* IEEE Transactions on Knowledge and Data Engineering, New York, v.7, n.4, p.513-532, Aug.1995.

Pastor, O. *OO-Method: An Object Oriented Methodology for Software Production.* Actas de DEXA 92, Springer-Verlag, pp 121-127. ISBN: 3-211-82400-6. 1992

Pastor, O.; Ramos, I. *OASIS 2.1.1.: A class-Definition Language to Model Information Systems Using an Object Oriented Approach.* February 94 (1 ed.), Mars 95 (2 ed.), Oct 95 (3 ed.). 1995.

Pastor, O.; Pelechano, V; Bonet, B.; Ramos, I. *An OO Methodological Approach for Making Automated Prototyping Feasible.* Proc. DEXA 96, Springer-Verlag. September 1996.

Pastor, O.; Insfrán, E.; Pelechano, V.; Romero, J.; Merseguer, J. *OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods.* CAiSE97. June 1997

Pelechano, V.; Insfrán, E.; Pastor, O. *El Metamodelo OO-Method y su Repositorio Relacional.* UPV, 1998.

Rose, E.; Segev, A. *TOODM: A Temporal object-oriented data model with temporal contraints.* Proceedings of the 10th International Conference on the Entity Relationship Approach, Oct. 1991, San Mateo, California.

Sarda,N.L. *Extensions to SQL for historical databases.* IEEE Transaction on Knowledge and Data Engineering, v.2, n.2, p. 220-230, June 1990.

Snodgrass, R. *The Temporal query language TQuel.* ACM Transactions on Database Systems, New York, v.12, n.2, p.247-298, June 1987.

Snodgrass, R. T. (Ed.) *The TSQL2 Temporal Query Language.* Norwell: Kluwer Academic Publishers, 1995. 674p.

Su, S.Y.W.; Chen, H.-H. M. *A Temporal knowledge representation model OSAM*/T and its query language OQL/T.* Proceedings of the 17th International Conference on Very Large Data Bases, Sept. 1991, Barcelona. p.431-442.

Tansel, A.U. *Adding time dimension to relational model and extending relational algebra.* Information Systems, v.11, n.4, p.343-355, 1986.

Tansel, A. *et al.* (Eds.). *Temporal Databases: Theory, Design and Implementation.* Redwood City: The Benjamin/ Cummings Publishing, 1993.

Tauzovich, B. *Towards temporal extensions to the entity-relationship model.* Proceedings fo the 10th International Conference on the Entity Relationship Approach, 1991, San Mateo, California.

Wuu, G. T. J.; Dayal, U. *A uniform model for temporal and versioned object-oriented databases.* In: TANSEL, A. *et al.* (Eds.). Temporal databases: theory, design and implementation. Redwood City: The Benjamin/Cummings Publishing, 1993. p.230-247.

Carlos Augusto Meneses *Recibió el grado de Licenciado en Informática en la Universidad Tecnológica de Pereira - UTP, Colombia. Es alumno de doctorado del Programa Ingeniería de Sistemas y Computación (PISC) en la UTP y actualmente realiza una estancia de investigación en la Universidad Politécnica de Valencia - UPV, España. Es miembro invitado del Grupo de Investigación en Programación Lógica e Ingeniería del Software en el Departamento de Sistemas Informáticos y Computación de la UPV. Siendo sus líneas de investigación: modelado conceptual orientado a objetos, generación automática de código, métodos de especificación formal de sistemas.*

Oscar Pastor López *Es profesor Titular del Dpto. de Sistemas Informáticos y Computación (DSIC) de la Universidad Politécnica de Valencia, España. Recibió el grado de Licenciado en Ciencias Físicas por la Universidad de Valencia y de Doctor en Informática por la Universidad Politécnica de Valencia. Es miembro del Grupo de Investigación en Programación Lógica e Ingeniería del Software en el DSIC y responsable de varios proyectos de I+D en modelado conceptual orientado a objetos y generación automática de código. Sus actuales líneas de investigación comprenden: ingeniería de requisitos, modelado conceptual, generación automática de código, diseño de bases de datos y metodologías orientadas a objeto.*

Juan Carlos Molina *Recibió el grado de Licenciado en Informática de Gestión en la Universidad Politécnica de Valencia, España. Es alumno de doctorado del Programa «Programación Declarativa e Ingeniería de la Programación» del Departamento de Sistemas Informáticos y Computación (DSIC). Trabajó como Programador Aplicaciones Informáticas de Gestión en el Ayuntamiento de Valencia y como becario de investigación del grupo OO-Method (DSIC-UPV) para el Diseño e implementación de un generador automático de código para Visual Basic en una arquitectura transaccional multi-capa a partir de modelos conceptuales orientados a objeto. Actualmente es el Coordinador del Área de Traducción Automática de Código en la empresa CARE Technologies S.A. de Denia, Alicante.*

Emilio Insfrán Pelozo *Es profesor Asociado del Departamento de Sistemas Informáticos y Computación (DSIC) de la Universidad Politécnica de Valencia, España. Anteriormente Profesor Encargado de Cátedra del Departamento de Ingeniería Electrónica e Informática de la Universidad Católica "Ntra. Señora de la Asunción", Paraguay. Recibió el grado de Licenciado en Análisis de Sistemas Informáticos por la Universidad Nacional de Asunción, Paraguay y de Master en Ciencias de la Computación por la Universidad de Cantabria, España. Es miembro del grupo de investigación de Programación Lógica e Ingeniería del Software de la Universidad Politécnica de Valencia, y participa en varios proyectos nacionales e internacionales de Investigación y Desarrollo. Sus actuales líneas de investigación incluyen: modelización conceptual, métodos OO, ingeniería de requisitos, lenguajes de especificación, generación automática de código y bases de datos.*