Design, Implementation, and Evaluation of a Real-Time Object-Oriented Database System

Zied Ellouze[1,*], Nada Louati[2], Mossaad Ben Ayed[3,4,*], Rafik Bouaziz[5], and Shaya Abdullah Alshaya[6]

[1]Preparatory Engineering Institute of Sfax, CES Laboratory, Sfax University
[2,5]Higher Institute of Computer Science and Multimedia of Sfax, Miracle Laboratory, Sfax University
[3,6]Faculty of sciences and humanities sciences, Majmaah University, Saudi Arabia
[4]Computer and Embedded System laboratory, Sfax University, Tunisia

Summary
Recent works on Real-Time (RT) databases are focused on simulations. These studies present shortcomings not only in verification but also in comparison with existing core techniques used for the management of RT databases. Besides, RT database applications are challenged by the lack of an RT database system open source to perform a realistic environment. In this paper, a design based on a open-source object-oriented environment database system is proposed for the RT database application development. The proposed system is based on the RTQL (Real-Time Query Language) which is defined as a high-level RT object-oriented query language. It supports the expression of timing constraints on data and it ensures transactions upon the data. Then, a QoS management approach is performed to reduce workloads and to detect overload. The proposed attempt is supported by the use of the feedback control RT scheduling theory. Our evaluation shows that our approach not only achieves the desired timeliness of transactions but also maintains high data freshness compared with other related approaches.

Key words:
Real-Time Database, Object-Oriented Database, Feedback Control, Data Model, Real-Time Query Language

1. Introduction

RT applications as traffic control, stock trading, and agile manufacturing require RT databases to execute transactions within their deadlines based on many temporal data that represent the current status of the real-world [1][2]. For example, in a stock trading application, a stock's price should be saved in the RT database, but only considered valid for 30 seconds from when the value was updated. The stock's price value will be considered temporally inconsistent if the update does not occur within 30 seconds. Also, there is some information required from the RT database that will be needed to check if a profit will be generated by a stock trade. This information has to be retrieved within a given deadline to decide to start a stock trade or not. Thus, RT databases aim to support applications in which the timeliness of processing and freshness of data are important.

An RT database system ensures all features of conventional database systems. At the same time, it must enforce timing requests imposed by applications. As a conventional database system, an RT database provides efficient storage, operates as a repository of data, and performs manipulation and retrieval of data. However, RT system improves features associated with the conventional database by adding timing requirements to ensure some degree of confidence [1]. In conventional database systems, performance is primarily measured by the average response time, while in RT databases the primary performance metric is miss ratio. It is defined by user transactions that fail their deadlines. Several off-the-shelf RT databases are available such as EagleSpeed [3], Polyhedra [4], TimesTen [5], and ExtremeDB [6]. However, these systems use traditional technology of RT databases and they do not meet the needs of timing and data freshness. Furthermore, they show low performance in RT data-intensive applications and they are not open to the public [7].

Although significant research covered many aspects of RT database systems design, there has been little work in transaction specification languages or query languages. A query language ensures the definition, manipulation, and control of data in a database system. For a RT database, the query language manages timed transactions to impose fresh data and to reflect the real-world status. The most previous research works in RT object-oriented query languages do not consider timing constraints on data and on transactions upon the data. An attempt named RTSQL based on SQL2 [8], standard specification tries to support timing for RT databased querying. It has relied on the relational data model which provides poor support for complex database applications. On the other hand, the Object Data Management Group (ODMG) has defined a standard for object-oriented databases, including an Object Model, an Object Definition Language (ODL), an Object Query Language (OQL), and an Object Manipulation Language (OML). These proposals do not consider RT requirements. Thus, there is a need to define a RT query language supporting RT database requirements.

Manuscript received October 5, 2019
Manuscript revised October 20, 2019

Nowadays, the need for RT data-intensive applications has increased due to the request of fresh data which gives the current status instantly. They also have to execute transactions within their deadlines. However, it seems to be difficult to have a compromise between processing timed transactions and using fresh data. This timing challenge competes for system resources to achieve transaction timeliness and data freshness requirements. Furthermore, due to the dynamic data/resource contention, database workloads vary. To overcome this issue, the feedback control scheduling method has been recently applied to systematically manage RT database performance and trade data freshness for the timeliness of transactions in the presence of unpredictable workload [9]. Authors in [7] prove that the most existing works on feedback control of RT database performance are based on simulations that have limitations in comparing and verifying existing core techniques for RT database performance management.

To address these problems, we propose in this paper a RT object-oriented environment for RT database application development, denoted RT2O, on top of EyeDB [10]. RT2O has been performed for RT data-intensive applications to support timed transactions. Contributions of this paper are as follows:

- Object-oriented design: Object-oriented data models were developed to deal with complex objects increasingly in use in today's computer applications. Because RT databases require advanced modeling concepts, an object-oriented data model represents an excellent candidate for their realization without requiring fundamental extensions to the basic data model.
- RT object-oriented query language: we propose a Real-Time Query Language (RTQL), on top of ODMG query language [11]. The objective of RTQL is to present an extension to the ODL, OQL, and OML languages to accommodate RT object-oriented databases. To identify the necessary constructs that should be added to the languages, we propose a RT Object Model which is an extension of the ODMG Object Model. The actual language constructs are developed based on this model.
- Feedback control architecture: In the RT2O, the metrics for QoS are the freshness of data and the timeliness of transactions. The RT2O based on a multi-version of data architecture achieves the wanted QoS using a feedback control technique. This feedback not only reduces data access conflicts between transactions but also decreases the deadline miss ratio.
- Implementation: Most RT database works are limited at the simulations phase [12] [13][14] [15] [16]. Other researches as [17] [7] [18] [19] attempt to evaluate RT data management techniques in real database systems. In our case, the RT2O has been implemented by extending EyeDB [10], which is an open-source object-oriented database system. The RT2O does not require a specific library tied to the operating system; it requests only on the standard RT features of POSIX [20].
- Evaluation: The RT2O has been evaluated on a real case study. The evaluation results demonstrate that the RT2O provides a proper amount of resources in a robust manner when the freshness of data and the timeliness of transactions are performed.

This paper covered the proposal as follow: the next section presents the previous studies related to the real-time databases, section three raises the proposed architecture and QoS management of the RT2O, the query language is described in section four, the implementation and the evaluation are discussed in section five, and the paper is ended by a conclusion.

2. Related Works

Most RT database studies is focused on simulations [13] [27] [12] [14]. However, recent studies attempt to introduce the prototyping of the RT databases for general purposes. This section is focused on researches that attempt to implement the RT databases.

Prichard et al. [30] implemented a prototype of a RT database architecture based on RTSORAC model. This architecture presents many shortcomings: (1) the model is too complex, (2) the architecture suffers from schedulability and timeliness aspect (3) Bounding execution time is difficult due to the supports of features such as semantic concurrency control.

DeeDS prototype [31] is an event-triggered RT database system. This attempt modeled reactive behavior based on Event-Condition-Action rules. The major limit of the DeeDS architecture is shown by the lack of time constraints of data.

An object-oriented and fault-tolerant database management system named RODAIN [32] is described to support the real-time. This architecture, based on a main-memory database, offers priority and criticality criteria. It ensures scheduling and optimistic concurrency control. Unfortunately, RODAIN architecture is designed for telecommunications applications.

The BeeHive [18] RT database is based on object-oriented databases. Authors propose to incorporate semantic information regarding RT, security, importance, QoS requirements, and fault tolerance to constitute an extension of the conventional object-oriented databases. BeeHive

system does not support the transaction deadline. It ensures transaction scheduling based on data deadline but the execution time would be computer offline for admission control. In fact, this architecture is limited to RT data services that transactions and their arrival data access patterns are known in advances [33].

StarBase [34] is a firm RT database system. The lack of information related to the transaction workload is the most limited for this architecture. This prototype runs on top of RT-Mach, a RT operating system developed at Carnegie Mellon University.

Chronos [7] based on the relational model is a soft RT database testbed. This prototype is designed for high-throughput business applications. Therefore, this attempt does not provide general-purpose RT databases.

ODEA is an architecture based on object-oriented language for RT database system [35]. However, this prototype suffers from the miss of the ODMG standard of object databases. ODEA is content with simulation and the model is not evaluated in a real case study.

Most of the systems presented are not available publicly. One of the challenges supported by our attempt is to provide an open-source architecture for RT database system.

Most of the cited researches previously use the relation model to perform their prototypes. But, the relational model presents a limitation in the case of complex RT applications. Besides, discussed previous studies do not provide some essential features related to the RT databases as concurrency control, temporal consistency, and guaranteeing logical consistency. Solutions proposed are challenged by reduced performance.

Fortunately, the proposed RT2O guarantees the QoS goals required by the application using a feedback control architecture even in the presence of dynamically changing workload. In addition, RT2O system use a very high- level RT object-oriented query/programming language RTQL.

3. RTO2 Architecture and QoS Management

In this section, RT2O database model, RT2O architecture and QoS management are discussed.

3.1 RT2O Database Model

RT2O is based on a firm RT object-oriented database model. In this model, the transaction is aborted if it missed its deadlines.

3.1.1 Data Model Choice

The term data model refers to the way in which data is structured by a DBMS (DataBase Management System). The two most popular data models are the relational model and the object model. Several research works have been directed towards using the relational model as a data model for RT databases [21]. As RT applications are becoming increasingly sophisticated in their data needs, the RT database community migrated towards object-oriented technology to deal with complex database applications [22]. The proposed approaches offer solutions to manage concurrency and schedulability issues of RT databases. However, they do not provide on the one hand concepts to support quantitative features such as deadline and period and, on the other hand, qualitative features that are related to behavior and communication.

The richness of the object-oriented paradigm in terms of concepts offers an interesting common modeling basis to adequately specify many design features of RT databases. In [11], the ODMG has defined an Object Model that specifies the constructs which are supported by an object-oriented DBMS. Among other things, this Object Model specifies the characteristics of objects, how objects can be named and identified, and how objects can be related to each other. In the next subsection, we present a RT Object Model for RT databases which is an extension of the ODMG Object Model to support the expression of the time-constrained attribute, time-constrained operations, and time-constrained classes.

3.1.2 RT2O Object Model

This model is based on our object-oriented data model introduced in [22] and it extends the ODMG Object Model to suit RT database features. As depicted in figure 1, four major enhancements to the conventional ODMG Object Model are proposed, namely, RT attributes, RT methods, RT classes, and RT objects.

RT Attributes: Attributes may be classified into two types: non-RT and RT. A non-RT attribute does not become outdated due to the passage of time. Whereas, the current status of the real-world is ensured by the continuous change of the RT attribute. The validity duration represents an essential criterion associated with the RT attribute. Validity duration performs the time during which the RT attribute's value is considered valid. Moreover, the RT attribute is composed of timestamp which defines the time of the last updated.

a) RT attributes are classified into two categories: sensor attributes and derived attributes. Sensor attributes are issued from sensors, whereas derived attributes, they are calculated from sensor attributes. By updating methods, RT attributes is updated periodically. These methods are activated at fixed intervals of time and when the data item value modified.

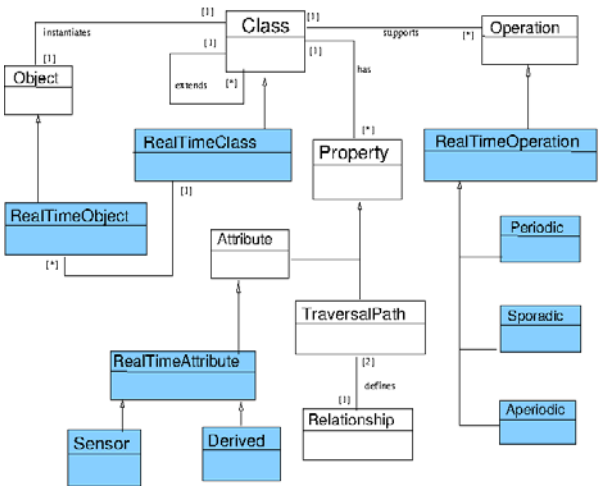

Fig. 1 A RT Object Model

b) RT Operations: The methods defined by objects are invoked to ensure access to transactions. In our RT Object Model, each operation execution is considered as a transaction. Each transaction is constituted by at least one sub-transactions. In our RT Object Model, transactions are classed into types: user transactions and update transactions. User requests are defined by user transactions. They arrive non-periodically and they ensure a read or a write in the case of a non-RT data, and only read in the case of a RT data. Update transactions are responsible for updating the values of RT data in order to reflect the state of the real-world. They are executed periodically to update sensor data, or sporadically to update derived data.

c) RT Classes and RT Objects: In order to deal with RT data and RT transactions, we introduce the RT class concept. This latter specifies that instances of a class will encapsulate RT attributes and RT methods (i.e., RT transactions). Both RT classes and instances of these classes are referred to as RT objects. RT objects are the RT database entities. They represent dynamic entities of time-critical systems in the real-world. Each RT object has some internal state which is protected by the object abstraction.

To close this section, we provide the IDL (Interface Definition Language) code, see figure 2, that we have added to the ODMG Object Model in order to support our RT Object Model.

```
interface RealTimeClass : Class {
};
interface RealTimeAttribute : Attribute {
attribute integer value;
attribute Timestamp timestamp;
attribute integer validityDuration;
void setValue(integer v);
integer getValue ();
void setTimestamp (in Timestamp t);
Timestamp getTimestamp();
void setValidityDuration(in integer v);
   int getValidityDuration();
};
interface Sensor : RealTimeAttribute
{ attribute integer maximumDataError;
integer getMaximumDataError ();
void setMaximumDataError (in integer m)
void periodicUpdate (in integer v)
};
interface Derived: RealTimeAttribute
{void sporadicUpdate(in integer v)};
interface RealTimeOperation: Operation {
attribute Timestamp deadline;};
interface Periodic: RealTimeOperation {
attribute integer period;};
interface Sporadic: RealTimeOperation {};
interface Aperiodic: RealTimeOperation {};
```

Fig. 2 Interface definition language code

3.2 RT2O Architecture

Figure 3 shows the overall architecture of RT2O that consists of the admission controller, QoS Manager, Scheduler, deadline controller, freshness controller, transaction handler, concurrency controller, object manager, and data manager. RT2O users can configure to turn on or off these components individually for performance evaluation purposes. The functions of these components are as follows:

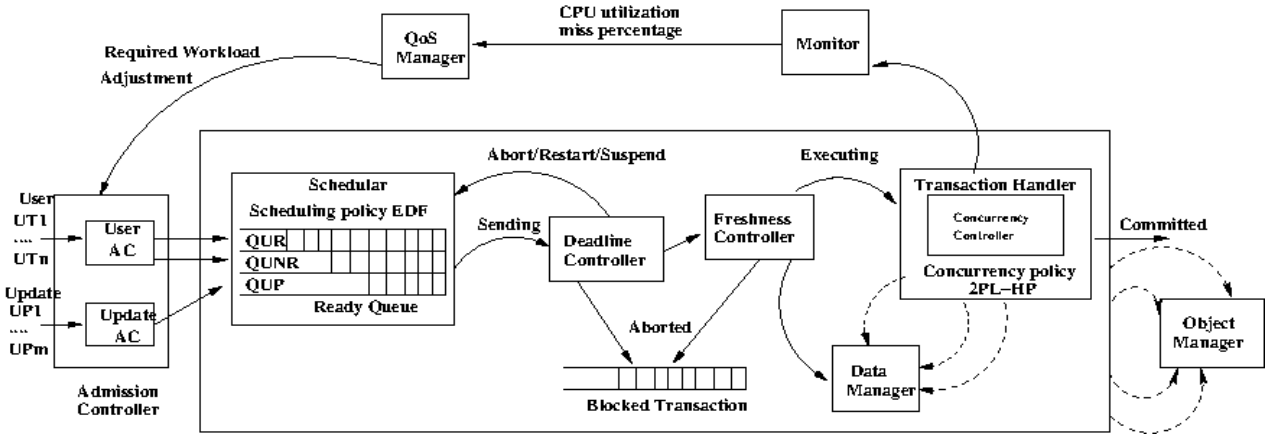

Fig. 3 RT2O Architecture

- Admission Controller: The RT2O system has a pool of schedulable resources. When no schedulable resources are available, the system may be overloaded. The admission controller is used to avoid system overload. In fact, when receiving a new transaction, the admission controller checks if it can be handled. Otherwise, the transaction is rejected. The admission controller performs the necessary tests to determine if the RT2O system has sufficient resources to support the requirements of an incoming transaction (user or update transactions) without compromising the service guarantees made to currently active transactions. It admits the incoming transactions based on system response times of the completed transactions, i.e., committed transaction, and on CPU and IO utilization of the admitted transactions.
- Scheduler: The scheduler manages the priority of the transactions' threads in the transaction thread pool, i.e., the ready queue. The scheduler prioritizes the transaction threads in the ready queue periodically according to their priority. The priority of a transaction depends on both its deadline and its type (user or update transaction). Note that since the updated data are needed by user transactions, update transactions receive a higher priority than user transactions. Figure 2 shows three separate queues in the ready queue, namely, QUP, QUR, and QUNR. QUP is used to schedule update transactions and it receives the highest priority, similar to [6]. RT user transactions are handled in QUR. Non-RT transactions have the lowest priority. They are scheduled in QUNR and they are dispatched only if QUP and QUR are empty. Transactions in each queue are scheduled in EDF (Earliest Deadline First) manner.
- Monitor: The monitor calculates various statistics such as RT performance and global resource utilization and updates the RT2O state table with this information. Moreover, it detects overload by periodically computing the CPU utilization and the miss percentage and sends them to the QoS Manager.
- QoS Manager: The QoS manager is used to compute the required workload adjustment to be used for admission control. In fact, it compares the performance reference with the computed variable sent by the monitor to get the current performance error. Based on the result, the QoS manager changes the total estimated requested load by adapting the quality of data, i.e., adjusting the Maximum Data Error (MDE). MDE is the upper bound of the deviation between the current attribute value in the RT database and the reported one [23].
- Transaction Handler: The transactions handler consists of a concurrency controller component. This component aims to maintain database consistency and to control the interactions between concurrent transactions. The RT transaction concurrency control mechanism performed by our RT2O system is 2PL-HP (Two Phase Locking-High Priority).
- Freshness Controller: The freshness controller checks the freshness of acceded data just before a transaction is sent to the transaction handler. This component is used to provide better QoS in RT databases where several transactions access the same data items. The data retrieved by committed transactions are usually fresh at commit time. In the case of the accessed data is fresh, the transaction is executed and sent to the transaction handler. Otherwise, if the acceded data items are currently stale or will be before the deadline of

the transaction, the freshness controller blocks the transaction, i.e., sent to the blocked queue. The blocked transactions will be moved from the blocked queue to the ready queue once the corresponding update transactions that are responsible for updating the values of acceded data items have committed. To measure the freshness of a data item Di, the freshness controller uses the notion of Absolute Validity Interval (AVI). As we have mentioned in our data model, each RT data is characterized by a timestamp that indicates the latest observation of the data item in the real-world. Di is considered fresh if the next equation is verified

$$\text{CurrentTime} - \text{Timestamp}(D_i) \leq \text{AV I}(D_i) \quad (1)$$

- Data Manager: The data manager is used to guarantee the data freshness even in the presence of conflicts. Most conflicts come from incompatible access patterns, for example, when an update transaction wants to modify the value of a data item that is accessed by a user transaction. One of these two transactions must be aborted and restarted according to the used concurrency control policy (in our work the 2PL-HP policy). Then the risk that transactions miss their deadline increases. To alleviate this risk, the data manager uses a multi-version data technique, similar to [24]. This technique consists of creating of data versions when a conflict occurs between transactions, i.e., read-write conflict. The maximum number of versions is fixed in advance by the RT2O administrator according to QoS requirement level.
- Deadline Controller: The deadline controller is used to control transaction validity. It uses two variables: current time and transaction deadline. The transaction is aborted when the current time is greater than the transaction deadline. Otherwise, the transaction is transferred to the freshness controller when the verification step is succeeded.
- Object Manager: The object model of RT2O database model described in Section 2.1.2 would be supported by adding an object manager module that provides support for retrieving, updating, removing, and adding objects in the RT2O database. All the persistent objects are created and stored in permanent storage using a storage manager module. In the beginning, the RT2O architecture creates a shared main memory segment and instantiates an object table. The organization of this main memory is provided by the objects module. Object manager aims to ensure fast and predictable access to the object stored in the main memory.

3.3 QoS Management

This section describes the admission control, overload detection, and the MDE-based update schemes provided by RT2O.

RT2O follows the client-server model. Each RT2O architecture could be considered as a client or as a server. One or many clients could be connected to one or many servers. The RT2O system could be loaded when many user transactions from multiple client threads are executed concurrently in addition to frequent RT data updates. As a result, many transactions can be aborted and restarted or blocked due to data contention. Moreover, computational resources such as memory space and CPU cycles can be exhausted.

Kang et al. [7] present a study that offers to detect the overload and determine the required workload adjustment. Authors perform the difference between the actual response time and the desired delay bound to provide the degree of timing constraint violation. In our work, the degree of timing constraint violation is computed according to the average service delay measured in a sampling period (denoted t_a) and the desired delay bound (denoted t_d). In this paper, t_a is measured for each 40s period to ensure a sufficient number of transactions committed within a measurement period. The RT2O is considered overloaded if $t_a > t_d$ and the degree of overload at the k^{th} measurement period is:

$$\delta(k) = (t_a(k) - t_d) / t_d \quad (2)$$

The degree of overload is computed to determine the workload adjustment requirement.

Actually, workloads change over time. Consequently, the response time varies from a period to another. The RTO2 performance changes when the admission control and MDE-based update schemes are performed according to instantaneous δ values. In order to deal with this issue, the measurement periods are computed using an exponential average of δ over various. The value of $\delta_s(k)$ the k^{th} measurement period is:

$$\delta_s(k) = \alpha.\delta(k) + (1-\alpha).\delta_s(k-1) \quad (3)$$

where $0 \leq \alpha \leq 1$ is a disciplined parameter

To reduce the workload when $\delta_s(k) > 0$, the RT2O performs the admission control to incoming transactions. For instance, if $\delta_s(k) = 0.1$, the admission controller attempts to support the desired delay by reducing the number of concurrent transactions by 10%.

The notion of MDE dealt in [23] is used to reduce update workloads under overload. The QoS manager rejects an update transaction writing to a data item D_i having an error less than or equal to the MDE allowed. However, when the data error of Di is greater than MDE, the update transaction is executed.

4. A Real-Time Object-Oriented Query Language

The ODMG [11] standard is composed of ODL and OQL. The ODL defines a data manipulation language and object classes. The OQL defines queries. Therefore, the ODMG standard ensures a data description language. In this section, we present enhancements to ODL and OQL to include RT support. Our data description language, RT-ODL, is ODL improved by a number of class definitions and syntactic constructs by adding a time dimension to types. The aggregation of the proposed RT Object Model and the OQL language provide the specification of these classes. Our query language, RT-OQL, presents an improvement of the OQL.

To highlight the importance of our propositions, we choose the Online Stock Trading (OST) as the motivating application to illustrate the use of RT-ODL and RT-OQL. OST is a data-intensive RT application which monitors the prices of stocks or other financial instruments (e.g., quotes and trades) and looks for trading opportunities. It requires a RT database to process timed-constrained transactions and maintain the freshness of stock prices. For more details, refer to [25].

The RT database for the OST case study contains the following classes: Stocks, Quotes, QuoteHistory, Portfolios, Accounts, Currencies, and Personal. Table 1 defines the list of attributes of each class.

Table 1: The list of attributes of the OST

Class	Attributes
Stocks	stockSymbol, fullName, and companyID.
Quotes	name, currentPrice, tradeTime, lowPrice, highPrice, percentagePriceChange, bidding Price, askingPrice, tradeVolume, and marketCapitalization.
QuoteHistory	the same structure as the Quotes class.
Porfolios	accountID, companyID, stockPurchases, and saleOrders.
Accounts	userName, and password.
Currencies	countryName, currencyName, and exchangeRate
Personal	accountID, lastName, firstName, address, city,state, country, phoneNumber, and email.
Company	Name, address, city,state, country, phoneNumber, and email.

4.1 RT-ODL: Syntax and Semantics

The RT-ODL is a specification language used to define the specification of object types that conform to the RT Object Model. The RT-ODL provides object schemas portability across conforming RT object database management systems.

The RT-ODL is a DDL of RT object types. It defines the characteristics of types, including their properties and operations. The RT-ODL does not provide a complete definition of the methods but only defines signatures of operations. The RT-ODL is a superset of ODL: it recognizes the syntax of the ODL but it also provides additional syntactic constructs to manipulate RT information. In the following, we present RT-ODL Extended Backus Naur Form (EBNF) constructs for specifying RT Classes, RT Attribute, and RT Operation. Table 2 provides the top-level EBNF for the whole RT-ODL.

- RT-ODL RT Class construct: This construct models the RT Class concept of our RT Object Model. It defines the abstract state of RT objects stored in an object database management system. RT Classes, like conventional classes, are linked in a single inheritance hierarchy whereby state and behavior are inherited from an extender class. RT Classes may define keys and extents over their instances.
- RT-ODL RT Attribute construct: This construct models the RT Attribute concept of our RT Object Model. It corresponds to the RT data stored in a RT database and it incorporates fields that support logical constraints and temporal constraints.
- RT-ODL RT Operation construct: This construct corresponds to the RT Operation concept of our RT Object Model. RT-ODL is compatible with ODL for the specification of operations.

```
RTclass Quotes {
(extent QuotesInformation key QuotesID): persistent
attribute String QuotesID;
RTattribute Sensor currentPrice;
attribute DateTime tradeTime;
RTattribute Derived percentageOfChange;
void updateCurrentPrice (in float p);
void computeChange ();
relationship QuotesHistory history inverse
Quotes ::historicalValue;
};
```

Fig. 4 RT-ODL code

To meet the operational deadlines from event to system response, a RT database may apply different timing constraints on operations such as absolute timing constraints (e.g. execution time, earliest start time, latest allowed finish time) and periodic timing constraints (e.g. frequency of transaction initiation). In order to specify such timing constraints on running RT operation, a new

clause is added to the ODL basic syntax which is *timing_ constraint_expr*. The next program code, see figure 4, is an extract from the RT-ODL code of the Quotes class presented previously at the beginning of this section.

4.2 RT-OQL: Syntax and Semantics

Users need RT query language to specify the semantics knowledge captured in RT databases and to use it in a various ways. In this section, we introduce a RT query language, called RT-OQL, for manipulating RT object-oriented databases. RT-OQL is an OQL-like language supporting the basic structure of OQL. This latter, like SQL, uses a select-from-where structure to write more complex queries. Because of the strong similarities between SQL and OQL, the explanations in the following are dedicated to the specification of timing constraints in queries. The more interested reader in OQL syntax is referred to the OQL in [11].

The RT-OQL supports the proposed RT Object Model. It performs complex objects without privileging the set construct and the select-from-where clause. RT-OQL recognizes the syntax of OQL but it also provides additional syntactic features to specify RT queries. If these additional syntactic features were not used, then the semantics of a query in RT-OQL would become the semantics of this query in OQL. For example, the following query (figure 5) does not consider the RT constructs of objects:

Query 1): Give me the current price of the quote named CAC40.

```
select i.currentPrice
from i in Quotes
where i.name = "CAC40";
```

Fig. 5 query 1 code

Typical OQL queries do not provide any mechanisms for placing constraints on statements. Timing constraints on execution are used to define the semantics of what constitutes the correct execution of a statement with respect to time. In the following, we discuss how RT-OQL extends the select-from-where structure with timing constraints to provide this functionality. In order to propose those constraints, we have been inspired by the work proposed in [8].

4.2.1 Specification of Execution Timing Constraints

RT-OQL specifies execution timing constraints by placing timing constraints on individual statement or block of statements. This specification uses the following EBNF clauses (figure 6) :

```
<timing_Constraint> ::= start after <datetime_Exp>
                      | start before <datetime_Exp>
                     | complete after <datetime_Exp>
                    | complete before <datetime_Exp>
                    | period <interval_Exp>
<opt_Start_At> <opt_Until>
<opt_Start_At> ::= start aT <datetime_Exp>
<opt_Until> ::= until <search_Condition>
<datetime_Exp>::= CURRENT_DATE | CURRENT_TIME
[<time_precision>]|CURRENT_TIMESTAMP
[<timestamp_precision>]
```

Fig. 6 EBNF code

The semantics of the timing constraints is explained in Table 2.

The following (figure 7) are some illustrative examples of statements specifying timing constraints on execution.

Query 2): Give me the currentPrice, lowPrice, and highPrice of the quote named CAC40.

```
SELECT A.currentPrice,A.lowPrice, A.highPrice
FROM A IN Quotes
WHERE A.name = "CAC40"
COMPLETE BEFORE CURRENT_TIME + 30 SECOND;
```

Fig. 7 Query 2 code

In this query example, the execution timing constraint on the statement specifies that it must complete execution within 30 seconds.

Query 3): Give me the name and the percentage of price change of quotes where the value of the tradeTime column is between 10-01-2019 and 20-01-2019.

```
SELECT F.name, F.percentagePriceChange
FROM F IN Quotes
WHERE F.tradeTime between '10-01-2019' and '20-01-2019'
START BEFORE CURRENT_TIME + 10 SECOND
COMPLETE BEFORE CURRENT_TIME + 30 SECOND;
```

Fig. 8 Query 3 code

Here, the execution timing constraint on the statement specifies that the execution of the statement should begin execution within 10 seconds and it must complete within 30 seconds.

Query 4):

```
B: BEGIN
SELECT A.speed, A.altitude, A.path
FROM A IN Avions
WHERE A.identifier = "A001380"
COMPLETE BEFORE CURRENT_TIMESTAMP + 20 SECOND;
SELECT C.captain
FROM C IN FlightPlane
WHERE C.destination= "New York"
COMPLETE BEFORE CURRENT_TIMESTAMP + 30 SECOND;
END B COMPLETE BEFORE CURRENT_TIMESTAMP + 1 MINUTE;
```

Fig. 9 Query 4 code

Here, we have a block of statements. The timing constraints on the compound statements specify that it must complete within 1 minute. Note that the value of CURRENT TIMESTAMP will be the same for the three-timing constraints since they are in the same compound statement.

4.2.1 Transaction Structural Specification

Programs that use persistent objects are organized into transactions. Transaction management is an important functionality in the object database management system and it is fundamental to ensure data integrity, shareability, and recovery. Any creation, deletion, modification, and access of persistent objects must be done within the scope of a transaction. In our RT Object Model, a RT transaction may be aperiodic, periodic, or sporadic. It has timing constraints such as deadlines and periods. To support these kinds of transactions within an object database management system, we define two types: RealTimeTransactionFactory and RealTimeTransaction. The RealTimeTransactionFactory type is used to create transactions. The following operations are defined in the RealTimeTransactionFactory interface, see figure 10:

```
interface RealTimeTransactionFactory {
RealTimeTransaction new();
RealTimeTransaction current();
};
```

Fig. 10 RealTimeTransactionFactory interface code

The new operation creates RT transaction objects. The current operation returns the RT transaction that is associated with the current thread of control. If there is no such association, the current operation returns nil. Once a RealTimeTransaction object is created, it is manipulated using the RealTimeTransaction interface. The following operations are defined in the RealTimeTransaction interface, see figure 11:

```
interface RealTimeTransaction {
attribute DateTime deadline;
attribute integer period;
void begin() raises(TransactionInProgress,
DatabaseClosed);
void commit() raises(TransactionNotInProgress);
void abort() raises(TransactionNotInProgress);
void checkpoint() raises(TransactionNotInProgress);
void join() raises(TransactionNotInProgress);
void leave() raises(TransactionNotInProgress);
boolean isOpen();
};
```

Fig. 11 RealTimeTransaction interface code

After a RT transaction object is created, it is initially closed. An explicit begin operation is required to open a transaction. The commit operation causes all persistent objects created or modified during a RT transaction to be written to the DB and to become accessible to other RT or non-RT transaction objects running against that DB. The abort operation causes the RT transaction object to complete and become closed.

5. Implementation and Performance Evaluation

5.1 Implementation

To implement RT2O in a prototype system, we have extended the EyeDB object-oriented database management system [10]. The open, modular design of EyeDB facilitates extending it with properties to support the specification and management of RT2O architecture. In the initial version of EyeDB, a database schema is specified as a collection of Java classes or as ODL programs and transactions are specified as Java programs, or as OQL programs that are compiled to Java programs. Recall that objects and transactions in the RT2O object model have additional features beyond those supplied by Java classes and programs. To handle these additional properties, we have added the RT extensions proposed by our RT-ODL AND RT-OQL languages, described previously in this paper, to the standard EyeDB languages. Recently, there is a trend in object databases to integrate the query language with the programming language. This approach is applied by RT2O programmatic query interfaces, namely RT-OML (Real-Time Object Manipulation language). RT-OML is an extension of ODMG OML language [11] that defines the binding between the ODMG Object Model (ODL and OML) and

the Java programming language as defined by the Java Platform. Our RT-OML language is based on Java Real-Time Specification (denoted RTSJ) which defines how Java should behave in a RT computing context [26]. The complete syntax of the RT-OML is not the scope of this paper. The following are two illustrative examples of using RT-OML taken from the OST case study.

```
List quotes = database.query(new Predicate(){
public boolean match(Quotes quote){
return quote.currentPrice
&& quote.getName().equals("CAC40");
&& completeBefore(System.currentTimeMillis()+30)}});
```

Fig. 12 The current price code

This coding program in figure 12 gives the current price of the quote named CAC40 (similar to Query 2). The execution timing constraint on the retrieved query specifies that it must complete execution within 30 seconds.
Let's constrain our query to those quotes belonging to a specific company.

```
final String companyName = "CYAH";
List<quotes> results = database.query(
new Predicate<Quotes>(){
public boolean match(Company company){
return quote.currentPrice
&& company.getName().equals(companyName);
&& completeBefore(System.currentTimeMillis()+30)}});
listResult(results);
```

Fig. 13 The current price for a specific company code

In order to support the Java RTS Real-Time Specification [26], our RT2O system uses a RT POSIX-compliant operating system [20]. By POSIX-compliant RT operating system, we mean an operating system that supports several important aspects of POSIX RT extensions, defined in IEEE Std. 1003.1[20]. Our current release of RT2O executes on 2.6.32 RT Linux kernel. RT linux contains many of the RT operating systems features specified in the IEEE POSIX RT operating systems standard [20].

5.2 Performance Evaluation

In this section, we evaluate the performance of RT2O for an increasing number of client threads. For RT2O server, we use a Lenovo laptop that has 4th Generation Intel Core i7 processor and 8 GB memory with the 2.6.32 RT Linux kernel. We use two Acer laptops to create up to 2000 client threads. These two laptops have 4 GB memory. The client and server machines are connected via an Ethernet switch. Each client machine generates between 500 to 1000 client threads. Therefore, we generate 1000-2000 client threads for performance evaluation.

5.2.1 Performance Metrics

The primary performance metrics used in the experiments are miss ratio and data freshness.

- Miss Ratio (MR): The transaction miss ratio is defined as follows:

$$MR = 100 * \frac{N_{miss}}{N_{miss} + N_{succeed}} (\%) \quad (4)$$

where N_{miss} is the number of transactions that have missed their deadline and $N_{succeed}$ is the number of transactions that succeed.
When admission control is on, MR can be rewritten as follows:

$$MR = 100 * \frac{N_{miss} + N_{rejected}}{N_{miss} + N_{succeed} + N_{rejected}} (\%) \quad (5)$$

where $N_{rejected}$ is the number of transactions rejected by the admission controller.

- Data Freshness (DF): a data item di is considered fresh if CurrentTime - Timestamp(d_i) ≤AVI(d_i), where AVI is the absolute validity interval and timestamp indicates the latest observation of this data item in the real-world. The database freshness can also be measured. It is defined as follows: the ratio between fresh data and all the data in the database.

5.3 Performance Parameters:

Update transactions periodically update the stock data in the RT2O server. For example, the next code (figure 14) is used to update the current price of a specified quote. This program uses our RT-OML language syntax.

```
ObjectSet result = database.QueryByExample(
new Quotes("CAC40"));
Quotes found =(Quotes) result.next();
found.updateCurrentPrice(100.12);
database.store(found);
```

Fig. 14 The Update of the current price code based on RT-OML language

In addition to RT data updates, user transactions can read the RT data to track, evaluate, and manage investments. For instance, a client can select for a specified quote her associated current price. The RT-OQL program code of this request is as follows (figure 15):

```
SELECT Q.currentPrice
FROM Q IN QuotesInformation
WHERE Q.QuotesID = "CAC40"
COMPLETE BEFORE CURRENT_TIME + INTERVAL 20 SECOND;
```

Fig. 15 The RT-OQL program code

We have studied and evaluated the behavior of RT2O according to a set of performance metrics. The performance evaluation is undertaken by a set of extensive experiments using the developed stock trading workloads in RT2O. Table 2, Table 3, and Table 4 summarize experiments parameters.

Table 2: Attributes of RT data and Transactions

Notation	Description
AVI(d)	Absolute data validity interval of a RT data item
A(tr)	Arrival time of a transaction
DL(tr)	Deadline of a transaction
PR(tr)	Priority of a transaction
UP(tr)	Update period of a transaction
RDL(tr)	Relative deadline of a transaction
LAVI	Length of absolute data validity interval

Table 3: User and update transactions

Update transaction	Update frequency	UP(tr) = 1/2*LAVI of RT data to be updated
	Deadline	DL(tr)=A(tr)+UP(tr)
User transaction	Deadline	DL(tr)=A(tr)+RDL(tr)

Table 4: Workload parameters

Update transaction	Number of transactions types	8
	LAVI	Varied (0.5s, 1s, 1.5s)
	Scheduling policy	EDF
User transaction	Number of transactions types	4
	Transaction length	Varied by transactions
	Total number of transaction requests	500
	Scheduling policy	EDF

5.4 Summary of Results and Discussion

Given the user and update transaction workload setting, we compare the performance of (i) a baseline approach (denoted BASE), (ii) Admission Control (denoted AC), (iii) MDE-based update policy (denoted MDE-UP), and (iv) Data Manager (denoted DM) for an increasing number of clients threads. The BASE approach accepts all incoming tasks and updates every RT data without using the admission controller, the data manager, and the MDE-based update policy.

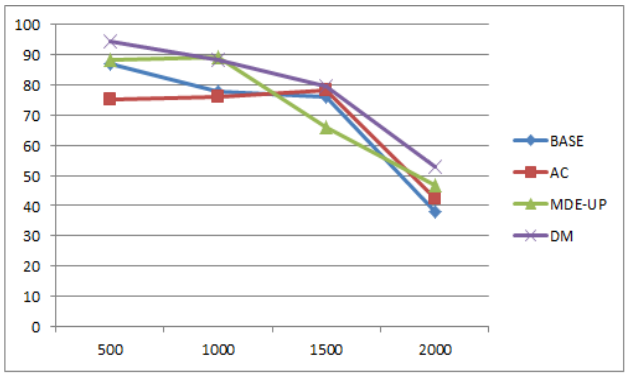

Fig. 16 Success Rate

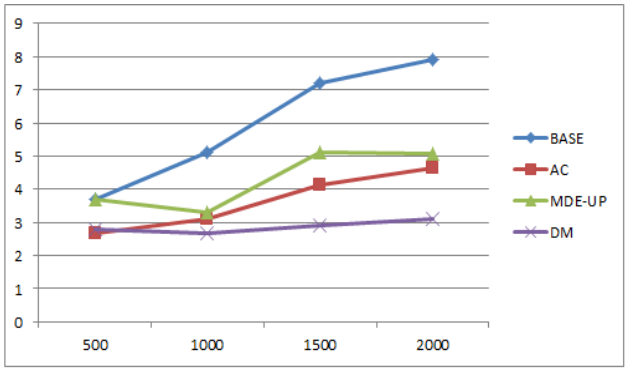

Fig. 17 Average Response Time

Figure 16 shows the number of user transactions per second (utps), i.e., success rate, that complete within the desired delay bound (in this paper we use 5 s as the desired response time-bound). Generally, the success rate of DM is the highest among the four approaches. For 500 client threads, DM achieves 94.62 utps and BASE 87.12 utps. When the number of client threads increases, the success rate of BASE quickly drops. For 2000 clients threads, BASE achieves only 37.92 utps due to severe overloads, while DM can achieve 52.86 utps. For 1000 client threads, MDE-UP achieves a higher success rate than BASE, AC, and DM. In fact, MDE-UP may always be able to improve the transaction timeliness. This result coincides with the QoS work [23] in which temporal data imprecision can consistently improve RT database performance compared to baseline approaches.

In Figure 16, AC generally presents the lowest success rate when the number of clients threads is 1500. Under overload, however, AC performs well. For 2000 client threads, its success rate is 42.35 utps, which is close to the success rate of DM.

This implies that admission control needs to be applied only under severe overloads. This result contrasts with the simulation-based QoS work [23] in which admission control improves the timeliness compared to BASE for most of the tested workloads. From this result, one can observe that RT database performance evaluation in a real system is required.

In Figure 17, DM shows the best average response time. This is because multi-versions data architecture allows limiting the deadline miss ratio even in the presence of unpredictable workloads. Moreover, this technique ensures the freshness of data accessed by timely transactions. So data used bey committed transactions are always fresh. This result coincides with the simulation-based work [24] in which a multi-version data notion allows to execute transactions on time using fresh and precise data.

According to the results found, table 4 compares the proposed architecture and architectures discussed in the related section. This table summarizes the benefits of the proposed RT2O database system.

Table 7: Comparison of different RT database prototypes

Prototypes	Data model	QoS management	Operating System	RT Query Language	Open Source
DeeDS	Relational	Not Addressed	OSE delta	Not Addressed	No
BeeHive	Object Oriented	Partially Addressed	Not specified	Not Addressed	No
RTSORAC	Object Oriented	Partially Addressed	Not specified	RTSQL	No
RODAIN	Object Oriented	Partially Addressed	Not specified	Not Addressed	No
StarBase	Relational	Not Addressed	RT-Mach	Not Addressed	No
Chronos	Relational	Feedback Control	Linux	Not Addressed	Yes
ODEA	Object Oriented	Feedback Control	RT Linux	RTNQL	No
RT2O	Object Oriented	Feedback Control	RT Linux	RTQL	Yes

3. Conclusions

RT database can be employed in a number of data-intensive RT applications such as agile manufacturing, traffic control, and target tracking. Due to the absence of a publicly available RT database testbed, it is very hard to evaluate RT data management techniques in a realistic environment. To address this problem, we develop a RT object-oriented environment for RT database application development, called RT2O. Timing constraints of data and transactions are considered throughout the design, development, and evaluation of RT2O. RT2O differs from previous RT database work in that (i) it relies on a RT operating system that provides time-based synchronization and priority-based scheduling, and (ii) it uses a RT query language that supports RT database requirements. We also develop a RT database QoS management approach in RT2O to detect overload detection and adjust the workload. In the future, we will further enhance our RT database testbed. We will investigate new techniques for RT database QoS management.

References

[1] K. Ramamritham, S. H. Son, and L. C. Dipippo, “Real-Time Databases and Data Services,” Real-Time System, vol. 28, no. 2, pp. 179–215, 2004.

[2] M. Ben Ayed, S. Elkosantini, S. Alshaya, and M. Abid, " Suspicious behavior recognition based on face features", IEEE Access, Vol. 7, 2019.

[3] L. Martin, “EagleSpeed Real-Time Database Manager,” 1998.

[4] P. Plc, “Polyhedra White Papers,” 2002.

[5] Oracle, “Oracle TimesTen In-Memory Database”, https://www.oracle.com/database/technologies/related/timesten.html.

[6] Majhi. MC, Behera. AK, Kulshreshtha. NM, Mahmooduzafar, Kumar. R, Kumar. A, “ExtremeDB: A Unified Web Repository of Extremophilic Archaea and Bacteria”, PLOS ONE, Vol. 12(6), 2017.

[7] K.-D. Kang, J. Oh, and S. H. Son, “Chronos: Feedback Control of a Real Database System Performance,” in RTSS, 2007, pp. 267–276.

[8] J. Prichard and P. Fortier, “Standardizing Real-Time Databases — RTSQL, in Real-Time Database and Information Systems: Research Advances", The Springer International Series in Engineering and Computer Science. Springer US, vol. 420, pp. 289–310, 1997.

[9] C. Lu, J. Stankovich, G. Tao, and S. Son, “Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms,” Real-Time Systems, vol. 23, no. 1/2, pp. 85–126, 2002.

[10] M. Vojtech, “Database standard ODMG and system EyeDB", 18th International Scientific Conference on Agrarian Perspectives - Strategies for the Future, 2009.

[11] Sampaio, SDM, Paton, NW, Smith, J, Watson, P, " Measuring and modelling the performance of a parallel ODMG compliant object database server ", Concurrency And Computation-Practice & Experience, Vol. 18(1), pp. 63-109, 2006.

[12] R. Abbott and H. Garcia-Molina, “Scheduling real-time transactions,” SIGMOD Rec., vol. 17(1), pp. 71–81, 1988.

[13] J. R. Haritsa, M. Livny, and M. J. Carey, “Earliest Deadline Scheduling for Real-Time Database Systems,” Proceedings Twelfth Real-Time Systems Symposium, 1991, pp. 232–242.

[14] S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley, “Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems”, Real-Time Systems, Vol. 3(3), pp 307–336, 1991.

[15] Y.-y. Xiao, H. Zhang, and F.-y. Wang, “Maintaining Temporal Consistency in Real-Time Database Systems”, International conference on Convergence Information Technology, pp. 1627–1633, 2007.

[16] M. Xiong, R. Sivasankaran, J. Stankovic, K. Ramamritham, and D. Towsley, "Sceduling transactions with temporal

constraints: Exploiting Data semantics", Real-Time Systems Symposium, 1996.
[17] B. S. Adelberg, “Strip: a soft real-time main memory database for open systems,” Ph.D. dissertation, Stanford, CA, USA, 1997, uMI Order No.GAX97-23315.
[18] J. A. Stankovic, S. H. Son, and J. Liebeherr, “BeeHive: Global Multimedia Database Support for Dependable, Real-Time Applications”, Lecture Notes in Computer Science, 1997.
[19] C.-S. Peng, K.-J. Lin, and C. Boettcher, “Real-Time Database Benchmark Design for Avionics Systems”, Real-Time Database Systems pp 123-138, 1997.
[20] T. IEEE and T. O. Group, The Open Group Base Specifications Issue 7, 2008.
[21] K. Ramamritham, “Real-Time Databases”, Distributed and Parallel Databases, Vol. 1(2), pp. 199–226, 1993.
[22] Z. Ellouze, N. Louati, and R. Bouaziz, “A real-time object-oriented data model and prototype implementation”, 16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing, pp. 1–8, 2013.
[23] M. Amirijoo, J. Hansson, and S. H. Son, “Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations”, IEEE Trans. Computers, vol. 55(3), pp. 304–319, 2006.
[24] E. Bouazizi, C. Duvallet, and B. Sadeg, “Multi-Versions Data for Improvement of QoS in RTDBS”, 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications , pp. 293–296, 2005.
[25] K.-D. Kang, P. H. Sin, and J. Oh, “A Real-Time Database Testbed and Performance Evaluation”, 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 319–326, 2007.
[26] J. Gosling and G. Bollella, "The Real-Time Specification for Java", Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.,2000.
[27] B. Adelberg, H. Garcia-Molina, and B. Kao, “Applying update streams in a soft real-time database system”, ACM SIGMOD Record, vol. 24(2), pp.245–256, 1995.
[28] J. Huang, J. A. Stankovic, D. Towsley, and K. Ramamritham, “Experimental Evaluation of Real-Time Transaction Processing”, Proceedings. Real-Time Systems Symposium,1989.
[29] A. P. Buchmann, H. Branding, T. Kudrass, and J. Zimmermann,“REACH: a REal-time, ACtive and Heterogeneous mediator system”, IEEE Computer Society, vol. 15, no. 1-4, pp. 44–47, 1992.
[30] J. Prichard, L. DiPippo, J. Packham, and V. Fay-Wolfe, “RTSORAC: A Real-Time Object-Oriented Database Model”, International Conference on Database and Expert Systems Applications, pp. 601–610, 1994.
[31] S. F. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Eftring, “DeeDS towards a distributed and active real-time database system,” ACM SIGMOD Record, vol. 25(1), pp. 38–51, 1996.
[32] J. Taina and K. Raatikainen, “RODAIN: a real-time object-oriented database system for telecommunications”, in CIKM. ACM, pp.10–14, 1996.
[33] S. Kim, S. H. Son, and J. A. Stankovic, “Performance Evaluation on a Real-Time Database,” in RTAS, pp. 253–265, 2002.
[34] Y. K. Kim and S. H. Son, “Developing a Real-Time DB: The Starbase Experience,” in Real-Time Database Systems, ser. The Springer International Series in Engineering and Computer Science. Springer US, vol. 396, pp. 305–324, 1997.
[35] Z. Ellouze, N. Louati, and R. Bouaziz, “A Next Generation Object- Oriented Environment for Real-Time Database Application Development", 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 1224–1232, 2013.