Framework para modelado de Transacciones en Sistemas de Bases de Datos de Tiempo Real

Carlos E. Buckle, José M. Urriza, Damián P. Barry, Lucas Schorb

Depto Informática, Facultad Ingeniería, Universidad Nacional de La Patagonia San Juan Bosco
Puerto Madryn, Argentina
+54 280-4472885 – Int. 117.
cbuckle@unpata.edu.ar, josemurriza@unp.edu.ar, damian_barry@unpata.edu.ar
.

Resumen. En los Sistemas de Bases de Datos de Tiempo Real, se requiere extender el modelo tradicional de transacciones para incorporar restricciones temporales. En este tipo de sistemas, una transacción se debe ejecutar dentro de un intervalo de tiempo específico, predeterminado con anterioridad. Además, se debe garantizar la validez de ciertos datos, cuyo valor caduca con el paso del tiempo. Los conceptos teóricos para modelado de transacciones de tiempo real, han sido desarrollados en varios trabajos del área. En este trabajo, se reúnen y exponen los conceptos principales en lo referido a consistencia temporal de datos y transacciones. Luego, se los incorpora a un framework orientado a objetos que guía y facilita el modelado de transacciones con restricciones de tiempo. El uso del framework se presenta y se verifica aplicándolo sobre un caso de ejemplo.

Palabras Clave: Transacciones de Tiempo Real, Consistencia Temporal, Bases de Datos de Tiempo Real, Ingeniería de Software de Tiempo Real, Datos de Tiempo Real.

1 Introducción

El procesamiento de *transacciones* ([1]), es la técnica utilizada para el acceso concurrente, consistente y tolerante a fallas en un Sistema de Bases de Datos (*DBS*) convencional. Los Sistemas de Tiempo Real (*RTS*) que manejan datos persistentes sobre un *DBS,* imponen que el modelo de transacciones sea ampliado para considerar restricciones temporales. Por ejemplo, en sistemas de supervisión y monitoreo de procesos (*SCADA*), en tableros de comando (*Balanced Scorecard*), en ambientes de transacciones on-line de compra/venta de acciones bursátiles (*Online Stock Trading*), etc. Esta aproximación entre las disciplinas de *DBS* y *RTS,* ha dado origen a los *Sistemas de Bases de Datos de Tiempo Real* (*RTDBS*) ([2, 3]). Los cuales, además de administrar un gran volumen de datos convencionales, deben manejar datos que se encargan de reflejar el estado de elementos variables del ambiente. Estos datos, tienen como característica que su valor *envejece,* hasta perder vigencia una vez alcanzado su

vencimiento (*Datadeadline* [4]). Por ese motivo, son identificados en el sistema como *datos de tiempo real* (*RTD Real-Time Data*). Mantener correctamente actualizados estos objetos, permite garantizar la consistencia entre el ambiente supervisado y el *RTDBS*. En este escenario, las transacciones además de garantizar consistencia lógica, deben garantizar *consistencia temporal* ([2]), dando origen a lo que se denomina *transacciones de tiempo real* (*RTT*) (*Real-Time Transaction*) ([5], [6]). Sobre ellas debe garantizarse: su instante mínimo de inicio (*arrival time*), su tiempo de respuesta previo al vencimiento (*deadline*) y la validez de los *RTD* involucrados.

Estas consideraciones, introducen una complejidad adicional al momento de construir aplicaciones. Además de la lógica propia del dominio del problema a resolver, deben atenderse otras cuestiones como: la contabilización de tiempos, la validación de reglas y la planificación de tareas para garantizar la *consistencia temporal* de datos y transacciones. Surge así la necesidad de contar con herramientas que incorporen estos conceptos, orienten el modelado y aporten la información temporal necesaria para que el planificador del *RTS* pueda ordenar debidamente la ejecución de las transacciones.

En el presente trabajo se presenta un *framework* orientado a objetos, para el modelado de *RTT*, como un aporte concreto para los diseñadores de *RTDBS*.

Este documento está organizado de la siguiente manera: en la sección 2 se presentan trabajos anteriores relacionados con el modelado de *RTDS*. En la sección 3 se presenta un conjunto de definiciones y clasificaciones asociadas con el concepto de *RTT*. En la sección 4 se presenta el *framework* desarrollado. En la sección 5 se aplica el *framework* sobre un caso de estudio. En la sección 6 se elaboran las conclusiones y se plantean los trabajos a futuro.

2 Trabajos previos

En el pasado, se han desarrollado una serie de trabajos que apuntan a modelar objetos y transacciones dentro de un *RTDBS*. Uno de los trabajos más importantes en el modelado orientado a objetos es *Real Time Semantic Objects Relationships And Constraints* (*RTSORAC*) ([7]), en el cual se definen tres propiedades básicas de los *RTDBS*: objetos, relaciones y transacciones de tiempo real. El modelo teórico contempla la mayoría de los requerimientos temporales y permite expresar las diferentes entidades que intervienen en un *RTDBS*. Sin embargo, es demasiado extenso y genérico como para ser aplicado directamente en el diseño de aplicaciones concretas. No obstante, es utilizado en un conjunto de trabajos posteriores que lo utilizan como base para la definición de perfiles de diseño, como los presentados en [8] y [9]. Dichos trabajos, se enfocan preferentemente a la definición de objetos de tiempo real y no al modelado de transacciones. Tampoco se consideran objetos de cambio discreto ([10]), ni datos derivados (datos calculados). El patrón de diseño presentado en [11], incorpora el modelado de transacciones pero solo de aquellas encargadas de actualizar datos desde sensores. El marco de trabajo (*framework)* presentado en [12], clasifica aquellas transacciones que actualizan los datos desde

sensores y aquellas que propagan derivaciones de los datos calculados, pero no incluye a las transacciones del usuario que resuelven la lógica de la aplicación.

En lo que sigue, se presenta un *framework* orientado a objetos para el modelado de *RTT*, en el cual se consideran aspectos ampliados respecto de los trabajos antes mencionados. El *framework* presentado utiliza el *Tipo de Dato Abstracto para Bases de Datos de Tiempo Real RTD* ([13]), resultante de un trabajo anterior, el cual se toma como base y permite definir cualquier tipo de atributo de tiempo real encapsulando la validación de *consistencia temporal*.

3 Gestión de Transacciones con Restricciones Temporales

El modelo de *RTT,* puede definirse como una extensión del modelo tradicional de transacciones ([14]), el cual se resume brevemente a continuación: Un *DBS* es un conjunto de *entidades de datos,* que representan información sobre un contexto determinado. El mapeo entre entidades y sus valores, define el *estado* de la base de datos en un determinado instante. Sobre ella se definen un conjunto de operaciones que permiten recuperar, crear, modificar y eliminar entidades. Estas operaciones provocan la transición de un estado a otro. Una *transacción*, es un conjunto de operaciones parcialmente ordenadas sobre la base de datos que debe ser ejecutada atómicamente. El orden parcial de las operaciones está dado por un algoritmo. La *atomicidad* significa que la transacción debe ser ejecutada satisfactoriamente o sino no debe ser ejecutada. Para esto el *DBS* ofrece dos operaciones: *commit,* que confirma la finalización satisfactoria de una transacción y *rollback,* que deshace la ejecución parcial de operaciones realizadas por una transacción que no puede finalizar satisfactoriamente. La transacción pasa de un estado *consistente* de la base de datos, a un próximo estado consistente. La ejecución de las operaciones ordenadas de una transacción se asume correcta, si se ejecuta en forma aislada. El *aislamiento,* oculta los cambios parciales que realiza una transacción hasta su finalización. Si la transacción finaliza con *commit* se garantiza la *durabilidad* (persistencia) de los resultados en la base de datos. De esta forma se han definido las características *ACID* (*atomicidad*, *consistencia*, *aislamiento* y *durabilidad*).

Sobre este modelo tradicional de transacciones, se puede incorporar la noción de *tiempo real*. Una transacción puede conocer el *tiempo actual* (*now*) accediendo a una entidad única del sistema llamada *reloj* (*clock)*. Esta entidad es solo-lectura y toma valores positivos que se incrementan monotónicamente en concordancia con el paso del tiempo. Esto permite establecer intervalos de vigencia sobre los *RTD* y establecer restricciones temporales sobre las transacciones. Si una transacción no cumple estas restricciones temporales al momento del *commit*, se debe deshacer (*rollback*) y si corresponde, volverse a ejecutar (*restart*).

3.1 Consistencia Temporal de los Datos de Tiempo Real

Los *RTD* reflejan objetos cambiantes del ambiente. Existen dos tipos ([13]): Los *RTDBase*, los cuales reflejan el estado de un objeto externo y actualizan su valor desde sensores o publicadores y los *RTDDerived* que son datos derivados, cuyo valor se determina con cálculos sobre un *Conjunto-Lectura* (*Read Set*) con otros *RTD*.

La *consistencia temporal* de los *RTD* garantiza:

- *validez temporal*: El valor de un *RTD* se considera válido dentro de un *intervalo de validez*, *VI* (*validity interval*). En el cual el límite inferior (*VILB*) es el momento en el que se actualiza el valor y el límite superior (*VIUB*) es el instante en el que el valor pierde vigencia (*DataDeadline*). Un *RTD* respeta *consistencia temporal absoluta* ([10]) en el instante *t* si $VILB_{dtr}(t) \leq now(t) \leq VIUB_{dtr}(t)$.
- *coherencia temporal*: Se debe garantizar que los datos calculados se deriven en base a los *RTD* actualizados en instantes cercanos de tiempo. Por esto, para los *RTDDerived* se debe garantizar *consistencia temporal relativa* ([10]) sobre todo su conjunto lectura, es decir, $\bigcap\{VI_x(t) | x \in \text{ReadSet}_{dtr}\} \neq \varnothing$ y su *VI* se calcula como:
  $VILB_{dtr}(t) = Max\{VILB_x(t) | x \in \text{ReadSet}_{dtr}\}$ y $VIUB_{dtr}(t) = Min\{VIUB_x(t) | x \in \text{ReadSet}_{dtr}\}$

Los *RTDBase* pueden ser continuos (*RTDBaseContinuous*) ó discretos (*RTDBaseDiscrete*). Los continuos, son actualizados con muestras periódicas y su *VI* depende de su *edad* (tiempo desde su última actualización). El vencimiento del dato se define en base a establecer la *edad máxima* (*maximumAge*) ([15]) y es posible garantizar la *consistencia temporal absoluta*, si se considera un período de actualización $P \leq maximumAge/2$ ([4]). Estos conceptos se desarrollaron en la definición del tipo de dato *abstracto RTD* presentado en [13].

La actualización de los *RTDBase* puede implementarse utilizando dos políticas: *actualización inmediata* o *a demanda* ([16]). La *actualización inmediata* garantiza que cada cambio en un objeto externo será inmediatamente reflejado en el *RTDBase* que lo representa. La política de actualización *a demanda*, significa que el *RTDBase* será actualizado solo cuando una transacción necesite utilizarlo. La *actualización inmediata* procura un sistema más predecible, pero genera una carga innecesaria al actualizar valores de *RTD* que quizás no requiera ninguna transacción. La *actualización a demanda* soluciona este problema, pero introduce un tiempo de latencia en las transacciones pues deben actualizar los *RTDBase* antes de utilizarlos.

3.2 Características de las Transacciones de Tiempo Real.

Como fue presentado inicialmente, una *RTT* tiene las restricciones temporales propias: un tiempo mínimo de inicio (*startTime*) y un tiempo de respuesta anterior al vencimiento (*deadline*). Además, las *RTT* están condicionadas por la validez temporal de los *RTD* involucrados en ella. En un *RTDBS* se identifican tres clases de *RTT*:

- *RTT de Actualización de RTDBase* (*RTTUpdate*): Son transacciones de solo-escritura sobre un conjunto de *RTDBase,* llamado *Write Set*. El sistema debe implementarlas para garantizar la *consistencia temporal absoluta* de los objetos del *Write Set*.
- *RTT de Derivación de RTDDerived* (*RTTDerivation*): Son transacciones de lectura sobre un *Read Set* de *RTD* y de escritura sobre un *Write Set* de *RTD*. El sistema debe implementarlas para garantizar el re-cálculo de datos derivados.

- *RTT del Usuario* (*RTTUsrApp*): Implementan la lógica de la aplicación de usuario. Son transacciones de solo-lectura sobre un *Read Set* de *RTD*, aunque además, utilizan datos convencionales que no tienen restricciones de tiempo real.

Dependiendo de la política de actualización de los *RTDBase*, las *RTTUpdate* se pueden implementar como transacciones independientes (en *actualización inmediata*), o como sub-transacciones de una *RTTUsrApp* (en *actualización a demanda*). Las transacciones independientes se deben planificar con un período *P* igual al menor período de los *RTDBase* en el *Write Set.* Su *DataDeadline* se puede calcular como el menor *VIUB* de dicho conjunto, generalmente como *P*2* ([4]).

Las *RTTDerived* pueden ser transacciones independientes o transacciones disparadas (*triggered*) por una *RTTUpdate* [16]. Si son independientes se debe planificar con un período *P* igual al menor período de los *RTD* en el *Read Set*. Su *DataDeadline* es el menor *VIUB* de todos los *RTD* incluidos en el *Read Set*.

Cada *RTT,* independientemente de su clase, puede tener su propio requerimiento de tiempo de respuesta, por ende, debe poder indicarse un tiempo para el vencimiento (*timeToDeadline*) propio de la transacción. Si no se explicita un *timeToDeadline* se puede considerar como máximo un *timeToDeadline = P* (período de la *RTT*). Para que sea posible la planificación, también es necesario poder estimar el peor caso de tiempo de ejecución de cada transacción (*Worst Case Execution Time – WCET*).

3.3 Planificación de Transacciones

Para garantizar las restricciones temporales mencionadas, es necesario que el *RTDBS* cuente con un planificador de tiempo real (*real-time scheduler*) que ordene la ejecución concurrente de transacciones. Para que esto sea posible, se debe manejar un conjunto de atributos y medidores de performance de cada *RTT*. Mínimamente se requiere:

- *arrivalTime*: Instante en que la *RTT* se pone lista para ejecutar.
- *startTime*: Instante en que la *RTT* comienza a ejecutar.
- *WCET*. *Worst case execution time:* Tiempo máximo de ejecución estimado.
- *period*: Magnitud del período, para aquellas *RTT* de ejecución periódica.
- *timeToDeadline*: lapso de tiempo para el *deadline.*
- *deadline*: vencimiento de la *RTT= (arrivalTime + timeToDeadline).*
- *dataDeadline(t)*: Menor *DataDeadline* de los *RTD* del *Read Set* al instante *t*.
- *estRemaining(t)*: lapso de ejecución remanente de la *RTT* al instante *t.*
- *estCompletionTime(t)*: Instante estimado de fin en *t* = (*t* + *estRemaining(t)*).
- *estSlack(t)*: lapso que se puede retrasar = *deadline – estCompletionTime(t)*.

En función de algunos de estos parámetros, el planificador de tiempo real debe determinar en cada instante *t* de planificación la prioridad de la *RTT*.

- $priority_{rtt}(t)$: prioridad de la *RTT* en el instante *t*.

El algoritmo de planificación, debe garantizar que las *RTT* del sistema se ejecuten antes de su *deadline* y que cumplan con las reglas de *consistencia temporal*. Para esto se debe implementar una política de planificación. Algunas de las posibles son:

- *Rate Monotonic* (*RM*) ([17]): Prioridad basada en la magnitud del *período*.
- *Earliest-Deadline-First* (*EDF*) ([17]): Prioridad basada en el *deadline.*
- *Earliest-DataDeadline-First* (*EDDF*) ([4]): Prioridad basada en el *dataDeadline.*
- *Highest-Value-First (HVF)* ([18]): Prioridad basada en el la *función valor* (Fig. 1).
- *Least-Slack-First* (*LSF*)([5]): Prioridad basada en el tiempo disponible (*estSlack*).
- *Shortest-Job-First (SJF):* Prioridad basada en el tiempo que resta (*estRemaining*).

Estas políticas se pueden mejorar con el agregado de los conceptos de *función valor* y *espera inducida*. Estos se describen en los siguientes párrafos.

Las *RTT* pueden tener diferentes niveles de criticidad en su vencimiento. Estos niveles de criticidad, influyen en las decisiones del planificador, el cual puede considerar el concepto de *Función Valor* introducido por Jensen en [18]. La *función valor* de una transacción permite medir el *valor* (*ganancia*) del sistema si finaliza la transacción en un determinado instante *t*. Los valores positivos son deseables y los negativos indeseables. Esta *función valor* presenta una discontinuidad en el vencimiento de la transacción, dependiendo de la cual se identifican vencimientos *duros* (*hard-real-time*), *blandos* (*soft-real-time*) o firmes (*firm-real-time*) (Figura 1).

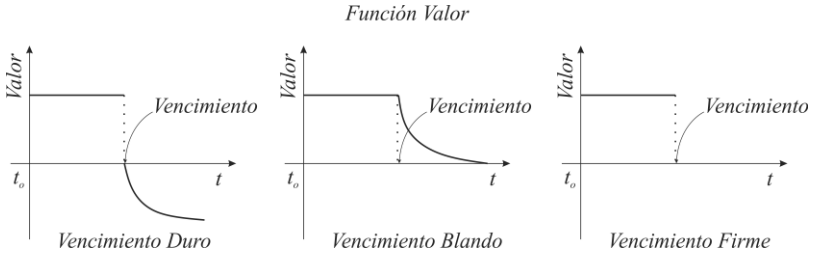

Figura 1. Función Valor.

Una decisión posible para el sistema, es que si al momento de fin de la *RTT* se obtiene una *función valor* negativa se debe deshacer la transacción (*rollback*).

Por otro lado, tampoco es posible finalizar satisfactoriamente una transacción cuyo *DataDeadline* expira antes que sea posible realizar el *commit*. Si se puede calcular el parámetro *estCompletionTime(t)* de una *RTT,* entonces es posible predecir si se podrá alcanzar el *commit* antes de que expire el intervalo de validez de los *RTD*. Si esto no es posible, el planificador puede analizar la posibilidad de retardar la transacción hasta que sus *RTD* se actualicen nuevamente. A este mecanismo se lo conoce como *Espera inducida* (*Forced Wait)*([19]). El planificador debe contemplar:

IF $dataDeadline_{RTT}(t) < estCompletionTime_{RTT}(t)$: WAIT UNTIL $dataDeadline_{RTT}(t)$

4 Framework RTT para Transacciones de Tiempo Real

La construcción de *RTDBS* requiere considerar los conceptos descriptos en el punto anterior. Resulta útil para los desarrolladores de aplicaciones contar con un marco de trabajo que facilite el diseño de *RTT* incluyendo atributos, servicios y reglas de validación de *consistencia temporal*. De esa manera, el desarrollador se puede enfocar más específicamente a la problemática propia del sistema a modelar.

En este trabajo se propone el *framework RTT*. Este es, un modelo orientado a objetos que contempla las características enunciadas para las *RTT* y que ofrece la información necesaria para que el *planificador de tiempo real* pueda implementar cualquiera de las políticas de planificación.

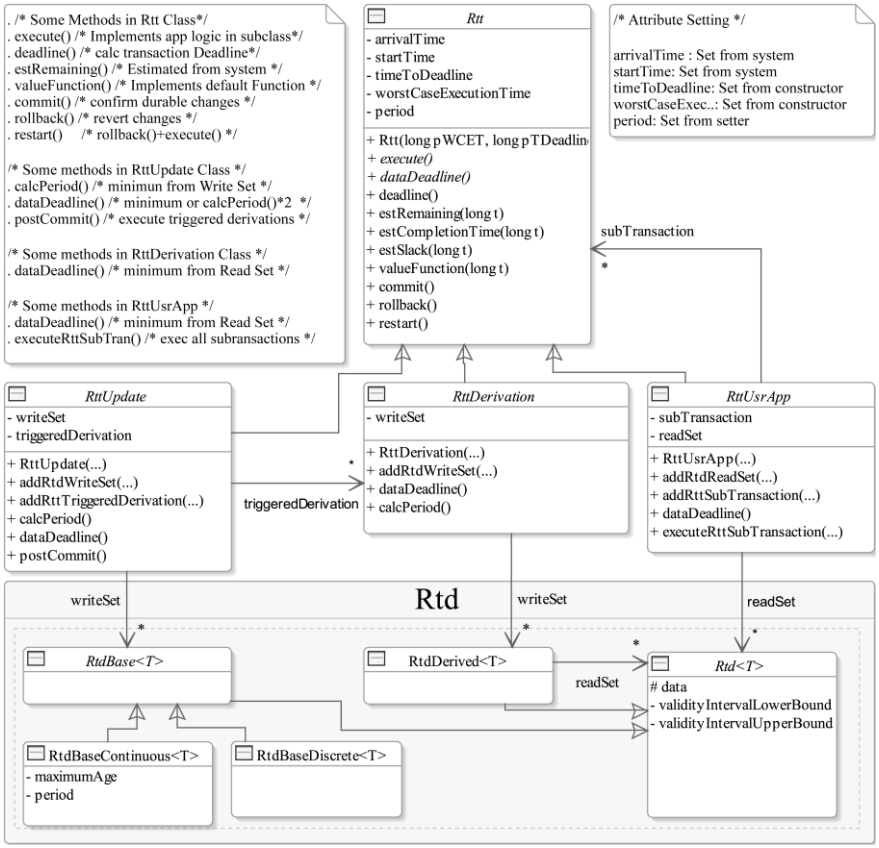

Figura 2. Framwork *RTT*

Para los datos de tiempo real se ha utilizado un paquete con el tipo de dato abstracto *RTD* ([13]), el cual caracteriza las diferentes clases de los *RTD*, encapsula la validación de *consistencia temporal* de los datos y calcula el *Datadeadline* de cada objeto. En la Figura 2 se describe el *framework RTT* en lenguaje *UML*.

El *framework* presenta cuatro clases abstractas. Una clase principal *Rtt* que define atributos y métodos comunes a todas las transacciones de tiempo real y tres subclases que implementan las particularidades de *RttUpdate*, *RttDerivation* y *RttUsrApp* como fue descrito en el punto 3.2. La subclase *RttUpdate,* permite asociar una lista de *RttDerivation* que serán disparadas (*triggered*) en el *postCommit*(). La subclase *RttUsrApp,* permite asociar una lista de subtransacciones *Rtt* que serán ejecutadas al invocar *executeRttSubTransaction*().

Los atributos, métodos y conjuntos (*Read Set* y *Write Set*) de cada clase fueron descriptos en 3.3 y se realizan aclaraciones adicionales en notas de la Figura 2.

El desarrollador de aplicaciones deberá implementar las *RTT* del sistema simplemente extendiendo la subclase que corresponda y definiendo la lógica de la transacción dentro del método *execute*(). Adicionalmente, se puede definir la *Función Valor* (*valueFunction*()) de acuerdo a si se desea implementar una *RTT* de vencimiento *duro*, *firme* o *blando*. Por defecto, *Rtt.valueFunction*() se define con vencimientos *firmes*. En la siguiente sección, se muestra el uso del *framework RTT* aplicándolo sobre un caso de ejemplo.

5 Aplicación en un caso de ejemplo

Se desea implementar un monitoreo de salud automatizado sobre una máquina *SCADA* (*Supervisory Control And Data Acquisition*). Se realizan diversas mediciones, entre ellas: la carga de trabajo (*measureVal*), la cual se calcula en función de la carga de CPU (*cpuWorkLoad*), la carga de memoria (*memWorkLoad*) y la tendencia de carga de los últimos minutos (*loadTrend*). Las mediciones tomadas se registran en la base de datos cada 20 segundos.

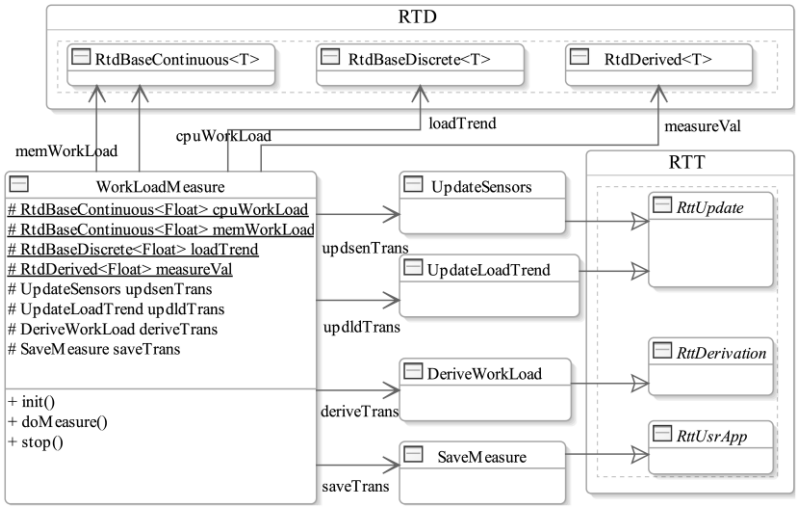

Figura 3. Aplicación del Framwork *RTT* para implementar sistema de mediciones

La carga de CPU y de memoria, se obtienen con sensados periódicos cada 4 y 6 segundos respectivamente. La tendencia de carga se obtiene desde la base de datos.

Para implementar este *RTDBS* se aplica el *framework RTT* y el diseño de transacciones se muestra en la Figura 3. En el diagrama se definen los *RTD* necesarios y cuatro *RTT* encargadas de obtener datos desde los sensores, calcular la medición y grabar en la base de datos. La clase *WorkLoadMeasure* es la encargada de automatizar la medición. En la Figura 4 se muestra la lógica aplicada:

```
init() :
 /*Define DTR para sensores */
 cpuWorkLoad = new RtdBaseContinuous<Float>(4 secs);
 memWorkLoad = new RtdBaseContinuous<Float>(6 secs);

 /*Define DTR derivado. Calcula la medición WorkLoad */
 measureVal = new RtdDerived<Float>();
 measureVal.addRtdReadSet(cpuWorkLoad);
 measureVal.addRtdReadSet(memWorkLoad);

/* Define transacción de Cálculo de derivaciones */
deriveTrans = new DeriveWorkLoad(pWCET→0.02 secs,
                                 pTDeadline→4 secs);
deriveTrans.addRtdWriteSet(measureVal);

/* Define transacción de Update de sensores */
updsenTrans = new UpdateSensors(pWCET →0.05 secs ,
                                pTDeadline→4 secs);
updsenTrans.addRtdWriteSet(cpuWorkLoad);
updsenTrans.addRtdWriteSet(memWorkLoad);
updsenTrans.addRttTriggeredDerivation(deriveTrans);

/* Tendencia de carga con dato de sensado discreto */
 loadTrend = new RtdBaseDiscrete<Float>();

 /* Define transacción de Update de sensado discreto */
 updldTrans = new UpdateLoadTrend(pWCET→0.1 secs,
                                  pTDeadline →4 secs);
 updldTrans.addRtdWriteSet(loadTrend);

 /* Define transacción de Grabar Medición */
 saveTrans = new SaveMeasure(pWCET →0.1 secs,
                             pTDeadline→20 secs);
 saveTrans.addRtdReadSet(measureVal);
 saveTrans.addRttSubTransaction(deriveTrans);

doMeasure() :
  /* Planifica Update de Sensores en RTT periódica*/
 updsenTrans.setPeriod(updsenTrans.calcPeriod());
 RTTScheduler.dispatch(updsenTrans, _PERIODIC);

 /* Planifica Registro de Medición en RTT periódica*/
 saveTrans.setPeriod(20 secs.);
 RTTScheduler.dispatch(saveTrans, _PERIODIC);
```

Figura 4. Métodos de la clase *WorkLoadMeasure*

El planificador de tiempo real *RTTScheduler* instancia las transacciones en el período indicado por el atributo *period* de la *RTT* pasada como parámetro y al llegar el momento de inicio, la ejecuta invocando al método *execute*().

6 Conclusiones y Trabajos Futuros

El trabajo presenta en detalle los conceptos de *consistencia temporal* en transacciones de tiempo real, que han sido presentados en diferentes trabajos del área. El objetivo principal fue elaborar una herramienta que guíe y simplifique el diseño de transacciones en un *Sistemas de Bases de Datos de Tiempo Real*. Consecuentemente, se desarrolló el *framework RTT*, que es un modelo orientado a objetos que incorpora clasificaciones, atributos y servicios necesarios, para implementar los tres tipos de *RTT* identificadas en el trabajo. Con esta herramienta, el desarrollador de aplicaciones se puede enfocar específicamente en el problema a resolver y dejar bajo la responsabilidad del *framework* y del *real-time scheduler,* lo referido a gestión de las transacciones de tiempo real. Finalmente y a modo de verificación, se aplica el *framework RTT* en la resolución de un problema concreto. No obstante, algunas cuestiones como control de concurrencia, tolerancia a fallos, ambientes distribuidos, infraestructuras de desarrollo, etc. no han sido consideradas. Estos temas se desarrollarán en futuros trabajos sobre esta temática.

Referencias

[1] R. Elmasri and S. Navathe, *Fundamentals of Database Systems 5/E*, 5/E ed.: Addison-Wesley, 2007.

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

[3] B. Purimetla*, et al.*, "Real-Time Databases: Issues and Applications," in. vol. ch.20, ed: in S.Son (ed.) Advances in Real-Time Systems, Prentice Hall, 1995.

[4] M. Xiong*, et al.*, "Maintaining Temporal Consistency: Issues and Algorithms," in *Proceedings of International Workshop on Real-Time Database Systems*, 1996, pp. 2-7.

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

[6] J. A. Stankovic*, et al.*, "Misconceptions About Real-Time Databases," *IEEE Computer,* vol. 32, pp. 29-36, 1998.

[7] J. J. Prichard*, et al.*, "RTSORAC: A Real-Time Object-Oriented Database Model," *In The 5th International Conference on Database and Expert Systems Applications,* pp. 601-610, 1994.

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

[9] N. Idoudi*, et al.*, "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.

[10] K. Ben*, et al.*, "Maintaining temporal consistency of discrete objects in soft real-time database systems," *Computers, IEEE Transactions on,* vol. 52, pp. 373-389, 2003.

[11] S. Rekhis*, et al.*, "Modeling Real-Time applications with Reusable Design Patterns," *International Journal of Advanced Science and Technology,* vol. 22, pp. 71-86, 2010.

[12] A. Hala*, et al.*, "A General Framework for Modeling Replicated Real-Time Database," *International Journal of Electrical and Computer Engineering. Word Academy of Science, Engineering and Technology,* pp. 4-8, 2009.

[13] C. Buckle*, et al.*, "Abstract Data Type for Real-Time Database Systems," in *XVII Congreso Argentino de Ciencias de la Computación*, UNLP. La Plata, Argentina, 2011.

[14] Soparkar*, et al.*, *Time-Constrained Transaction Management: Real-Time Constraints in Database Transaction Systems*: Kluwer Academic Publishers, 1996.

[15] B. Adelberg*, et al.*, "Applying update streams in a soft real-time database system," *Proceedings of the 1995 ACM SIGMOD,* vol. 24, pp. 245-256, 1995.

[16] Y. Wei*, et al.*, "Maintaining Data Freshness in Distributed Real-Time Databases," presented at the Proceedings of the 16th Euromicro Conference on Real-Time Systems, 2004.

[17] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM,* vol. 20, pp. 46-61, 1973.

[18] E. D. Jensen*, et al.*, "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings of Real-Time Systems Symposium,* pp. 112-122, 1985.

[19] M. Xiong*, et al.*, "Scheduling access to Temporal Data in Real-Time Databases," in *Real-Time Database Systems: Issues and Applications*, Bestavros*, et al.*, Eds., ed: Kluwer Academic Publishers, 1997, pp. 167-191.