MDE-based QoS Management Framework for RTDB Management Systems Development

Salwa M'barek, Leila Baccouche, Henda Ben Ghezala
RIADI-GDL laboratory
INSAT, National Institute of Applied Science and Technology
C.U. Nord, B.P 676, Tunis Cedex 1080, Tunisia
salwa.mbarek@riadi.rnu.tn leila.baccouche@insat.rnu.tn henda.bg@cck.rnu.tn

Abstract—**This paper sets out a framework for real-time database management systems (RTDBMS) model design integrating QoS management. We use a Model Driven Engineering (MDE) approach based on model transformation techniques. The aimed systems apply the feedback control scheduling for QoS management which gives a robust and controlled behavior of the system even in transient overloads. The framework provides metamodels and processes to extend, reuse and transform RTDBMS models for different QoS requirements and different real-time applications. A RTDBMS design tool has been developed based on EMF (Eclipse Metamodeling Framework) and Kermeta metamodeling and transformation language.**

Keywords-real-time database management systems; QoS management; feedback control scheduling; MDE; model transformation ; Kermeta.

I. INTRODUCTION

The real-time database management systems (**RTDBMS**) are database management systems manipulating real-time data and real-time transactions with time constraints [4, 15].

The *real-time data* must be updated periodically by *real-time update transactions* to reflect the real world state at any time, otherwise they become unfresh which may cause a disaster.

The *real-time user transactions* which have to access real-time data, must be executed within a deadline otherwise they become useless for the application.

Recent works in RTDBMS [1, 2, 3, 8, 9, 10, 16] propose QoS management architectures and QoS management algorithms based on the *Feedback Control Scheduling Architecture* (**FCSA**) to give a robust and controlled behavior of the RTDB even during transient overloads and when we have inaccurate run-time estimates of the transactions [12].

We propose a model design framework for RTDBMS using FCSA, which is based on the **MDE** (*Model Driven Engineering*) approach. This new approach for software systems engineering is centered on the models and not on the implementation [11].

Our aim is to support designers to easily set up the appropriate model of the RTDBMS with a QoS management approach based on the Feedback Control Scheduling. Moreover, to satisfy new real time requirements, persistent RTDBMS models can be easily reused, extended and combined based on model transformation techniques.

The framework provides metamodels and processes for RTBDMS model and code generation.

This paper begins in Section 2 with an overview of the MDE (*Model Driven Engineering*) approach. Section 3 sets out the Feedback Control Scheduling Architecture (FCSA) for QoS management in RTDBMS. The metamodel of Feedback Control Loops is presented in Section 4. Section 5 gives a metamodel of QoS management approaches in RTDBMS. Section 6 illustrates the proposed MDE-based framework for RTDBMS development. The Model transformation to integrate the QoS management to basic RTDBMS is explained in Section 7. We conclude this paper by a summary of contributions and perspectives.

II. MODEL DRIVEN ENGINEERING

The *model-driven engineering* (**MDE**) approach has allowed several significant improvements in the development of complex systems by putting the focus on a more abstract concern than the classical programming. It is a form of generative engineering in which (all or part of) an application is generated from models [5, 6].

Modeling allows the generation of parts of an application instead of implementing the source code manually. This increases the development speed and even more importantly, it increases the implementation quality. Models can be checked for consistency before source code is created from them. If an application evolves, changes only have to be applied in the model, while the source code can be re-generated automatically.

Models provide a higher level of abstraction than source code. Developers can focus on key aspects of an application, instead of dealing with the complexities inherent in a programming language. The creation of custom models, so-called Domain-Specific Languages (DSL), can make the application understandable without a background in programming.

The Eclipse Modeling Top-Level Project facilitates MDE for Eclipse and is one of the biggest and most active areas in the Eclipse ecosystem. The Eclipse Modeling Framework (EMF) builds the foundation for a variety of modeling technologies such as the Graphical Modeling Framework (GMF) or textual modeling (XText), and has become a widely used standard for modeling worldwide.

III. FEEDBACK CONTROL SCHEDULING ARCHITECTURE

The *Feedback Control Scheduling Architecture* (**FCSA**) as visualized in Figure 1, gives a robust and controlled behavior of the RTDBMS even during transient overloads and when we have inaccurate run-time estimates of the transactions [12].

This architecture is based on the following principle: "***observation then auto-adaptation***". The database administrator defines some parameters and their thresholds to give the QoS specification. For instance the *Miss Ratio (MR)* is a QoS parameter which measures the percentage of user transactions that missed their deadlines (MR ≤ 20%). The observation consists on measuring periodically the *QoS parameters* to compute the system performance.

The *auto-adaptation* consists on adjusting the system behavior, using *control loops, update policies* and *QoS management algorithms.*

The Figure 1 shows the FCSA as proposed in [10]. It consists of several main components:

Sources generate *user transactions* to be submitted to the system. Each *Update Stream* periodically submits an *update transaction* for a certain temporal data object (real-time data). *Admission control* is applied to *user transactions*.

Transaction handler consists of a *concurrency controller* (CC), a *freshness manager* (FM) and a *basic scheduler*.

Update transactions with *highest importance* are scheduled in the *high priority ready queue* while *user transactions* and *Update transactions* with *lowest importance* are scheduled in the *low priority queue*. In each queue, transactions are scheduled in **EDF** (Earliest Deadline First) manner. A transaction can be aborted and restarted by CC. It can also be preempted by a higher priority transaction. *Freshness manager* (FM) checks the freshness of real-time data before the initiation of a *user transaction*. FM blocks the corresponding transaction if an accessing data is currently stale. The blocked transaction(s) will be transferred from the block queue to the ready queue as soon as the corresponding update commits.

Monitor periodically measures ***QoS parameters*** (miss ratio, utilization, and perceived freshness) and reports the statistics to the *feedback QoS controllers* (MR/Utilization Controllers) and *QoD manager*. *QoS controllers* compute the ***control signals*** **ΔU** based on the current performance error using the **PID control** (Proportional Integral Derivative control) [14, 15].

QoD Manager adapts the *update policy*, if necessary. It informs the *admission controller* of the new ***control signal*** **ΔU (ΔU_{new})** after potential QoD adaptations. *Update scheduler* decides whether or not to schedule an incoming

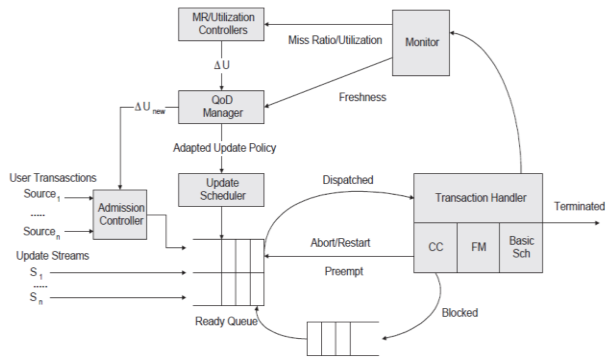

Figure 1 : Feedback Control Scheduling Architecture (FCSA)

update depending on the selected *update policy*.

IV. THE FEEDBACK CONTROL LOOPS METAMODEL

An important step in designing the FCSA of an RTDBMS is to decide the following concepts: controlled variables, performance reference, control signal, manipulated variable, control loop and control function.

1. *Controlled variables* are the performance metrics controlled by the scheduler. Controlled variables of a real-time system may include the *deadline miss ratio* $M(k)$ and the *CPU utilization* $U(k)$ (also called miss ratio and utilization, respectively), both defined over a time window ($(k\text{-}1)W$, kW), where W is the *sampling period* and k is called the *sampling instant*. The miss ratio $M(k)$ at the k_{th} sampling instant is defined as the number of deadline misses divided by the total number of completed and aborted tasks in a sampling window ($(k\text{-}1)W$, kW). Miss ratio is usually the most important performance metric in a real-time system.

The utilization $U(k)$ at the k_{th} sampling instant is the percentage of CPU busy time in a sampling window ($(k\text{-}1)W$, kW). CPU utilization is regarded as a controlled variable for real-time systems due to cost and throughput considerations. CPU utilization is important because of its direct linkage with the deadline miss ratio [1, 3, 10].

2. *Performance reference* is a target value specified by the DBA for a specific *controlled variable*. Each controlled variable must converge to its performance reference (reference). For instance: in steady state, the controlled variable MR must be less than 30% so its reference is MR_r =30%. An overshoot noted Mp is allowed in transient overloads; so that

$$MR \le MR_r \times (M_p + 100)\%$$

3. *Control loop is a closed loop* using a *control function* to generate a performance adjusting signal called a *control signal*. The entry of the control loop is the error e(t) between the target value of a controlled variable and its current value. There is a control loop for each controlled variable. The unique control signal generated by the QoS Controller is derived from all control signals generated by its control loops.

$$\mathrm{e(t)} = controlled\ variable\ reference - measured\ controlled\ variable$$

4. *Control signal* (generally noted ΔU) is provided periodically by control loops of the ***QoS Controller***. It is computed by a certain *control function* based on the error e(t) between the target values of the controlled variables (performance reference) and their measured values. In [1], the control signal ΔU which is the requested CPU utilization adjustment is computed as follows.

$$\Delta U(t) = \boldsymbol{k_p} \,.\, \mathrm{e}(t) + \boldsymbol{k_i}.\int_0^{\mathrm{t}} \mathrm{e}(t).\,\mathrm{dt} + \boldsymbol{k_d}.\frac{\mathrm{de}(t)}{\mathrm{dt}}$$
$$k_p, k_i, k_d \text{ are PID parameters}$$
$$\mathrm{e(t)} = MR_r - MR(t)$$

5. *Control function* represents the relation between the *control signal* ΔU(t) and the *controlled variable error e(t).*

$$\Delta U(t) = \boldsymbol{f}(e(t))$$
$$\mathrm{e(t)} = V_r - V(t)\ ;\ \textit{V is the controlled variable}$$

6. *Manipulated variable* is a QoS parameter which has an impact on the performance of the system and the controlled variables. Its value must be adjusted dynamically to guarantee QoS specification and so system robustness. For instance, the data freshness has an impact on the miss ratio MR, so, it can be considered as a manipulated variable. In fact, decreasing the number of update transactions will degrade data freshness. Consequently, the number of completed user transaction will increase and so the average miss ratio MR will decrease.

Auto-adaptation consists on adjusting (decreasing or increasing); by a certain function; the value of the *manipulated variable* depending on the value of the *control signal* ΔU which is computed from *Controlled variables errors* (based on PID function or other control function).

The *QoS management algorithm* makes, at each sampling period, the *Auto-adaptation*. It is running on the *QoD Manager* which is considered as the *QoS regulator*. Regulation orders come from *QoS controller* which sends him the *control signal*.

We propose a metamodel to design feedback control loops for QoS management in RTBDMS presented in Figure 2. This metamodel establish relations between the different concepts explained in this section.

From this metamodel, different models can be generated for specific requirements.

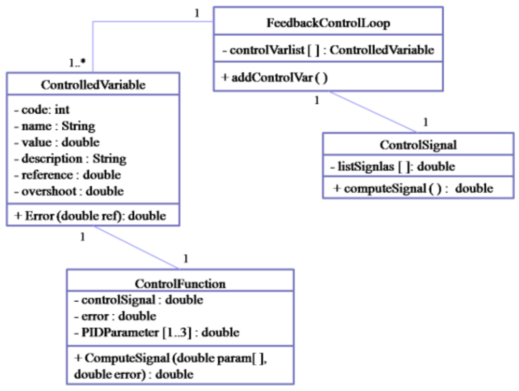

Figure 2 : Feedback Control Loops metamodel

V. QOS MANAGEMENT APPROACHES METAMODEL

All studied works [1, 2, 3, 8, 9, 10, 16] use the Feedback Control Scheduling Architecture. However, we notice many differences on their QoS management approaches.

They propose different QoS metrics. The specification of feedback control loops in the QoS controller varies from an approach to another. They are applying the PID control (Proportional Integral Derivative Control) [14, 15] as a control function, but there are some differences in their formula.

Each approach use only two kinds of transactions: user transactions and update transactions with firm deadlines (if they miss their deadlines, they will simply be rejected from the system).

Compared approaches use different transaction models. In [1, 7], they propose the milestone model where a transaction is decomposed into one mandatory sub-transaction which must obligatory meet its deadline and many optional sub-transactions which can be rejected in overload situations without affecting QoS specification.

However, in [8, 12] they use a service differentiation. They don't decompose transactions, they classify them in three service classes regarding to an importance factor.

All approaches consider only base data which hold the view of the outside environment, in opposition to derived data which are derived from other base or derived data.

Each approach uses a specific Data model and update policy [8, 9, 12].

Even transactions queues are configured differently. Queues configuration depends on transactions model.

In these approaches, are applied different *update policies* (adaptive policy, MDE policy) and different *scheduling* *algorithms* (EDF: Earliest Deadline First, HEF: Highest Error First, HEDF: Highest Error Density First).

Many QoS management algorithms are proposed such as: FCS-IC1(Feedback Control Scheduling-Imprecise Computation-1) [1]; FCS-IC2 (Feedback Control Scheduling-Imprecise Computation-2) [1]; FCS-HEF (Feedback Control Scheduling-Highest Error First) [13]; FCS-HEDF (Feedback Control Scheduling- Highest Error Density First) [13], QMF1 (QoS-sensitive approach for Miss ratio and Freshness guarantees 1) [10] and QMF-Diff (QoS-sensitive approach for Miss ratio and Freshness guarantees with Differentiated Services) [10].

However, these algorithms are little similar. These algorithms try to balance the system load between user transactions and update transactions. For example, in overload situations, the QoD is decreased applying the corresponding QoS management algorithm until steady state reaching, where QoD will be increased without QoT violating. When the system is saturated, all arrived transactions are discarded by the admission controller [3, 9].

An evaluating study of these approaches and algorithms is detailed in [17].

We propose a metamodel of QoS managements approachs as shown in the following figure. Based on this metamodel we can derive any QoS management approach model for specific QoS requirements.

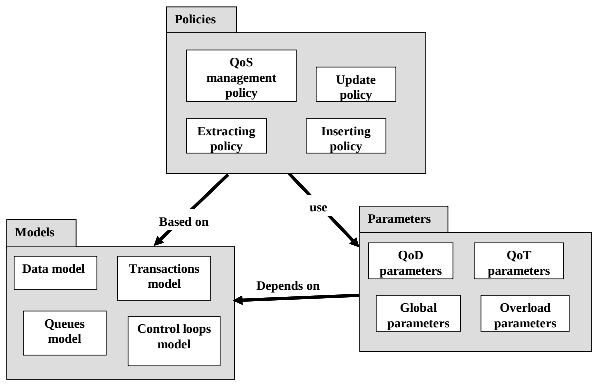

Figure 3 : QoS management Approachs metamodel

VI. Processes for RTBDMS development

Existing QoS management approaches are very interesting. However, it is difficult to reuse them or a part of them. Furthermore, it is very difficult to develop a new RTDBMS architecture from scratch, to extend or to reconfigure an existing architecture, to modify the QoS management algorithm or to add other QoS parameters and QoS specification.

The proposed framework tries to answer these issues. We are interested only in RTDB management systems with feedback control scheduling, because they are complex

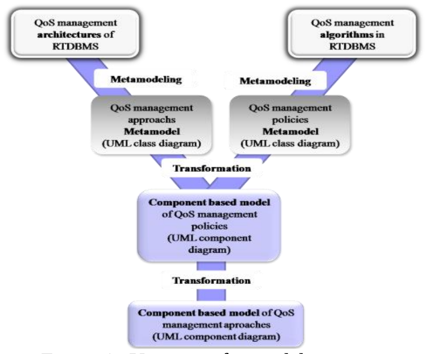
QoS management **architectures** of RTDBMS
QoS management **algorithms** in RTDBMS
Metamodeling
Metamodeling
QoS management approachs Metamodel (UML class diagram)
QoS management policies Metamodel (UML class diagram)
Transformation
Component based model of QoS management policies (UML component diagram)
Transformation
Component based model of QoS management aproaches (UML component diagram)

Figure 4 : Y-process for model generation

systems but at the same time, they are robust systems offering QoS constraints specification and management.

Our first aim was to design a metamodel for QoS management loops and then a metamodel for QoS management approaches.

Metamodels will be transformed to generate abstract component based models for QoS management in RTDBMS: Component based model of QoS management policy and Component based model of QoS management approach.

The component based models resulting from the Y-process are the entry to the second process (as shown in Figure 7) which allows the reuse of these models to build new ones and to generate the implementation into a specific language.

We built a three-layered database (Figure 7) for QoS models reuse and code generation. Component based models of QoS management approaches and policies are stored in the "Models level" of the database with platform models (J2EE…). Models are decomposed and components are stored separately in the "Component level". The data about metamodels, models, components and bindings are stored in the "Metadata level".

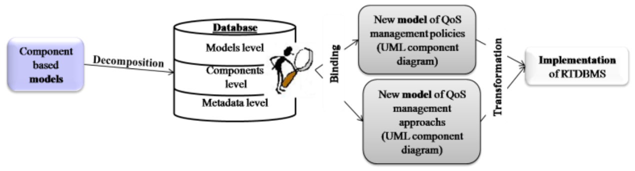

Figure 5: Model reuse process

VII. Framework implementation

Every real time database management system can be modeled using our metamodels. The QoS management layer can be added through models transformation as shown in the following figure. We used the Eclipse Modeling Framework tool to implement different metamodels conformant to the Ecore metametamodel and to generate models in the XMI (XML Metadata Interchange) format. The Kermeta langage is used to load and transform models.

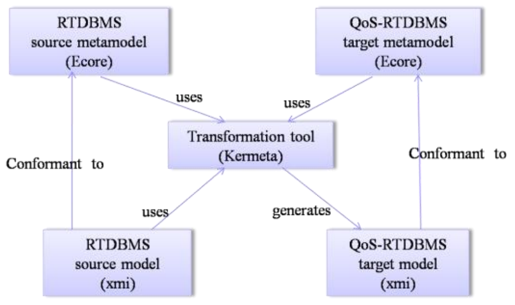
RTDBMS source metamodel (Ecore)
QoS-RTDBMS target metamodel (Ecore)
uses
uses
Transformation tool (Kermeta)
Conformant to
Conformant to
uses
generates
RTDBMS source model (xmi)
QoS-RTDBMS target model (xmi)

Figure 6 : Models transformation for QoS integation

A graphical user interface with the java language is under development to facilitate the RTDBMS model design and transformation for designer not expert at EMF and Kermeta.

VIII. CONCLUSION

This paper focused on the real-time database management systems (RTDBMS) using the Feedback Control Scheduling Architecture (FCSA) for QoS and time constraints management. Studied architectures are interesting but can't be easily reconfigured, extended or reused. To answer these issues, we proposed a model driven framework for QoS-aware RTDBMS development based on the Model Driven Engineering principles and the Model Driven Architecture standards. It provides processes, metamodels, component-based models, models transformation and models repository. It is possible to generate new approaches and QoS policies from stored ones or from scratch. Generated models are component based for two reasons: (1) make easy the reuse and reconfiguration of models (2) generate a code for component oriented platforms (J2EE). Object-oriented or aspect-oriented code may be generated through mapping between considered metamodels.

REFERENCES

[1] M. Amirijoo, "Algorithms for Managing Real-time Data Services Using Imprecise Computation", Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA), Taiwan, 2003.

[2] E. Bouazizi C. Duvallet and B. Sadeg, "Using Feedback Control Scheduling and Data Versions to enhance Quality of Data in RTDBSs". Proc. of IEEE International Computer System and Information Technology (IEEE ICSIT'2005), Alger, Algérie, 2005, pp. 322-327.

[3] E. Bouazizi, C. Duvallet and B. Sadeg, "Une nouvelle approche pour la gestion de la QdS dans les SGBD temps réel", Proc. of INFORSID'2006, Hammamet, Tunisie, 2006, pp. 547-559.

[4] L. Cingiser, H. Son and K. Ramamritham, "Real-Time Databases and Data Services", Journal of Real-Time Systems, 28, 179-215, 2004.

[5] B. Combemale, "Approche de métamodélisation pour la simulation et la vérification de modèle : Application à l'ingénierie des procédés", PhD Thesis, Institut National Polytechnique de Toulouse, 2008.

[6] B. Combemale, "Ingénierie Dirigée par les Modèles (IDM) : État de l'art", hal-00371565, 2008.

[7] J. Den Haan, "MDA and Model Transformation", 2008. http://www.theenterprisearchitect.eu/archive/2008/02/18/mda-and-model-transformation.

[8] C. Duvallet, E. Bouazizi and B. Sadeg, "Improvement of QoD and QoS in RTDBS". Proc. 14th International Conference on Real-Time and Network System (RTNS'2006), Poitiers, France, 2006, pp. 87-95.

[9] J. Hansson, M. Amirijoo and S. H. Son, "Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations". IEEE Transactions on Computers, 2006, V. 55, No. 3.

[10] K. Kang, S. Son A. Stankovic, "Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases". IEEE Transactions on knowledge and data engineering, Vol. 16, No. 10, 2004, p. *1200-1216.*

[11] S. Kent, "Model Driven Engineering", IFM 2002, 2002, p. 286-298.

[12] C. Lu, A. Stankovic, G. Tao and H. Son, "Feedback control real-time scheduling: Framework, modeling and algorithms", Journal of Real-Time Systems, vol.23, n. 1, 2002, p.85-126.

[13] O. Patrascoiu, "Model Transformations in YATL". Studies and Experiments, 2004, Technical Report 3-04.

[14] J.D. Poole, "Model-Driven Architecture: Vision, Standards And Emerging Technologies", ECOOP 2001, 2001.

[15] K. Ramamritham, "Real-Time Databases", International Journal of Distributed and Parallel Databases, 1996.

[16] B. Sadeg, C. Duvallet and E. Bouazizi, "Prise en compte des données dérivées temps réel dans une architecture de contrôle par retroaction". Proc. of MAJECSTIC'2006, 2006.

[17] S. M'barek, L. Baccouche and H. Ben Ghezala, "An evaluation of QoS management approaches in Real-Time Databases". ICONS 2008, 2008, p. 41-46.

[18] S. H. Son, M. Amirijoo and J. Hansson, "Specification and Management of QoS in Imprecise Real-Time Databases", IEEE Database Engineering and Applications Symposium (IDEAS), Hong Kong, 2003.

[19] D. Xue, Y. Chen and D.P. Atherton, "PID Controller Design", Linear Feedback Control, Chapter 6, 2007. Society for Industrial and Applied Mathematics.

[20] M.J. Willis, "Proportional-Integral-Derivative Control", 1999.