ENTORNOS AUTOMATICOS DE PRODUCCION DE SOFTWARE A PARTIR DE MODELOS CONCEPTUALES ORIENTADOS A OBJETOS Y TEMPORALES

Carlos Meneses[†], Oscar Pastor[††], Emilio Insfrán[††], Carlos Heuser[†††]

[†] Programa de Ingeniería de Sistemas y Computación (PISC)
Universidad Tecnológica de Pereira.
La Julita. Pereira - Colombia
cmeneses@dsic.upv.es

[††] Departament de Sistemes Informàtics i Computació (DSIC)
Universitat Politècnica de València.
Camí de Vera s/n. 46022 Valencia - España
{opastor | einsfran}@dsic.upv.es

[†††] UFRGS – Instituto de Informática
Caixa Postal 15064 – 91501-970 Porto Alegre RS - Brasil
heuser@inf.ufrgs.br

RESUMEN

Para poder especificar la información que represente de manera adecuada y completa los aspectos estáticos y dinámicos de un sistema, es importante incluir una expresividad temporal sobre los elementos utilizados para la construcción del modelo conceptual del sistema estudiado.

En el contexto del proyecto IDEAS (Ingeniería de Ambientes de Software) financiado por el 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) en colaboración con PISC de la UTP (Pereira, Colombia) y del Instituto de Informática de la UFRGS (Porto Alegre, Brasil), 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 sistemas, a partir de su aproximación metodológica OO-Method. El objetivo de este trabajo es incluir los aspectos necesarios que permitan capturar la expresividad temporal de modelos conceptuales orientados a objetos y la estrategia de implementación en el marco de la programación automática.

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

1. INTRODUCCION

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 de requisitos, 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 (entornos de programación, depuradores, etc.). 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.

El *Modelado Orientados a Objetos* constituye una forma de pensar en la que los problemas se resuelven a partir de modelos basados en conceptos del mundo real. 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.

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. La especificación de aspectos temporales en sistemas de información facilita el manejo de datos temporizados y de tareas que pueden tener precondiciones temporales asociadas a su ejecución o presentar interacciones temporales con otras para el control de su ejecución; además de expresar condiciones de integridad temporizadas.

Diversos modelos de datos temporales se han presentado [Ede94, Ozs95], constituyendo la mayoría extensiones temporales de modelos ya existentes. Entre estas, podemos destacar las basadas en modelos relacionales [Gad88, Lor88, Nav89, Sar90], E-R [Elm92, Elm93, Lee98] y las basadas en modelos orientados a objetos [Ede93, Kaf92, Wuu93]. Recientemente, esfuerzos importantes en el área del modelado temporal han sido realizados en la definición del lenguaje de consulta temporal TSQL2 [Sno95], que pretende ser propuesta para manipulación de bases de datos relacionales temporales.

En la aproximación presentada en este trabajo se toman características temporales de TempEr-Tr [Heu97, Ant98] de la UFRGS, el cual ha sido definido como un modelo temporal suficientemente expresivo para usarse en la especificación de sistemas de información utilizando una primitiva (elemento) temporal, diccionarios de datos separados para simplificar diagramas y conceptos de Redes de Petri que modelan transacciones posibles.

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. En la sección 5 se presentan algunas conclusiones y trabajos futuros.

2. OO-METHOD

OO-Method [Pas97] 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 convencionales (UML, OMT,...). 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*** [Pas95] 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 guía 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 aspectos de implementación. Con esta descripción, se genera automáticamente una especificación formal del sistema en OASIS.
2. Aplicación de un *Modelo de Ejecución*. La especificación formal es la fuente del *modelo de ejecución* que determina de forma automática, de acuerdo a su semántica operacional, las características del sistema dependientes de la implementación: interfaz de usuario, control de acceso, activación de servicios, etc.

2.1. Modelo Conceptual

Se empieza la fase de modelado con la construcción de tres modelos: Objetos, Dinámico y Funcional, que recogen las propiedades relevantes del sistema en desarrollo desde tres complementarios puntos de vista: estructura, comportamiento y cambios de estado, dentro de un riguroso marco orientado a objetos.

2.1.1. Modelo de Objetos

El Modelo de Objetos es descrito gráficamente por un Diagrama de Clases (DC) que muestra la estructura de las clases identificadas en el dominio del problema así como sus relaciones. Una clase se representa gráficamente con una caja dividida en secciones donde se recoge información sobre atributos y servicios. Para el tratamiento de la complejidad se pueden definir relaciones estructurales en términos de agregación (parte-de) y herencia (es-un).

En la Figura 2.1 se presentan dos clases DEPTO y EMPLEADO, y una relación de agregación entre ellas, incluyendo información sobre cardinalidades. Además, se pueden indicar otras propiedades: inclusiva/referencial o estática/dinámica (información más detallada puede verse en [Pas95]).

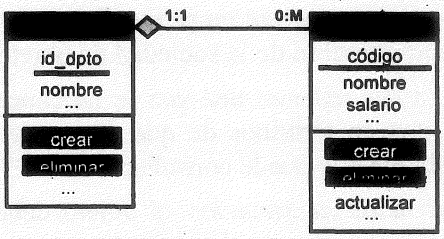

Figura 2.1: Relación de agregación

La relación de herencia puede ser expresada mediante los operadores de especialización y generalización. Gráficamente se representan como flechas que conectan la subclase con la superclase y son etiquetadas con sus propiedades relevantes. En el caso de especialización puede ser etiquetada por una condición de especialización o con los eventos correspondientes de activación/cancelación de la relación de *rol* (especialización temporal).

2.1.2. Modelo Dinámico

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

Los *Diagramas de Transición de Estados (DTE)* están basados en los *Statecharts* de Harel y describen el comportamiento de objetos estableciendo *vidas válidas*. Una *vida válida*, es una secuencia correcta de *estados* que caracterizan un comportamiento correcto para todos los objetos de una clase. Los estados denotan *situaciones* en las que pueden encontrarse los objetos como consecuencia de la ocurrencia de eventos. Las transiciones representan los cambios de estado válidos. Estas pueden ser restringidas introduciendo condiciones.

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

2.1.3. Modelo Funcional

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 una relación atributo-evento [Pas97] entre un conjunto predefinido de *categorías*, las cuales indican la información necesaria para determinar el cambio de valor del *atributo* ante la ocurrencia del *evento*.

2.2. 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 define el sgte. modo de trabajo:

- *Control de acceso*: en primer lugar, el objeto[1] que desee conectarse al sistema deberá identificarse como miembro de la sociedad de objetos.
- *Vista del sistema*: una vez se ha conectado un usuario, tendrá una visión clara de la sociedad de objetos en términos de qué clases de objetos puede ver, qué servicios que puede activar y qué atributos que puede consultar.
- *Activación de servicios*: 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, verificación de transición válida de estado, 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 (*trigger*). La implementación de estos pasos, claramente definidos en [Pas97], guía el proceso de implementación asegurando la *equivalencia*

[1] En nuestra visión de sistema los usuarios son modelados como objetos que solicitan la ejecución de servicios.

funcional entre la descripción del sistema recogida en el modelo conceptual y su reificación en un entorno de programación.

3. ESPACIO DEL PROBLEMA. TEMPORALIDAD EN LOS MODELOS CONCEPTUALES

Se tienen tres formas de marcas (*primitivas temporales*), para especificación temporal [Jen94]:

- *Instante temporal.* Por ejemplo: el instante en que una empresa contrata a un empleado. En este caso sólo se registra un dato temporal.
- *Intervalo temporal.* Definido por dos instantes temporales, indicando el inicio y el fin del intervalo. Por ejemplo: el tiempo que un empleado trabaja para una empresa.
- *Elemento temporal.* Constituido por la unión finita de intervalos temporales. Por ejemplo: los intervalos de tiempo en los que un empleado ha estado contratado por una empresa.

En el modelo presentado en este artículo utilizamos el *intervalo temporal* (que proporciona información explícita y con un conjunto finito de instancias constituye el *elemento temporal*), excepto en la expresividad de las relaciones de agente, que por modelar cambios de estado que ocurren en un instante de tiempo se utiliza el *instante temporal* (que proporciona información implícita). Para agilizar y facilitar las consultas, se utiliza en las tablas históricas, una especificación por intervalo temporal.

En cuanto al *significado de dicha marca temporal*, pueden emplearse: *Tiempo de transacción* (en el que la información fue introducida en la base de datos que implementa la aplicación), *tiempo de validez* (en que la información es válida en la realidad modelada y corresponde al tiempo de existencia o vida de un objeto en el sistema) o *ambos*. El *tiempo de validez* es usado en OO-Method, por ser relevante en la fase de modelo conceptual.

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.

La especificación de la información temporal en un modelo orientado a objetos se puede hacer bajo cinco diferentes niveles: clases, atributos, relaciones de agregación entre objetos, relaciones de herencia y agentes que actúan como activadores de servicios.

3.1. Clases

Las *clases temporizadas*, son aquellas en la que nos interesa conservar la información de los distintos intervalos temporales en los que un objeto de dicha clase ha existido. La información asociada a *clases de objetos temporizados* indica una evolución temporal del objeto como un todo – cuándo se crea, cuándo deja de existir, periodos de suspensión de actividad, etc. Cuando un objeto de una *clase temporizada* es eliminado, no desaparece de la base de datos utilizada, sino que se cerrará adecuadamente su intervalo de existencia. Por ejemplo, si una clase *empleado* se declara temporizada, se mantiene un "registro histórico" de su vida en distintos intervalos de tiempo. (Ver Tabla 3.1, columna *Existencia*)

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 (estáticos). Por ejemplo, el atributo *salario* de la clase *empleado* (un empleado puede haber tenido diferentes salarios en diferentes periodos de existencia, ver Tabla 3.1)

Existencia	*Código*	*Nombre*	*Salario*
[3,10] ∪ [20,null]	e1	José Fernández	1800 [3,6] ∪ 2200 [7,10] ∪ 2700 [20,null]
[8,null]	e2	Manuel Gómez	1700 [8,20] ∪ 1800 [21,null]
[7,35]	e3	María López	1600 [7,20] ∪ 1900 [21,35]
...	...	...	...

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.

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 permite la *coexistencia* de especializaciones temporizadas y no temporizadas. Por ejemplo: Los objetos de la clase *empleado* de acuerdo al cargo que desempeñan se clasifican como *operativos* o *administrativos*; podemos estar interesados en mantener la información histórica de quiénes y cuándo han ocupado cargos administrativos, pero sin que se requiera la información histórica de los que han ocupado cargos operativos.

3.5. Relaciones de Agentes

Las *relaciones temporales de agente*, mantienen una historia del instante de tiempo en el que un agente activa un servicio. Esta información histórica se corresponde con un *log* sobre las operaciones realizadas por los usuarios. Por ejemplo: Si se especifica como temporal la relación de agente entre la clase *empleado* y un servicio *cambio_salario* también de la clase *empleado*, cada vez que se ejecuta este servicio, queda un registro histórico que indica el identificador del objeto que activó la ejecución del servicio, el servicio, el identificador del objeto modificado y la marca de tiempo.

La especificación de la expresividad temporal en el Modelo de Objetos de OO-Method (ver Figura 3.1) se representa mediante un icono (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.

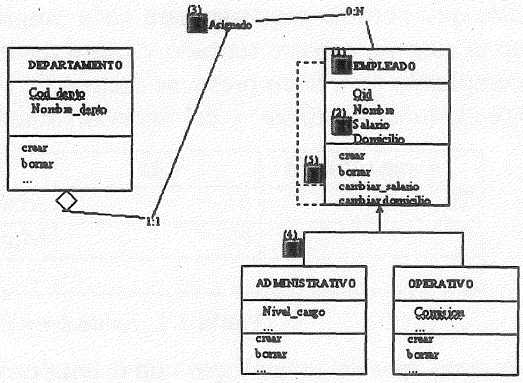

Figura 3.1. *Especificación de temporalidad en OO-Method*: (1 Clases. (2) Atributos. (3) Relaciones de agregación. (4) Herencia (5) Agentes activadores de servicios.

4. ESPACIO DE LA SOLUCION. TRATAMIENTO DE LAS PROPIEDADES TEMPORALES EN EL MODELO DE EJECUCIÓN

La principal característica de OO-Method es su capacidad de generar de forma automática aplicaciones en entornos industriales de producción de software. Estas aplicaciones utilizan un SGBDR como repositorio de datos, y los servicios declarados en las clases se representan como procedimientos almacenados, métodos de clases o estructuras de programación, dependiendo del entorno de desarrollo.

El que OO-Method utilice en sus entornos de producción SGBDRs como contenedores de datos, nos orienta hacia cómo representar apropiadamente esa expresividad temporal en las aplicaciones generadas. A continuación se muestra como se introduce el tratamiento de las características temporales en el modelo de ejecución y como afecta 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, o separar los datos *actuales* de los *históricos* en tablas distintas. El hecho de que utilizar una única tabla obligue 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 tenga que modelar el manejo de conjunto de valores históricos para una instancia de un elemento temporal, hace que se opte por una representación basada en tablas diferentes para la información actual e histórica. Además, así evitaremos problemas asociados a la normalización para el manejo de tablas y al tratamiento de la *información faltante* en las implementaciones relacionales actuales.

4.1. Clases

Cada clase se representa como una tabla (llamada *tabla base*). Si la clase se especifica como *temporizada*, se introduce a la *tabla base* un atributo adicional que incluya información del intervalo actual de existencia de los objetos de la clase y este atributo llamado *tiempo_inicio* representa el instante en el que el objeto se ha creado. La tabla base tendrá la siguiente estructura:

(oid, tiempo_inicio, {lista de atributos})

Donde, *oid* es el identificador del objeto; *tiempo_inicio* referencia el instante en que se creó el objeto; *lista de atributos*, contiene los demás atributos asociados a la tabla base.

Por ejemplo: Si la clase *empleado* se especifica como temporal, la tabla base incluye un atributo *tiempo_inicio*, que permite registrar para cada empleado, el instante de tiempo en el que ingresó (o reingresó, en el caso de haberse retirado y luego readmitido) a la empresa. El total de tiempo en que ha existido un empleado para la empresa, se deduce de la suma de los intervalos cerrados de las instancias temporales (en la *tabla histórica*) y del intervalo abierto (en la *tabla base*).

Tiempo_inicio	*Oid*	*Nombre*	*Salario*
20	e1	José Fernández	2700
8	e2	Manuel Gómez	1800
...	...	...	...

Tabla 4.1. ***Tabla base*** de la clase *empleado*

Cada clase que se especifique como temporizada, tendrá asociada a su *tabla base*, una *tabla histórica*, que incluye las instancias eliminadas de la *tabla base* y con la información de cierre del intervalo temporal de existencia de dichos objetos. La estructura de la *tabla histórica* es:

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

Donde *oid* es el identificador del objeto, *tiempo_inicio* referencia al momento en el que se abrió el intervalo temporal de existencia considerado, instanciado al crear el objeto en la *tabla base*, *tiempo_fin* representa el instante (en el que se generó el objeto histórico) de cierre de dicho intervalo, y *lista de atributos* incluirá el conjunto de atributos asociados al instante en el que se introduzca esta tupla en la *tabla histórica*. Este instante será justamente el de ocurrencia del evento de destrucción declarado en la clase.

Se presentan dos situaciones: Una en la que sólo se genera una instancia histórica asociada al borrado de una tupla en la *tabla base* (la tupla solamente existirá en la *tabla histórica* cuando haya sido eliminada de la *tabla base*), y otra en la que se adiciona una instancia histórica cuando se la inserta una tupla en la *tabla base*, obligando a tener una modificación de la tupla histórica asociada al borrado de la tupla en la *tabla base*. La primera opción es más fácil de implementar, pero se escoge la segunda porque facilita las consultas sobre el tiempo de existencia de un objeto temporal en el sistema, con lo cual solo se accede a la *tabla histórica* (y no a la *tabla base*)

Ello permite dos implementaciones posibles para esta situación: Que la implementación del servicio *insert* 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*; o que se defina un *disparo* ("trigger") asociado a la inserción y otro al borrado de tuplas de esa *tabla base*. Opción que se elegirá, si el SGBDR permite la definición de disparos de la base de datos; de lo contrario, usaremos la primera.

Tiempo_inicio	*Tiempo_fin*	*Oid*	*Nombre*	*Salario*
3	10	e1	José Fernández	2200
7	35	e3	María López	1900
...	...	...	...	...

Tabla 4.2. ***Tabla histórica*** de la clase *empleado*

En el ejemplo anterior, el atributo *tiempo_fin* de la tabla histórica, contiene el instante en que cada empleado se ha retirado de la empresa; y *tiempo_inicio* de la misma tabla corresponde al instante en que se había creado la instancia del objeto en la tabla base antes de ser borrado.

4.2. Atributos

Cada atributo declarado temporizado (espacio del problema) origina en la base de datos (espacio de la solución) la inclusión de un nuevo atributo *tiempo_inicio_atributo* (que corresponde al instante de apertura del intervalo en que el atributo toma un nuevo valor) en la *tabla base* y la creación de una *tabla histórica* asociada a la *tabla base* (de la clase en la que están los atributos).

La estructura de la *tabla base*, sería la siguiente:

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

Donde, *lista de tiempo_inicio_atributos*, corresponde al conjunto de atributos que indican el instante de apertura del intervalo de cada atributo marcado como temporal. Inicialmente, coinciden con el instante de creación del objeto (aún cuando existan atributos con valor nulo), después cambian en la medida en que van cambiando los valores de sus respectivos atributos.

En el ejemplo, la *tabla base* tiene el atributo *salario* como temporal:

Tiempo_inicio	*Oid*	*Nombre*	*Salario*	*Tiempo_inicio*
20	e1	José Fernández	2700	20
21	e2	Manuel Gómez	1800	21
...	...	...	...	...

Tabla 4.3. Tabla histórica de la clase *empleado* con atributo temporal *salario*

La estructura de la *tabla histórica de atributo*, sería la siguiente:

(oid, tiempo_inicio_atributo, tiempo_fin_atributo, valor del atributo)

Donde *oid* es el identificador del objeto, *tiempo_inicio_atributo* el instante de apertura del intervalo temporal en el que se asigna un valor al atributo en la tabla base; *tiempo_fin_atributo es* el instante de cierre (en el que se modificó el atributo temporal) y *valor_del_atributo* el valor correspondiente del atributo durante el intervalo considerado.

En el ejemplo, la *tabla histórica* del atributo *salario*, debe incluir el atributo *tiempo_fin_atributo*, con lo cual queda registrado el instante en que cambia el valor del salario o cuando el empleado se retira de la empresa (al borrar la instancia del objeto en la *tabla base*, es necesario conservar el último valor de cada uno de los atributos temporales).

Tiempo_inicio_salario	*Oid*	*Tiempo_fin_salario*	*Salario*
3	e1	6	1800
7	e1	10	2200
...	...	...	...

Tabla 4.4 Tabla histórica para el atributo temporal *salario.*

De nuevo se presentan dos situaciones: Una en la que solo se genera una instancia histórica asociada a la modificación del atributo temporal y al borrado de tupla (al borrar un objeto, además se debe generar una instancia histórica para cada atributo temporizado) en la tabla base. La otra, en la que se adiciona una instancia histórica cuando se inserta una tupla (creación del objeto) y cuando se

modifica el atributo temporal en la tabla base; esto obliga a tener una modificación de la tupla (para cerrar el intervalo temporal) en la tabla histórica del atributo, asociada al borrado de la tupla y a la modificación del atributo temporal en la tabla base. La primera opción es más fácil de implementar, pero se escoge la segunda porque facilita las consultas sobre el tiempo de existencia de un valor para un atributo temporal con lo cual solo se accede a la tabla histórica del atributo (y no a la tabla base).

Si el SGBDR lo permite, la implementación seleccionada será la de declarar un disparo ("trigger") asociado a la creación del objeto y a la modificación del atributo (de la forma "*create trigger on update of atributo_temporizado*"); y un disparo (con lo cual se cierra el intervalo del valor del atributo en la tabla histórica) para la modificación del atributo y el borrado del objeto, siempre que el gestor de base de datos usado posea esa posibilidad. En caso contrario, el *insert* del objeto y cada servicio que afecte al atributo temporal será el encargado de ejecutar el *insert* correspondiente a la tabla histórica del atributo; y el *destroy* del objeto y cada servicio que afecte al atributo temporal será el encargado de ejecutar la modificación correspondiente al cierre del intervalo para el valor del atributo en la tabla histórica.

4.3. Relaciones de Agregación

La agregación se implementa como una restricción de clave ajena entre *las tablas base* correspondientes a las clases involucradas. La cardinalidad de la relación determinará donde ubicar esta clave ajena (en la compuesta o en la componente). Para las *relaciones de agregación temporizadas*, esta clave ajena en la clase base se tratará como un atributo temporizado más, que recogerá la información sobre los intervalos de tiempo en los que la relación se mantiene.

En el ejemplo, debido a que la cardinalidad de la relación de agregación temporal entre las clases *departamento* y *empleado* es de 1 a N, se adiciona el atributo Cod_depto (como clave ajena) en la tabla base de empleado. Este atributo es tratado como un atributo temporal de acuerdo a la representación descrita anteriormente.

Si una relación de agregación temporal es de cardinalidad N a M, se genera una tabla histórica (además de las posibles dos tablas históricas relacionadas con las tablas base de las clases involucradas), la cual contiene los atributos correspondientes a las claves ajenas de las tablas base. En este caso, estos atributos se tratan como temporales.

4.4. Relaciones de Herencia

Para las relaciones de herencia temporizadas, la clase especializada se define 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. En el ejemplo de la figura 3.1, la clase *administrativo* es definida como temporal.

4.5. Relaciones de Agente

La especificación de temporalidad de *agente* que actúa como activador de un *servicio o transacción*, origina 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 y transacciones. Debido a que la ejecución de eventos y transacciones, son procesos atómicos, la marca temporal usada es el instante temporal.

Se presentan dos alternativas: La primera es generar solo una tabla de *log* para el manejo histórico de agentes que activan servicios y la segunda es generar una tabla histórica por cada *relación temporal de agente*, en donde la estructura solo necesita como atributos *instante_ejecución y oid_agente*. Con esta alternativa se tienen tablas más pequeñas, pero se podría generar un gran número de estas y las consultas serían más costosas. La primera facilita las consultas, por eso se utiliza esta representación. La estructura de esta tabla histórica es:

(instante_ejecucion, oid_agente, clase, servicio)

Donde, *instante_ejecución* corresponde al momento en el cual se empieza a ejecutar el servicio o transacción, *oid_agente* es la clave ajena que identifica al agente, *clase y servicio* identifican como clave ajena al servicio o transacción activada y la clase a la que pertenece.

Tiempo_ejecución	*Oid_agente*	*Clase*	*Servicio*
13	e1	Empleado	cambiar_salario
26	e2	Factura	registrar_factura
...	...	...	...

Tabla 4.5 Tabla histórica para la relación de *agente.*

El instante en el que un agente activa la ejecución de un servicio (cuya relación se especifica temporal), permite dos implementaciones posibles: Que en el servicio ejecutado por el agente sea el encargado de ejecutar el *insert* correspondiente a la tabla histórica correspondiente, o que se defina un disparo ("trigger") asociado al evento de inicio de ejecución del servicio. Se elegirá esta mientras el SGBDR destino lo permita, de lo contrario se elegirá la primera.

5. CONCLUSIONES Y TRABAJO FUTURO

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 (constantes o 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 *evento o transacción*, 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 eventos y transacciones.

En este trabajo 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.

Este esfuerzo, realizado en el ámbito del proyecto IDEAS, tiene como proyección futura poder utilizar información temporal para modelar el comportamiento de los objetos tanto en el modelo dinámico como funcional. En el caso del modelo dinámico, permitiría especificar precondiciones temporizadas (es decir, que son válidas sólo en determinados periodos de la vida del objeto) y aplicado

al modelo funcional, se podrían especificar análogamente evaluaciones que tengan un comportamiento determinado dependiendo de condiciones basadas en información temporal.

REFERENCIAS

[Ant98] Antunes, D.C.; Heuser C.A.; Edelweiss N. *Modelagem temporal de transaçoes em banco de datos.* I Jornadas Iberoamericanas de Ingeniería de Requisitos y Ambientes de Software, IDEAS 98, Torres (Brasil), Abril 1998.

[Ede93] 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).

[Ede94] 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.

[Elm92] 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).

[Elm93] 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.

[Gad88] 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.

[Heu97] Heuser, C.; Antunes, D. *Conceptual Modeling of Transactions combined with Temoral ER Diagrams.* Report Interno UFRGS – Instituto de Informática, Porto Alegre RS, Brasil 1997.

[Jen94] 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.

[Kaf92] 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.

[Lee98] Lee, J. Y.; Elmasri, R.A. *An EER-Based Conceptual Model and Query Language for Time-Series Data.* CLEI/98, p.21-34, 1998.

[Lor88] Lorentzos, N.A.; Johnson, R.G. *Extending relational algebra to manipulate temporal data. Information Systems.* v.13, n.3, p.289-296, 1988.

[Nav89] Navathe, B.; Ahmed, R. *A Temporal relational model and a query language.* Information Sciences, v.49, p. 147-175, 1889.

[Ozs95] 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.

[Pas95] Pastor, O.; Ramos, I. *OASIS 2.1.1: A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach.* February 94 (1ª ed.), Mars 95 (2ª ed.), October 95 (3ª ed.)..

[Pas97] 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.

[Sar90] 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.

[Sno95] Snodgrass, R.T. *The TSQL2 Temporal Query Language.* Norwell: Kluwer Academic Publishers, 1995.

[Wuu93] 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.