World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:4, 2009

A General Framework for Modeling Replicated Real-Time Database

Hala Abdel hameed, Hazem M. El-Bakry and Torky Sultan

***Abstract*—**There are many issues that affect modeling and designing real-time databases. One of those issues is maintaining consistency between the actual state of the real-time object of the external environment and its images as reflected by all its replicas distributed over multiple nodes. The need to improve the scalability is another important issue. In this paper, we present a general framework to design a replicated real-time database for small to medium scale systems and maintain all timing constrains. In order to extend the idea for modeling a large scale database, we present a general outline that consider improving the scalability by using an existing static segmentation algorithm applied on the whole database, with the intent to lower the degree of replication, enables segments to have individual degrees of replication with the purpose of avoiding excessive resource usage, which all together contribute in solving the scalability problem for DRTDBS.

***Keywords*—**Database modeling, Distributed database, Real-time databases, Replication.

I. Introduction

MANY real-time applications are inherently distributed in nature, and need to share data that are distributed among different sites. For example, military tracking, medical monitoring, naval combat control systems and factory automation etc. Such applications introduce the need for distributed real time database systems (DRTDBs) [9]. A DRTDBS is a collection of multiple, logically interrelated databases distributed over a computer network [4]. A real-time database has two distinguishing features: the nature of its data that has a temporal constrains, and marinating a real-time constraints on transactions [7]. Transactions in a real time database are classified into three types, viz. hard, soft and firm. A general model of distributed real-time systems was presented by Kopetz & Verissimo (1994) [8]. This model based on the interaction between real-time entities which is an element of the environment whose state is relevant to the DRTS such as temperature and pressure. A DRTS observes or modifies the states of RT entities; for example, based on an observation of the fluid level in a tank, the system could modify the position of a valve that affects the fluid drain. A DRTS interacts with the environment via sensors (hardware that samples the state of RT entities, such as temperature and motion sensors).

Many issues affecting the design of A DRTDBS to maintain its requirements; Data Consistency and Scalability are the main issues that are considered in this paper. All of those critical systems need data to be obtained and updated in a timely fashion, but sometimes data that is required at a particular location is not available when it is needed and getting it from remote site may take too long before which the data may become invalid, this potentially leads to large number of transactions miss their deadline and violating the timing constraints of the requesting transaction. One of the solutions for the above-mentioned problem is replication of data in real-time databases. By replicating temporal data items, instead of asking for remote data access requests, transactions that need to read remote data can now access the locally available copies which help transactions meet their time and data freshness requirements. Replication in DRTDBs also, is used to remove unpredictability of network delays or network partitioning, that the database is fully replicated to all nodes. It also improves fault tolerance for the main-memory resident data. In order to suite the different needs of the distributed real-time systems such as different data workloads and database specifications, multiple ways to handle the replication control and different replication schemes are proposed. However, such a database has another scalability problem since an update to an object of a fully replicated database needs to be sent to all other nodes.

In a replicated system we can define three different types of predicators for consistency; (i) *External temporal consistency,* which deals with the relationship between an object of the external world and its image on the server. (ii) *Inter-object temporal consistency* which is the relationship between different objects or events (within a single node) and it also include the relationship between temporal data item and non-temporal data item that depend on that item. *(iii)Mutual consistency,* which reflects the relationship between the object and its copy (replica) in different remote sites.

As illustrated earlier, real-time databases need a specific and appropriate concepts and tools in their design, which are not achieved using the ordinary methods. Although the relational model is useful for many applications, we believe that it is not as well-suited for applications that require

H. Abdel hameed is assistant lecturer with Faculty of Information Technology – Misr University for Science and Technology – Al-Motamayez District 6th of October City - Egypt. (e-mail: Hala_hameed@hotmail.com).

H. M. El-Bakry is assistant professor with Dept. of Information Systems - Faculty of Computer Science and Information Systems – Mansoura University – Egypt. (phone: +2-050-2349340, fax: +2-050-2221442, e-mail: helbakry20@yahoo.com).

T. Soltan is professor with Faculty of Computer Science and Information Systems – Helwan University, Helwan – Egypt. (e-mail: torkyibrahimsultan@hotmail.com).

complex data, complex relationships among data, and support for timing constraints. UML is a good option, but it couldn't be used in its standard form. Recently, an UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) has been standardized by the OMG [15] [19]. In this paper we try to give a model for designing a replicated DRTDB considering both the consistency of data and scalability requirements thought maintaining temporal aspects of data and timing constraints of transactions.

II. Related Work

Several UML approaches were already proposed to take into account the real-time system requirements, such as UML-RT. The UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) has been adopted by the OMG in June 2007. This profile provides support for specification, design, and verification/validation stages and is intended to replace the existing UML Profile for Schedulability, Performance and Time. However, UML constructs used by these approaches do not support real-time database requirements. Real-time databases have all requirements of traditional databases, such as the management of accesses to structured, shared and permanent data, but they also require management of time-constrained data and time constrained transactions [2]. In the literature, there is a little work for designing models for DRTDBS; RTSORAC (Real-Time Semantic Objects Relationships And Constraints) is a real-time object-oriented database model base on RT-Object package. The RTSORAC model is based upon an earlier model called SORAC [13]. The RTSORAC data model combines features of the object-oriented [18] and semantic data models [6]. As a correctness criterion, RTSORAC object is defined in the context of Epsilon Serializability (on transactions) and does not support the notion of QoD introduced in [12]. The QoD concept allows a robust and controlled behavior of real-time databases during transient overloads. Another framework in [20], has been developed using UML-RTDB, this profile consider factors such as sensor data and derived data. Unlike RT-Object package, UML-RTDB the profile of this framework supports the Quality of Data (QoD) concept by introducing the notion of maximum data error (MDE) introduced in [12]

This paper is divided in to two parts, in the first part of this paper the database model proposed for small to medium scale database and it could be distinguished from the previous two models by considering replicated objects in the database model and maintaining its consistency. In the second part we considers segmenting the whole database on node allocation, with the intent to lower the degree of replication, enables segments to have individual degrees of replication with the purpose of avoiding excessive resource usage, in particular excessive bandwidth usage when replicating updates which all together contribute in solving the scalability problem for DRTDBS.

III. A Framework to Model Real-time Replicated DataBase (FM-RRDB) model

Real-time database models the relationships between its objects and the external environment from time point of view. We study a distributed real-time database system which consists of a group of main memory real-time databases connected by high-speed networks. We assume that a reliable real-time communication is maintained, i.e., any messages sent over the network is eventually delivered and have predictable transmission time. According to the distributed nature of the database, the objective is to give the clients the illusion of service that is provided by one server and the clients have no knowledge a bout the data existence behind.

Traditional RDBMS are based on the assumption that data resides primarily on disk. In a dynamic runtime environment, data might be on disk or cached in main-memory at any given moment. Because disk input/output (I/O) is far more expensive than memory access, main memory databases have been used because of the high performance of memory accesses and the decreasing main memory cost [2]. FM-RRDB model consists of two components, Data Object model and Transaction Model.

Data Object Model

We can define two different types of data items, *temporal and non-temporal data*. Non temporal data doesn't change with time, thus they don't have validity interval and they do not need to be updated by periodic system updates, this type includes *static data*, such as locations and lookup data, and *application Data*, that used by different user transactions. On the other hand, *temporal Data* change with time and have validity interval and are updated periodically. It includes *sensor Data*, which is collected from physical world. For example, in a submarine control system, they could be ship maneuvering data (such as position, speed and depth). *Replicated Data*; items hosted by a specific site at which it is not originated, as it will be illustrated next. Although, replicated data has the same specifications of the sensor data, it differs in the methods defined for that type which is invoked by the transactions. And *derived Data*, the sensed data is processed further to derive new data called Derived Data that depends on past sensor data, for example the temperature and pressure information pertaining to a reaction may be used to derive the rate at which the reaction appears to be progressing which is in turn could be used to derive a new data.

In our distributed real-time database model, a database $\mathcal{D}$ is distributed over $\mathcal{N}$ nodes each database di (where $0 < i \leq \mathcal{N}$) resident in a node N called a site. Each site hosts a set of temporal data objects and non-temporal objects. The site is called the primary site for those data objects. Each site also maintains a set of replicas of temporal data objects hosted by other nodes. Fresh values of temporal data objects are periodically submitted from sensors to their primary sites and propagated to the replicas. For a specific data item, the data copy at the primary site is called primary copy and the copies

that are replicated are called replicated copies or replicas. The database size, $\mathcal{D}$, is the total sizes of the databases at all sites defined as follows:

$$\mathcal{D} = \sum_{i=1}^{N} di \quad (1)$$

As in ACCORD/UML approach [17] and [20], the data object encapsulates time-constrained data, time-constrained methods and concurrency control mechanisms. Each real-time object is made of three components: (i) a set of real-time attributes, (ii) a set of real-time methods (iii) a local controller. But unlike those models which encapsulate also the inter objects interactions and the exchanging information through message passing they use a component called mailbox used to store messages received by the RTO (Real-Time Object). A mailbox is attached to an object and is generally for a given object (instance). It is used only to store messages received by the object and waiting to be processed. Our model mange this interaction using transaction model as it will be illustrated next.

The abstracted data object class consists of (N, A, M, CC) where N is the name of the class, A is a set of attributes, and M is a set of interface methods and CC is the local controller associated with each object. The data object O is a logical instance of the object class. As it was stated, our model maintains the replicated data and the consistency of that data items, an object replica r is a physical representation of a logical object O. A replica $r \in \mathcal{R}$ of logical object o is one node's local view of the state of o. Each node contains at most one replica of a particular logical object. Ideally, all replicas of an object o agree on a common, consistent state of object o (i.e., their states are identical), and this is the responsibility of the replication model used which is not considered here. Note that the implementation of a non temporal data object is not considered in this paper. Notes that, the replicated set $\mathcal{R}$ for each node N, is dynamically changing, as the replica could be created and deleted continuously, thus the local database at each node consists of a set of its data object and a set of remote replicas hosted by other nodes. We will distinguish between two types of attributes for sensor or derived data objects:

1- Attributes for sensor and replicated data object

Any object has the following specifications; for each $a \in \mathcal{A}$ a= (Id, $PsiteId$, $value$, TS, VI, BUF)

Id: is the unique identifier for the object on his primary site.

$PsiteId$: the site id where the object was originates, this attribute give an indication of whether the object is a primary data object or it is a replica e.g., if $PsiteId$ = local site, this object is a primary object originated at this site, if not it is a replica for a remote data object. $Value$: is used to store the final attribute value captured by the related last update method. This field is used by the system to determine logical integrity constraints of the attribute value. TS: is used to store the last time at which the attribute's value was updated [7]. Time stamp is necessary to determine temporal consistency of the object. For example, if we store an attribute for storing the temperature, which a sensor regularly provides readings, this update is reported every 5 seconds. Thus the temperature object is considered temporally inconsistent if the update does not occur within that time frame. There are many ways to define timestamps. In our model, the timestamp is the time when the value is produced. If the value is produced by a sensor device, then timestamp is the time when the value is read by the sensor. If the value is produced by a transaction, then timestamp is the time when the transaction completes. This field is used by the system to determine whether or not timing constraints have been violated. VI: denotes object's absolute validity interval i.e., the length of the time interval following timestamp during which the object is considered to have *absolute validity*.

As mentioned before, we need to maintain consistency between the actual state of the environment and the state as reflected by the contents of the database, this leads to the notion of temporal consistency. Temporal consistency has two components:

Absolute consistency: it reflects the state of the environment and its image in the database.

Relative consistency: it reflects the consistency among the data used to derive other data; this arises from the need to produce the sources of derived data close to each other. So, it could be stated that the value of the temporal object is logically consistent if it satisfies all integrity constraints and it is temporally consistent if it satisfies the Absolute consistency (current time - timestamp) > VI. Another type of consistency is related to the derived data objects called relative consistency, for example if we considered two objects o1 and o2 which have to timestamps TS_1 and TS_2 respectively, O_1 and O_2 satisfied the relative consistency called relative valid interval RVI if: $|TS_1 - TS_2| \leq RVI$

Consider the following example Suppose temperature rvi =5 and pressure rvi =10 R {temperature , pressure } and R rvi If current time=100, then (a) temperature = (347,5,95) and pressure = (50,10,97) are temporally consistent but (b) temperature = (347,5,95) and pressure = (50,10,92) are not. In (b) even though the absolute consistency requirements are met, relative consistency is violated Whereas a given rvi can be realized by sampling the corresponding real world parameter often enough realizing an rvi may not be that straight forward This is because achieving a given rvi implies that the data items that belong to a relative consistency set have to be observed at times close to each other. BUF: is the Basic Update Frequency, for each temporal data object it is updated periodically at a given update frequency received from its primary site (sensor), while its replicas are updated at a different general update frequency (GUF).

2- object attributes for derived data object

The object attributes have the following specifications; for each $a \in \mathcal{A}$, a = (Id, $PsiteId$, $value$, RCS, RVI, BUF). Id: is the unique identifier for the object on his primary site. $PsiteId$: as for sensor object the site id where the object was originates. $Value$:, the current value of the object and is used by the system to determine logical integrity constraints of the attribute value. RCS is the Relative Consistency Set, which

includes a set of sensor data objects from which the object is derived. $\mathcal{RVI}$ denotes object's relative consistency as described above. $\mathcal{BUF}$, if this item is originated at another site, so it could be replicated periodically at a given update frequency received from its primary site.

3- Real-time methods

As in [20], we classify the real-time object methods into three classes: *periodic* methods, *sporadic* methods, and *aperiodic* methods.

Periodic methods:

These methods include the temporal consistency for temporal data objects thought update of sensor data or update remote replica for each data object, thus we can distinguishes between two actions related to the replication strategy: *push* action in which a replica updates are sent periodically at a given update frequency, and *pull* action when the updates are request from the remote sites. We define Update method using $\mathcal{BUF}$ to update sensor data object, and Propagate method to update its replica on a remote sites using $\mathcal{GUF}$.

Sporadic methods:

These methods associated with the derived data that calculated from sensor data [3]. We can define a Calculate method to calculate its value from sensor data items specified on its $\mathcal{RCS}$. Another method Check could be defined to maintain relative consistency of that object by checking $\mathcal{RVI}$.

Aperiodic methods:

They include the remainder of methods that allow reading the derived and sensor data objects as user transactions typically arrive aperiodically. They do not write any temporal data, but they can read/write non temporal data and only read temporal data [3]. Another method Check validity could be defined to check the validity of each item using timestamp and its valid interval as described above.

4- Concurrency Controller CC

To improve performance and maintain temporal constrains of both temporal data objects and transactions, CC associated for each object allows the concurrent execution of transactions thought allowing multiple methods to run concurrently, by using any existing concurrency protocol supporting replicating environment. Our framework depends on locking methods, either local lock or global lock. We can distinguish between read and write lock by defining a certain value for each data object indicating its status and its availability to be locked by incoming transaction or not.

Transaction Model

Transaction is a sequence of operations on the database that maintain the ACID property, transaction is an Atomic unit that executed all or nothing, and Consistent means it takes the database for a consistent state to another consistent state. I, is for Isolation, that the transaction execution is completely independent from the execution of another one. The last D means durability that when the transaction commits, its result is written in permanent storage that can not roll back [5].

As mentioned earlier, data items are classified to temporal and non temporal data items according to their timing nature and requirements. Accordingly, transactions in a replicated real-time database could be classified as replication transactions, which concern with system update transactions that include both sensor update transaction at the primary site and the replica update transactions on the other nodes where replicas for specific temporal data object resident. Application transactions on the other hand includes all user requests for both temporal and non temporal data items, here, we consider only the transactions that manipulate the temporal data items (either update or query). Transactions can also be classified into local transactions, if all its operations are performed locally without a need for any remote access to any other site. And global transactions if at least one of its operations executes on a remote site. It is the job of the transaction manager to map the replication and application transactions to both local and global transaction. Note that our model doesn't consider or maintain replication guarantee for non temporal data objects. According to the previous classifications a general transaction can be donated as follow: $T_{type} = (\mathcal{T}_{id}, \mathcal{L}site\ Id, \mathcal{R}site\ id, \mathcal{DL}, e)$.Where transaction type specifies its type, replication or application transaction, $\mathcal{L}site\ Id$ is the id of the site where the transaction was made, $\mathcal{R}site\ id$ is the site to which the transaction is sent. $\mathcal{DL}$ and e is the dead line time and execution time for the transaction respectively.

Replication transactions:

$\mathcal{T}_{update} = (\mathcal{T}_{id}, \mathcal{L}site\ Id, \mathcal{R}site\ id, \mathcal{WS}, \mathcal{PUF}, \mathcal{DL}, e)$

Where $_{update}$ type is a sensor data update transaction, $\mathcal{WS}$, is the write set of the transaction. $\mathcal{GUF}$ is the general update frequency, and $\mathcal{L}site\ Id = \mathcal{R}site\ id, \mathcal{PUF}$.

$\mathcal{T}_{propagation} = (\mathcal{T}_{id}, \mathcal{L}site\ Id, \mathcal{R}site\ id, \mathcal{WS}, \mathcal{GUF}, \mathcal{DL}, e)$

Type $_{propagation}$ is for updating replicas at remote sites.

Application Transactions:

$\mathcal{T} = (\mathcal{T}_{id}, \mathcal{L}site\ Id, \mathcal{R}site\ id, \mathcal{RS}, \mathcal{WS}, \mathcal{DL}, e)$, It includes all transactions which all are performed either locally or remotely .$\mathcal{RS}$, $\mathcal{WS}$, is the Read set and the write set of the transaction respectively.

IV. Real-time Replicated DataBase model With Segmentation (FM-RRDB WS)

As previously illustrated, replication is useful in improving the availability of data. The most extreme case is replication of the whole database at every site in the distributed system, thus creating a full replicated database. This approach obviously improves availability because the system can continue work as long as at least one site is up and it also improves performance by enabling both local access and retrieval of data. On the other hand it leads to slow down system update, that each update requires update all copies at all the other remote sites. This problem is became worsen in the real-time systems because of their timing requirements and the expensive cost to replicate the sensor data especially in case of heavy work loads. In real-time database systems, the workload of temporal

data update can be very high (e.g., over 500 up- dates/sec [11). Thus, the cost of maintaining replicas for all data, especially sensor data, is too high which causes a scalability problem, with respect to bandwidth usage for replication of updates, storage usage for replicas, and processing usage for propagating, integrating and conflict resolving of detached updates. The usage of each of these resources grows as O (n2), where n is the number of nodes in the system, assuming that the number of updates grows as O (n). The concept of "virtual full replication" was introduced in [10] to address scalability issues. In the paper, virtual full replication is provided by segmenting the database and allowing different segments to have different degrees of replication. Segment is a group of data objects that share properties, capturing some aspects of the application semantics, and is allocated to a specified subset of the nodes (possibly temporarily inconsistent with each other). Each segment is considered as a partition of the database as units of allocation of replicas, which can be individually replicated based on specified replication requirements from all the clients at a certain database node. If the specification indicates that a segment will never be used by any clients on a node, it does not need to be replicated to that node. In [10] a segmentation algorithm has been proposed that uses a table to collect all properties of interest for a set of data objects, the most important properties of data objects, or group of objects, is the set of nodes where they are accessed by transactions. And this is the addressed property needed by our model. The algorithm has been omitted for space purposes.

The following tables illustrate the way by which a segment regarding accessing the data object is performing. As in table 1(a), a database of 6 objects replicated to 5 nodes. Rows in the table could represent groups of data objects, such as object classes or user-defined object clusters that share the same properties. By assigning a binary value to each column, each row can be interpreted as a binary number. This object key uniquely identifies the nodes where the object is accessed. By sorting the table on the object key, we get the table in table.1 (b). Passing through this table once, we can collect rows with same key value into unique segments.

A data object can be assigned to only one segment, by this rule we can treat each segment independently and decide a multiple replication degree. And we can use transaction grouping, by witch the propagation transaction are grouped to be by replicating the whole segment rather than replicate an individual object to a certain site, thus contributing in solving the scalability problem. To implement the segmentation concept in our framework, each data object will have segment specifications as follows: O = (N, A, M, CC, *Sid*), where *Sid* is a unique identifier for the segment to which the object belongs. Anyway, the implementation of this part is out of scope of this paper, and is considered as a future work.

V. UML Profile

Complex systems, such as real-time database require the modeling of dynamic properties for data and information. Therefore, the development of methods to design real-time databases with support for both static and dynamic modeling is an important issue. In the literature, there are a few works for real-time database modeling, taking into account the replication and scalability issues. UML presents a number of favorable characteristics for modeling complex real-time systems, it also used for modeling object-oriented database systems. In this paper we use an UML profile, entitled UML-Magicdraw 15.5, which is a specialized variant of the UML Profile for MARTE for real-time database applications.

In our work, we tried to supply the designers of real-time databases, UML extensions to support real-time database requirements. An UML extension is specified in the UML metamodel by a stereotype. This latter allows designers to extend the vocabulary of UML in order to create new model elements, derived from existing ones, but that have specific properties that are suitable for a particular problem domain. In [20] UML-RTDB stereotypes extend metamodel classes with specific sensor and derived attributes, specific periodic and sporadic operations and a specific real-time class that allow the design of class diagrams for real-time databases. While, in our work, UML-RTDB stereotypes have been extended for both data object and transaction object for specific sensor, replicated, and derived attributes as illustrated in figuers 1 and 2. So, temporal real-time object is either sensor, derived, or replica, so we define three stereotypes, *<<Sensor>>*, *<<Derived>>*, and *<<Replica>>* to declare respectively sensor, derived and replicated objects in the class diagrams.

VI. Conclusion and Future Work

A general framework to model a replicated real-time database has been designed for small and medium scale distributed real-time database systems. Both data and transactions timing constrains are maintained. The presented framework has been extended to model large scale database systems. This has been done by presenting a general outline that considers improving the scalability by using an existing static segmentation algorithm applied on the whole database on node allocation. It is desirable to implement this framework on a real distributed real-time database system and evaluate it with real transactions requirements. We are currently developing a replication control algorithm. FW-RRDB will be used as a framework for our real-time database. We are also looking into implementing the segmentation part as a contribution in solving the scalability problem of distributed real-time database systems.

References

[1] A. Bestavros, K.-J. Lin, and S. Son. "Real-Time Database System: Issues and Applications", chapter Advances in Real-Time DataBase Systems Research, in *The Springer International Series in Engineering and Computer Science, Vol. 396*, Kluwer Academic Publishers, 1997, pages 1–14.

[2] J.Baulier, P.Bohannon, S.Gogate, C.Gupta, S.Haldar, "DataBlitz storage manager: Main memory database performance for critical applications", in *Proc. ACM SIGMOD international conference on Management of data*, vol 28, no.2, pp.519–520, June 1999

[3] B. Selic. "Using the object paradigm for distributed real-time systems". *In Proc of First IEEE International Symposium on Object oriented Real-time distributed Computing (ISORC'98)*, Kyoto, Japan, April 1998, pp. 478–480.

[4] Y-W.Chen, and L.Gruenwald, "Effects of Deadline Propagation on Scheduling Nested Transactions in Distributed Real - Time Database

Systems," *Journal of Information Systems*, Vol. 21, No. 1, pp. 103 - 124, 1996.
[5] J.Gray and A.Reuter, "*Transaction Processing: Concepts and Technique*," Morgan Kaufman, San Mateo, CA, 1993.
[6] P.Joan, M.Fred, "Semantic data models, *ACM Computing Surveys*, vol 20, no. 3, pp. 153-189, September 1988.
[7] K.Ramamritham and C.Pu. "A Formal Characterization of Epsilon Serialisability". *IEEE Transaction Journal on Knowledge and Data Engineering,* vol 7, no.6, pp.:997–1007, December1995.
[8] H.Kopetz, and P.Verissimo, "Real time and dependability concepts", in ACM Press Frontier Series, ed., 'Distributed Systems' 2nd Ed, New York, NY, USA, Addison-Wesley, 1994, ch. 16.
[9] Lee Juhnyoung, "Concurrency Control Algorithms for Real-time Database Systems", PhD Thesis, Department of Computer Science, University of Virginia, 1994.
[10] G. Mathiason and S.F. Andler "Virtual Full Replication: Achieving Scalability in Distributed Real-Time Main-Memory Systems", *in Proc. Euromicro Conf. on Real-Time System,* Porto Portugal, July 2003.
[11] M. Cochinwala and J. Bradley. "A multi database system for tracking and retrieval of financial data". *In Proc of 20th International Conf. on Very Large Data Bases*, Morgan Kaufmann Publishers Inc. San Francisco,1994, pp. 714 – 721
[12] M. Amirijoo, J. Hansson, and S. H. Son. "Specification and management of QoS in real-time databases supporting imprecise computations". *IEEE Transactions on Computers,* vol. 55, no. 3, pp. 304–319, March 2006.
[13] D.Michael, P. Joan, and W.F.Victor. "Implementing relationships and constraints in an object-oriented database using monitors". *In Rules in Database Systems, Proc. 1st International Workshop on Rules in Database*, pp. 347-363, 1993.
[14] N. Idoudi, C. Duvallet, R. Bouaziz, B. Sadeg, and F. Gargouri. "Structural model of real-time databases: an illustration". *In Proc. 11th IEEE International Symposium on Object oriented Real-time distributed Computing (ISORC'2008)*, Orlando, United State, May 2008, pp. 58 – 65.
[15] OMG,"UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems", Version 1.0, OMG Document Number: formal/2009-11-02, available at URL: http://www.omg.org/spec/MARTE/1.0, updated at November 2009.
[16] OMG. "UML Profile for Schedulability, Performance and Time", v1.1, formal/2005-01- 02, January 2005, available at http://www.omg.org/cgi-bin/doc?formal/2005-01-02.
[17] S. Gerard, C. Mraidha, F. Terrier, and B. Baudry. "A UML-Based Concept for High Concurrency: The Real-Time Object", *In Proc. 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'04)*, Vienna, Austria, 2004, pp. 64-67.
[18] Z.Stanley and M.David. "*Readings in Object Oriented Database systems*". Morgan Kau_man, San Mateo, CA, 1990.
[19] S. Demathieu, F. Thomas, C. André, S. Gérard, and F. Terrier. "First experiments using the UML profile for MARTE". *In Proc. 11th IEEE International Symposium on Object oriented Real-time distributed Computing (ISORC'2008)*, Orlando, United State, May 2008, pp. 50–57.
[20] S. G.Bruno, I.Nizar , L.Nada , D.Claude , B.Rafik , and G. Faiez " A Framework to Model Real-Time Databases". International Journal of Computing & Information Sciences Vol. 7, No. 1, January 2009.

TABLE 1(a)
ACCESS TABLE

	Accesses						
Column value		1	2	4	8	16	
Objects		N1	N2	N3	N4	N5	Object Key
O1		X			X		9
O2			X			X	18
O3			X	X			6
O4				X			4
O5			X	X			6
O6		X			X		9

TABLE 1(b)
SEGMENT TABLE

Column value		1	2	4	8	16		Object Key
Objects		N1	N2	N3	N4	N5		Segment key
O1				X				4
O2			X	X				6
O3			X	X				6
O4		X			X			9
O5		X			X			9
O6			X			X		18

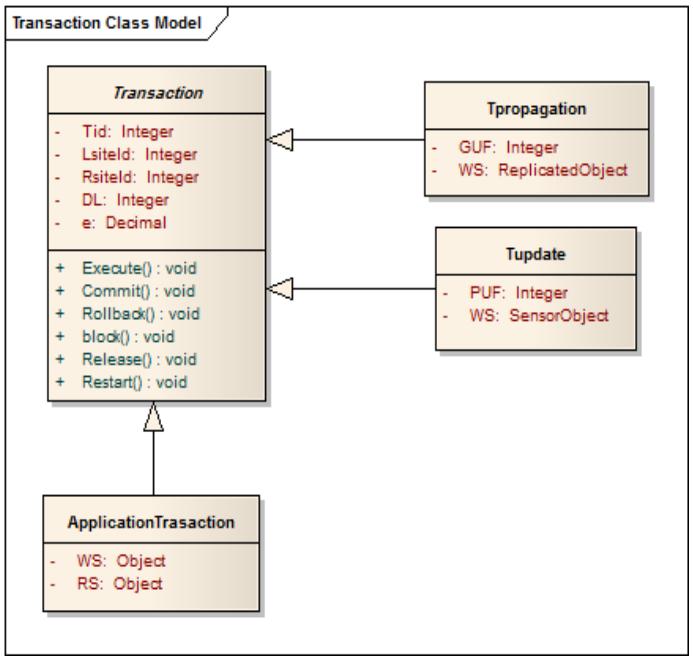

Fig.1 Object Class Model.

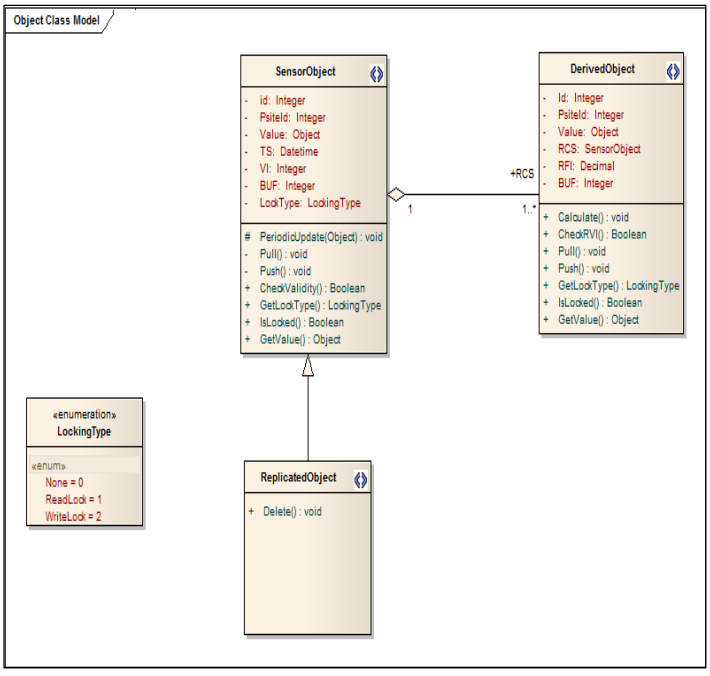

Fig. 2: Transaction Class Model.