Tipo de Dato Abstracto para Sistemas de Bases de Datos de Tiempo Real

Carlos E. Buckle, José M. Urriza, Damián P. Barry, Francisco E. Páez

Facultad de Ingeniería, Departamento de Informática
Universidad Nacional de La Patagonia San Juan Bosco - Puerto Madryn, Argentina
cbuckle@unpata.edu.ar, josemurriza@unp.edu.ar, demian.barry@gmail.com,
franpaez@gmail.com

Resumen. En la actualidad, los Sistemas de Tiempo Real incorporan aplicaciones con uso intensivo de datos dentro del extenso espectro de soluciones en los cuales son aplicados. Un problema común a abordar por los desarrolladores, es el diseño orientado a datos con constricciones temporales, debido a que requiere considerar propiedades y reglas específicas para garantizar la validez de objetos respecto a instantes particulares de tiempo. Este trabajo propone un modelo para facilitar dicha tarea. Se presenta el concepto de *dato de tiempo real* con garantía de *consistencia temporal* y un conjunto de definiciones y clasificaciones asociadas. En base a esto, se modela un tipo de dato abstracto parametrizable que encapsula atributos y validaciones de constricciones temporales, de manera que el desarrollador de aplicaciones pueda abstraer y desligarse de esas responsabilidades. El resultado se verifica con la aplicación del modelo sobre un caso concreto de diseño, en un problema de informática industrial.

Palabras Claves: bases de datos de tiempo real, consistencia temporal, datos temporales, ingeniería de software de tiempo real, transacciones de tiempo real.

1 Introducción

Gran parte de los trabajos relacionados con el modelado de Sistemas de Tiempo Real (*STR*) han considerado problemas clásicos generalmente aplicables en sistemas embebidos de automatización industrial, aviónica, exploración espacial, centrales telefónicas y otros. La evolución en los desarrollos, ha permitido expandirse hacia otras áreas de aplicación que involucran sistemas de propósito general e interacción con el usuario, como sitios web de subastas electrónicas, sistemas de supervisión, tableros de comando para toma de decisiones empresariales, etc. En estos sistemas las tareas de tiempo real manejan importantes volúmenes de datos con requerimientos de persistencia. El estudio de estos escenarios ha sido principalmente desarrollado por la subdisciplina de *Sistemas de Bases de Datos de Tiempo Real* (*SBDTR*) ([1, 2]).

Los *SBDTR* deben manejar objetos de datos convencionales y objetos de datos de tiempo real. Estos últimos, son los encargados de reflejar el estado de elementos

variables del ambiente. Para ello, sus valores se actualizan con lecturas periódicas sobre los sensores que se comunican con el entorno. Estos objetos poseen una característica: su valor *envejece* hasta perder vigencia llegado su vencimiento (*Datadeadline* [3]). Las tareas que manejan estos *objetos de dato de tiempo real* son llamadas *transacciones de tiempo real* ([4]). Las cuales, además de garantizar la consistencia lógica de los datos involucrados, deben garantizar la *consistencia temporal* (vigencia) y cumplir con el tiempo de vencimiento establecido para su respuesta (*deadline*).

En la construcción de *SBDTR*, los diseñadores no solo deben considerar el modelado de procesos, sino que necesitan poner fuerte énfasis en el modelado de datos con constricciones de tiempo.

La ingeniería de software ha generado importantes avances en las disciplinas de *STR* y Sistemas de Base de Datos (*SBD*) convencionales, pero ambas han trabajado desligadas entre sí. Por un lado, en *SBD*, el modelado de información variable en el tiempo fue desarrollado por las bases de datos temporales, algunos trabajos con enfoque relacional ([5]) y otros orientados a objetos ([6]). Estos *SBD* manejan el dominio *tiempo*, su estructura de representación y diferentes dimensiones de tiempo a considerar en una transacción. Lamentablemente no contemplan el contexto específico de Tiempo Real, donde los datos pueden perder vigencia durante la transacción y ésta a su vez, debe planificarse para cumplir con su vencimiento.

Por otro lado, en la disciplina de los *STR*, existen trabajos de ingeniería como los patrones de diseño de Douglas en [7], donde se presentan soluciones a problemas comunes de *STR* y se orienta en la metodología de desarrollo. También hay recomendaciones como *MARTE* ([8]) (*OMG: Modeling and Analysis of Real-Time Embedded systems*) que definen un perfil para las etapas de especificación, diseño y validación de *STR*. Estas propuestas son amplias y abarcan todo el espectro de *STR*, guiando el proceso de desarrollo a alto nivel. Sin embargo, no se han enfocado a requerimientos de *SBDTR*.

En la subdisciplina de *SBDTR*, los trabajos apuntan específicamente al modelado de datos con constricciones de tiempo real. En el modelo de *Real Time Semantic Objects Relationships And Constraints* (*RTSORAC*) ([9]) se definen y se describen los componentes que debe tener un modelo de *SBDTR*. Posteriormente, algunos de estos mismos autores presentan un paquete *UML* para especificar objetos de tiempo real ([10]), que ha sido tomado como base para otros perfiles de diseño como el propuesto en [11]. Aunque estos trabajos introducen un punto de partida, es necesario un mayor nivel de refinamiento para que puedan ser aplicados en diseños concretos. Además, deben ampliarse los alcances para abarcar otros aspectos, como por ejemplo, consistencia temporal relativa y objetos de tiempo real de cambio discreto.

Este trabajo presenta una solución genérica al problema puntual de reflejar la semántica de los elementos de tiempo real dentro de los confines de un modelo de datos. Se define un tipo de dato abstracto parametrizable, que encapsula las propiedades de tiempo y las validaciones de consistencia temporal, de manera que el programador de aplicaciones se pueda desligar de estas responsabilidades. El ser parametrizable permite implementar cualquier clase de objeto, extendiendo su funcionalidad con características y constricciones propias de un objeto de tiempo real.

El modelo soporta las diferentes clasificaciones y reglas de validación identificadas en este trabajo. Para verificar el resultado, se ha aplicado sobre un caso concreto de diseño de *SBDTR* en el área de informática industrial.

El resto de este documento está organizado de la siguiente manera: en la sección 2 se presentan conceptos básicos sobre consistencia temporal en *SBDTR*. En la sección 3 se presenta el trabajo realizado y su correspondiente verificación. En la sección 4 se elaboran las conclusiones y se plantean los trabajos a futuro.

2 Consistencia Temporal en *SBDTR*

Los sistemas de la subdisciplina *SBDTR* son los aplicados en contextos dinámicos, en los cuales es necesario: detectar cambios en el ambiente y reflejarlos en datos del sistema, luego procesarlos en transacciones que acceden a una base de datos y finalmente transformarlos en salidas hacia algún actuador del sistema.

Esta dinámica está restringida por un principio rector: las transacciones se consideran satisfactorias si además de arrojar resultados lógica y aritméticamente correctos los mismos se producen antes de su tiempo de vencimiento ([12]) y éste además, se realiza dentro del tiempo de vigencia de los datos (*datadeadline*).

En un *SBDTR,* tanto las operaciones como los datos tienen constricciones temporales ([13]). Consecuentemente, surge el concepto de *datos de tiempo real* (*DTR*). Estos se clasifican en *datos base* (*DTRB*), que no dependen de otros datos y reflejan el valor de un objeto externo y *datos derivados* (*DTRD*) que se calculan en función de *DTRB* o de otros *DTRD*.

Por otra parte, las actividades de sensado del ambiente, pueden ser *pasivas* o *activas*. El *sensado pasivo,* es iniciado desde el sistema, quien encuesta al sensor para obtener su valor (mecanismo *pull*). En el *sensado activo,* por el contrario, es el sensor quien toma la iniciativa de trasmitir su valor (mecanismo *push*). A continuación se presenta un ejemplo que será utilizado a lo largo de este trabajo.

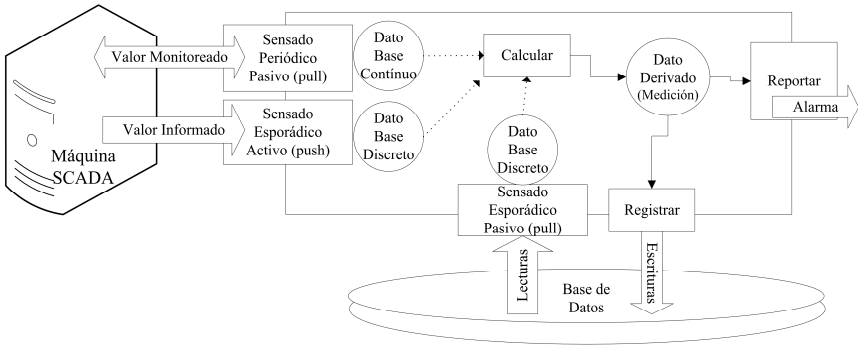

Fig. 1. Ejemplo *SBDTR*. Sistema de Tiempo Real para Monitoreo de Máquinas *SCADA*

Una fábrica posee una red de máquinas *SCADA* (*Supervisory Control And Data Acquisition*) encargadas de supervisar los procesos industriales. Para garantizar el

correcto funcionamiento de dichas máquinas y para poder adelantarse a posibles fallas se ha implementado un monitoreo automatizado. El sistema realiza mediciones de hardware (temperatura de procesador, velocidad de los ventiladores de enfriamiento, etc.), del sistema operativo (uso de CPU, uso de memoria, ejecución de procesos, reinicios, etc.) y de la aplicación *SCADA* (valores puntuales de su base de datos). Si en las mediciones se detectan valores fuera de un rango deseable, se generan alarmas para que un operador de mantenimiento atienda la situación. Las mediciones y alarmas se registran en una base de datos. Además, algunas mediciones, necesitan analizar valores históricos para generar alertas, por ejemplo, variación de carga de procesamiento en el último minuto. El caso descrito puede verse esquemáticamente en la Figura 1.

2.1 Datos Base y Consistencia Temporal Absoluta

Los *DTRB* reflejan el estado de un objeto externo y permiten detectar los cambios del ambiente. Los cambios pueden tener un comportamiento *continuo* o *discreto* [14]. Se define como *una entidad externa de cambio continuo*, a la que genera variaciones continuas en el tiempo. Por ejemplo, la temperatura del procesador o la velocidad de ventiladores de enfriamiento. A su vez, se define como *una entidad externa de cambio discreto,* a la que genera variaciones en instantes discretos y espaciados, por ejemplo, la cantidad de procesos ejecutando en la máquina SCADA o el promedio de las últimas mediciones de carga.

En función de lo anterior, los *DTRB* se clasifican, de acuerdo al objeto externo que reflejan, en *DTRB continuos* y *DTRB discretos*. Los *DTRB continuos* son actualizados con muestras periódicas. Los *DTRB discretos,* sin embargo, son actualizados en forma esporádica, solo cuando cambia el valor externo.

Un *DTRB* pierde vigencia con el correr el tiempo. El sistema debe garantizar que siempre se trabaje con *DTRB* correctamente actualizados. La garantía de que un *DTRB* es actual y equivalente con su contraparte en el ambiente se denomina *consistencia temporal absoluta* o *consistencia externa* ([15]).

Llamaremos IV_b al *Intervalo de Validez Absoluta* de un *DTRB b.* El límite inferior de dicho intervalo (*LIIV*), es el instante en el que se actualiza el dato *b*, y se llamará *Límite Superior de Intervalo de Validez absoluta* (*LSIV*), al instante en que el dato pierde vigencia.

Un *DTRB b* satisface la consistencia temporal absoluta si el instante en el que se accede (*now*(*t*)) pertenece al intervalo IV_b. Esto es:

$$LIIV_b(t) \leq now(t) \leq LSIV_b(t)$$

En un *DTRB continuo* su validez depende de su *edad*. La *edad* es el tiempo desde su creación o última actualización, hasta el instante actual. El sistema establece cual es la *edad máxima* (*MA*) ([16]) válida para un determinado dato *b*. Superada esa edad el dato pierde vigencia, esto significa: $LSIV_b(t) = LIIV_b(t) + MA_b$.

Esto no es aplicable a un *DTRB discreto*, pues en este caso el valor es válido mientras no varíe. Consecuentemente, $LSIV_b(t)$ es el instante en el que *b* renueva su valor.

2.2 Datos Derivados y Consistencia Temporal Relativa

Los *DTRD* son datos *calculados*. Su valor se determina con operaciones sobre un *Conjunto-Lectura* (*read-set*) de otros *DTR*. Un *DTRD* contiene al menos un *DTR* en su *Conjunto-Lectura*. Los objetos del *Conjunto-Lectura* de un *DTRD* deben reflejar intervalos cercanos de tiempo. A este requerimiento se lo conoce como *consistencia temporal relativa*.

El *Conjunto-Lectura* de un *DTRD d* cumple consistencia temporal relativa si existe intersección entre los *IV* de sus elementos. Esto es:

$$\bigcap \{ IV_x(t) | x \in \text{ ConjuntoLectura}_d \} \neq \varnothing$$

Esto define para un *DTRD d*, un intervalo de validez relativa IV_d con:

$$LIIV_d(t) = Max\{ LIIV_x(t) | x \in \text{ ConjuntoLectura}_d \}$$

$$LSIV_d(t) = Min\{ LSIV_x(t) | x \in \text{ ConjuntoLectura}_d \}$$

2.3 Transacciones que utilizan *DTR*

Como lo presenta Stankovic en [12], las constricciones de tiempo para las transacciones en un *SBDTR* están definidas por los intervalos de validez de los *DTR* involucrados en ella y por características propias de la transacción, como periodicidad o tiempo de respuesta anterior al vencimiento (*deadline*). En modelos de *SBDTR* se identifican dos clases de transacciones: aquellas que son de actualización de los *DTR* (base o derivados) y aquellas en las que el usuario implementa la lógica de su aplicación. Se pueden identificar:

Transacciones de Actualización de DTRB (*TA*): Transacciones de solo-escritura (*write-only*) de un *DTRB*. El sistema debe implementarlas para garantizar la consistencia temporal absoluta. Pueden ser periódicas o esporádicas de acuerdo a si actualizan *DTRB* continuos o discretos.

Transacciones de Actualización de DTRD (*TD*): Transacciones de lectura sobre algunos *DTR* y de escritura sobre otros *DTRD*. El sistema debe implementarlas para garantizar el re-cálculo de datos derivados. En ellas debe verificar que se cumpla la consistencia temporal relativa sobre el conjunto-lectura.

Transacciones del Usuario (*TU*): Implementan la lógica de la aplicación de usuario. Son transacciones de solo-lectura (*read-only*) sobre los *DTR*, aunque además utilizan datos convencionales que no tienen restricciones de tiempo real.

Las transacciones de actualización (*TA* o *TD*) pueden implementarse como transacciones independientes o como sub-transacciones de una transacción *TU*, dependiendo si la política es actualización *inmediata* o *a demanda* [17]. La transacción *inmediata* garantiza la actualización de los *DTR* independientemente de las *TU* que los usan. Mientras que transacción *a demanda* solo se ejecuta cuando una *TU* necesita acceso a los *DTR*.

En el caso de las *TA independientes,* la actualización periódica de un *DTRB continuo b* garantiza su consistencia si se ejecuta en un período $P \leq MA(b)/2$. Consecuentemente, el período no debe ser superior a la mitad de la edad máxima

([3]). Con períodos pequeños es posible que se genere demasiada sobrecarga de transacciones *TA* y *TD* encadenadas, quizás por aplicar un cambio insignificante en el objeto externo. Esto introduce el concepto de *Error Máximo* (*mde*) ([18]), el cual permite descartar *TA* si la variación entre el valor registrado del dato y el nuevo valor no es suficientemente significativa. Una *TA* se descarta si:

$$|ValorNuevo_b - ValorAnterior_b| \leq mde.$$

2.4 Versionado de *DTR*

Si *TA*, *TD* y *TU* son transacciones independientes deben considerarse conflictos de concurrencia. Esto se debe a que pueden instanciarse transacciones *TA* que necesiten reescribir un dato base cuando aún no han terminado las derivaciones *TD* o las transacciones de usuario *TU* que leen dicho dato. Si se utiliza control de concurrencia optimista esto puede resultar en una alta tasa de re-ejecución de transacciones *TD* o *TU*. Un mecanismo de bloqueo, en cambio, puede incorporar retrasos en las transacciones *TA*.

Song y Liu ([19]) han propuesto, para estos casos, el modelo de múltiples versiones de *DTR*. Cada vez que un *DTR* (base o derivado) actualiza su valor se genera una nueva versión. Cada versión tiene su propio intervalo de validez y en un determinado instante puede haber más de una versión vigente. De esta manera, las transacciones de lectura pueden ejecutarse sin realizar contención de recursos para las transacciones de escritura. La implementación de versionado de *DTR* puede incrementar el costo de la solución pero minimiza los conflictos por concurrencia y ha demostrado mejorar la performance general del sistema ([19]).

3 Tipo de Dato Abstracto para Objetos de Tiempo Real

El objetivo del presente trabajo es lograr un modelo que represente a los *DTR*, incluyendo las clasificaciones enunciadas en la sección anterior, y que garantice las correspondientes validaciones de consistencia temporal en forma encapsulada.

Se define entonces, un tipo de dato abstracto parametrizable *DTR*, que como atributos mínimos tiene un *valor* y un intervalo de validez (*LIIV, LSIV*). Luego se definen subtipos específicos *DTRBContinuo*, *DTRBDiscreto* y *DTRD*. Los mismos pueden ser utilizados para tipificar variables de transacciones o atributos de entidades en un *SBDTR*, de una manera simple, como por ejemplo:

DTRBContinuo<float> cargaProcesador;

Con esta sentencia el programador declara un objeto *cargaProcesador* que tomará valores tipo *float*, pero que además respetará las propiedades y reglas de un *DTRB* continuo. Al crear una nueva instancia se deberán indicar parámetros que permitan configurarlo, por ejemplo:

cargaProcesador = new DTRBContinuo(MA→4, mde→0.5)

Esto indica que *cargaProcesador* tendrá una edad máxima (*MA*) de 4 segundos y un error máximo (*mde*) de 0.5. Luego, en sucesivas transacciones *TA* se actualizará el valor del objeto utilizando la operación *set*. Por ejemplo:

cargaProcesador.set(*36.52*)*;*

Si esto sucede en el instante *t*, el sistema definirá el intervalo de validez de *cargaProcesador* con límites (*LIIV*→*t*, *LSIV*→*t* + 4seg), y luego le asignará el valor *36.52,* siempre y cuando tenga una diferencia superior a *0.5* con el valor registrado anteriormente (*mde*).

Las transacciones *TD* o *TU* que necesiten recuperar el valor de *cargaProcesador* utilizarán *get*() y podrán operarlo transparentemente como *float*. Por ejemplo:

cp = (*float*) *cargaProcesador.get*() * *0.01 ;*

Si esto sucede en un instante *t'* el sistema chequeará la consistencia temporal absoluta validando que (*cargaProcesador.LIIV* ≤ *t'* ≤ *cargaProcesador.LSIV*), generando un error en caso de no cumplirse.

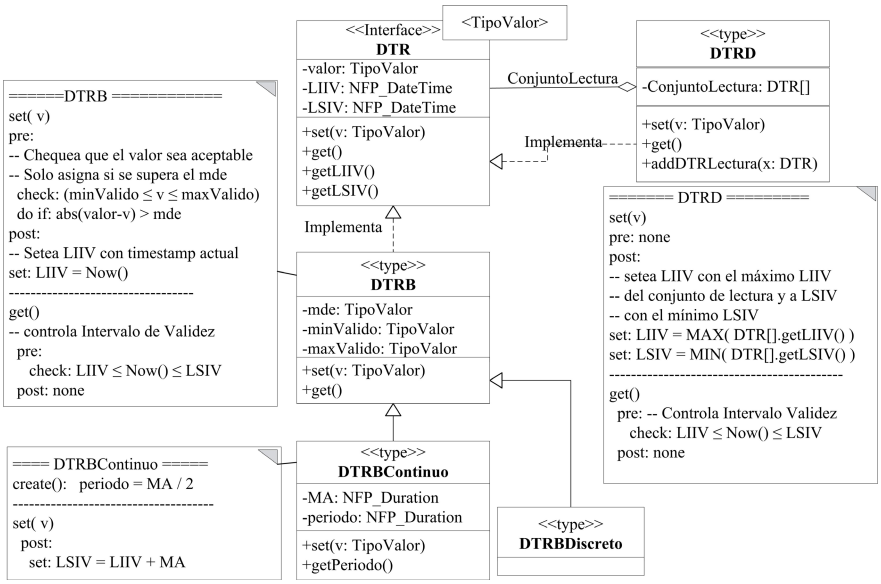

Fig. 2. Diagrama de clases del modelo propuesto. Tipo de dato abstracto *DTR*.

El intervalo de validez del *DTR* se actualiza internamente cada vez que se asigna un valor, y el chequeo de consistencia temporal se realiza cada vez que se recupera dicho valor. Tanto para *DTRB* como para *DTRD*. El modelo propuesto se presenta como diagrama de clases UML en la Figura 2.

En las notas del diagrama se describen las acciones y validaciones realizadas por cada subtipo. Se ha utilizado la definición de tipos no-funcionales de *MARTE* (*NFP_Types*) ([8]) para declarar valores y duraciones de tiempo.

Los *DTRB* definen su intervalo de validez absoluta de acuerdo al instante de asignación de valor, y además de contemplar el máximo error *mde* contemplan la definición opcional de un rango de valores válidos para descartar posibles errores de lectura (*minValido* y *maxValido*). En el caso especial de los *DTRBContinuos*, se maneja un atributo *periodo*, el cual se calcula en función de la máxima edad *MA*. Este atributo es útil a quien tenga que planificar las ejecuciones periódicas de las *TA*.

Los *DTRD* definen a su Conjunto-Lectura como una colección de *DTR*. Al momento de crear un *DTRD* se deben incluir los *DTR* que forman parte de su conjunto lectura utilizando la operación *addLectura*. Esto permite que el sistema pueda validar la consistencia temporal relativa, definiendo el intervalo de validez como la intersección de los intervalos de validez de los *DTR* de su *Conjunto-Lectura*.

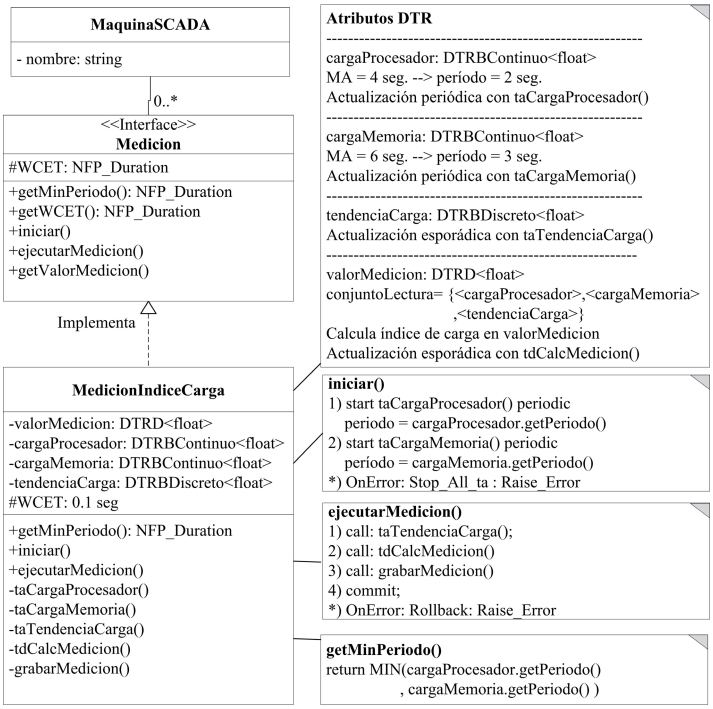

Fig. 3. Aplicación del Tipo *DTR*. Mediciones sobre máquinas *SCADA*

La verificación del modelo se realiza implementando el diseño de mediciones sobre el ejemplo de la sección 2 (Figura 1).

Sobre una determinada máquina *SCADA* se deben realizar un conjunto de mediciones. Cada una debe especificar su peor caso de tiempo de ejecución (*WCET*) y su mínimo período de ejecución, como información útil para el planificador de mediciones. Además, cada medición debe implementar las operaciones públicas:

- *iniciar*()*:* para realizar tareas de inicialización de la medición e iniciar transacciones *TA* de actualización sobre sus *DTRBContinuos.*

– *ejecutarMedicion*(): para calcular el valor de la medición y registrarlo en la base de datos.

En la Figura 3 se muestra el modelo de mediciones genéricas y se implementa una de ellas en particular: la medición del índice de carga de procesamiento de una máquina *SCADA*. Este índice se calcula en función de la carga del procesador, de la carga de memoria y de la tendencia de carga del último minuto registrada en la base de datos. La carga del procesador y la carga de memoria son *DTRBContinuos* y tienen una edad máxima de 4 seg y 6 seg. respectivamente. La tendencia de carga es un *DTRBDiscreto* que se obtiene leyendo registros históricos de la base de datos. El valor de la medición es un *DTRD* que se calcula en función de los datos anteriores. Puede verse en las notas del diagrama como se implementa el modelo y como se definen los *DTR* utilizando el tipo de dato abstracto parametrizable.

La operación *iniciar*() dispara dos *TA* independientes para obtener los *DTRBContinuos*: *taCargaProcesador*() con un período de 2 seg y *taCargaMemoria*() con un período de 3 seg. La operación *ejecutarMedicion*() primero ejecuta una *TA taTendenciaCarga*() para leer la tendencia de carga desde la base de datos, luego deriva el valor del índice de carga actual con *tdCalcMedicion*() y finalmente graba en la base de datos con la operación *grabarMedicion*().

Esta aplicación de diseño concreto permite ver como el uso del tipo de dato abstracto *DTR* facilita la definición del problema, permite organizar modularmente las transacciones y desliga al programador de la validación de consistencia temporal.

4 Conclusiones y Trabajos Futuros

Se ha logrado encapsular el conjunto de reglas y características referidas a *consistencia temporal de datos de tiempo real* dentro de un tipo de dato abstracto parametrizable que puede ser aplicado directamente sobre variables o atributos de entidades en un *STR*. También ha sido posible verificarlo aplicándolo en el diseño de clases de una aplicación concreta. En esta etapa de verificación, al plantear el esquema de tareas, se ha podido ver que así como fue posible un modelo reusable para *DTR* también es posible un modelo reusable para transacciones *TA, TD y TU*. Los trabajos a futuro serán obtener un modelado genérico de operaciones de tiempo real que involucren *DTR*, con el objeto de facilitar el diseño y la implementación de transacciones en un *SBDTR*.

Referencias

[1] K. Ramamritham, "Real Time Databases," *International Journal of Distributed and Parallel Databases,* vol. 1, pp. 199-226, 1993.

[2] B. Purimetla, R. Sivasankaran, K. Ramamritham, and J. Stankovic, "Real-Time Databases: Issues and Applications," in. vol. ch.20, ed: in S.Son (ed.) Advances in Real-Time Systems, Prentice Hall, 1995.

[3] M. Xiong, J. A. Stankovic, K. Ramamritham, D. Towsley, and R. Sivasankaran, "Maintaining Temporal Consistency: Issues and Algorithms," in *Proceedings of International Workshop on Real-Time Database Systems*, 1996, pp. 2-7.

[4] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," in *Proceedings of the 14th VLDB Conference*, 1988.

[5] C. Combi, S. Degani, and C. S. Jensen, "Capturing Temporal Constraints in Temporal ER Models," in *Proceedings of the 27th International Conference on Conceptual Modeling*, ed Berlin, Heidelberg: Springer-Verlag, 2008, pp. 397-411.

[6] Ellen Rose and A. Segev, "TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints," in *Proceedings of the 10th International Conference on Entity-Relationship Approach (ER'91)*, San Mateo, California, USA, 1991.

[7] B. P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*: Addison-Wesley, 2002.

[8] O. M. G. (OMG). (2009, OMG Document Number: formal/2009-11-02). *A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems*. Available: http://www.omg.org/spec/MARTE/1.0/

[9] J. J. Prichard, L. C. DiPippo, J. Peckham, and V. F. Wolfe, "RTSORAC: A Real-Time Object-Oriented Database Model," *In The 5th International Conference on Database and Expert Systems Applications,* pp. 601-610, 1994.

[10] L. C. DiPippo and L. Ma, "A UML Package for Specifying Real-Time Objects," *Computer Standards & Interfaces* vol. 22, pp. 307-321, 2000.

[11] N. Idoudi, C. Duvallet, B. Sadeg, R. Bouaziz, and F. Gargouri, "Structural Model of Real-Time Databases: An Illustration," in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, 2008, pp. 58-65.

[12] J. A. Stankovic, S. Son, and J. Hansson, "Misconceptions About Real-Time Databases," *IEEE Computer,* vol. 32, pp. 29-36, 1998.

[13] K. Ramamritham, S. H. Son, and L. C. Dipippo, "Real-Time Databases and Data Services," *Real-Time Syst. Kluwer Academic Publishers,* vol. 28, pp. 179-215, 2004.

[14] K. Ben, L. Kam-Yiu, B. Adelberg, R. Cheng, and T. Lee, "Maintaining temporal consistency of discrete objects in soft real-time database systems," *Computers, IEEE Transactions on,* vol. 52, pp. 373-389, 2003.

[15] P. Yu, K. Wu, K. Lin, and S. H. Son, "On Real-Time Databases: Concurrency Control and Scheduling," in *Special Issue on Real-Time Systems*, Proceedings of IEEE, 1994, pp. 140-157.

[16] B. Adelberg, H. García-Molina, and B. Kao, "Applying update streams in a soft real-time database system," *Proceedings of the 1995 ACM SIGMOD,* vol. 24, pp. 245-256, 1995.

[17] Y. Wei, J. A. Stankovic, and S. H. Son, "Maintaining Data Freshness in Distributed Real-Time Databases," presented at the Proceedings of the 16th Euromicro Conference on Real-Time Systems, 2004.

[18] M. Amirijoo, J. Hansson, and S. H. Son, "Specification and management of QoS in real-time databases supporting imprecise computations," *Computers, IEEE Transactions on,* vol. 55, pp. 304-319, 2006.

[19] X. Song and J. Liu, "Maintaining temporal consistency: pessimistic vs. optimistic concurrency control," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 7, pp. 786-796, 1995.