Modeling and Verification of Priority Assignment in Real-Time Databases Using Uppaal*

Martin Kot

Center for Applied Cybernetics, Department of Computer Science, FEI,
VSB - Technical University of Ostrava,
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
martin.kot@vsb.cz

Abstract. Real-time database management systems (RTDBMS) are recently subject of an intensive research. Model checking algorithms and verification tools are of great concern as well. In this paper we show some possibilities of using a verification tool Uppaal on some variants of priority assignment algorithms. We present some possible models of such algorithms expressed as nets of timed automata, which are a modeling language of Uppaal.

Keywords: real-time database systems, priority assignment, timed automata, model checking, verification, verification tool, Uppaal

1 Introduction

Many real-time applications need to store some data in a database. It is possible to use traditional database management systems (DBMS). But they are not able to guarantee any bounds on a response time. This is the reason why so-called real-time database management systems (RTDBMS) emerged (e.g. [1, 6]).

Formal verification is of great interest recently and finds its way quickly from theoretical papers into a real live. It can prove that a system (or more exactly a model of a system) has a desired behavior. The difference between testing and formal verification is that during testing only some possible computations are chosen. Formal verification can prove correctness of all possible computations. A drawback of formal verification is that for models with high descriptive power are almost all problems undecidable. It is important to find a model with an appropriate descriptive power to capture a behavior of a system, yet with algorithmically decidable verification problems.

There are two main approaches to fully automated verification – equivalence checking and model checking. Using equivalence checking, two models of systems (usually model of specification and model of implementation) are compared using

* Author acknowledges the support by the Czech Ministry of Education, Grant No. 1M0567.

J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2010, pp. 147–154, ISBN 978-80-7378-116-3.

some behavioral equivalence. In this paper we consider the other approach – so called model checking (see e.g. [4, 9]). This form of verification uses a model of a system in some formalism and a property expressed usually in the form of formula in some temporal logic. Model checking algorithm checks whether the property holds for the model of a system. There are quite many automated verification tools which implement model checking algorithms (see e.g. [11] for overview). Those tools use different modeling languages or formalisms and different logics.

The idea of our research is to explore possibilities of using existing verification tools on RTDBMS. To our best knowledge, there are only rare attempts of automated formal verification of real-time database system. In fact we know about one paper ([10]) only where authors suggested a new pessimistic protocol and verified it using Uppaal. They presented two small models covering only their protocol.

There is not any verification tool intended directly for real-time database systems. We have chosen the tool Uppaal because it is designed for real-time systems. But, it is supposed to be used on so-called reactive systems, which are quite different from database systems. So we need to find some possibilities how to deal with it. Big problem of verification tools is so called state space explosion. Uppaal is not able to manage too detailed models. On the other hand, too simple models can not catch important properties of a real system. So we need to find a suitable level of abstraction.

One of the most important and crucial parts of all database management systems allowing concurrent access to data records is concurrency control. Some of protocols used for concurrency control were modeled and verified using Uppaal in [7, 8].

In this paper, we will concentrate on other important part of real-time database systems – priority assignment. There are many parameters that can be used for determination of priority of database transaction. Some of them are criticality, deadline, amount of resources already used by transaction etc. There were several algorithms presented for this task. In this paper we will consider two of them presented in [1] – First Come First Serve (Section 4) and Earliest Deadline (Section 5). The nature of this paper should be mainly proof of concept. There are not any new informations found about described algorithms or any errors in them discovered. But some general possibilities of modeling using nets of timed automata are shown which can be used for verification and comparison of, e.g., newly designed algorithms in the future.

Before we will discuss concrete models of algorithms, we will shortly describe the tool Uppaal in the Section 2 and talk some general possibilities and assumptions over in Section 3.

2 Verification tool Uppaal

Uppaal ([3]) is a verification tool for real-time systems. It is jointly developed by Uppsala University and Aalborg University. It is designed to verify systems that can be modeled as networks of timed automata extended with some further

features such as integer variables, structured data types, user defined functions, channel synchronization and so on.

A timed automaton is a finite-state automaton extended with clock variables. A dense-time model, where clock variables have real number values and all clocks progress synchronously, is used. In Uppaal, several such automata working in parallel form a network of timed automata. An automaton has locations and edges. Each location has an optional name and invariant. An invariant is a conjunction of side-effect free expressions of the form $x < e$ or $x \leq e$ where x is a clock variable and e evaluates to an integer. Each automaton has exactly one initial location.

Particular automata in the network synchronize using channels and values can be passed between them using shared (global) variables. A state of the system is defined by the locations of all automata and the values of clocks and discrete variables. The state can be changed in two ways - passing of time (increasing values of all clocks by the same amount) and firing an edge of some automaton (possibly synchronizing with another automaton or other automata). Some locations may be marked as committed. If at least one automaton is in a committed location, time passing is not possible, and the next change of the state must involve an outgoing edge of at least one of the committed locations.

Each edge may have a guard, a synchronization and an assignment. Guard is a side-effect free expression that evaluates to a boolean. The guard must be satisfied when the edge is fired. It can contain not only clocks, constants and logical and comparison operators but also integer and boolean variables and (side-effect free) calls of user defined functions. Synchronization label is of the form *Expr*! or *Expr*? where *Expr* evaluates to a channel. An edge with c! synchronizes with another edge (of another automaton in the network) with label c?. Both edges have to satisfy all firing conditions before synchronization. Sometimes we say that automaton firing an edge labeled by c! sends a message c to the automaton firing an edge labeled by c?. There are urgent channels as well (synchronization through such a channel have to be done in the same time instant when it is enabled) and broadcast channels (any number of c? labeled edges are synchronized with one c! labeled edge). An assignment is a comma separated list of expressions with a side-effect. It is used to reset clocks and set values of variables.

Uppaal has some other useful features. Templates are automata with parameters. These parameters are substituted with given arguments in the process declaration. This enables easy construction of several alike automata. Moreover, we can use bounded integer variables (with defined minimal and maximal value), arrays and user defined functions. These are defined in declaration sections. There is one global declaration section where channels, constants, user data types etc. are specified. Each automaton template has own declaration section, where local clocks, variables and functions are specified. And finally, there is a system declaration section, where global variables are declared and automata are created using templates.

Uppaal's query language for requirement specification is based on CTL (Computational Tree Logic, [5]). We do not present any formulas in this paper hence details about query laguage are omitted due to space restrictions.

The simulation and formal verification are possible in Uppaal. The simulation can be random or user assisted. It is more suitable for the user of the tool to see if a model is behaving like he want and like it corresponds to the real system. Formal verification should confirm that the system has desired properties expressed using the query language. There are many options and settings for verification algorithm in Uppaal. For example we can change representation of reachable states in memory or the order of search in the state space (breadth first, depth first, random depth first search). Some of the options lead to less memory consumption, some of them speed up the verification. But improvement in one of these two characteristic leads to a degradation of the other usually. For more exact definitions of modeling and query languages and verification possibilities of Uppaal see [3].

3 General comments and assumptions

In real-time database systems we consider usually transaction processing. Each transaction incoming to a system is assigned a priority. Resources are than apportioned according to priorities to transactions that are processed concurrently. The number of concurrently processed transactions in system is usually bounded – it is controlled by overload management policy. Incoming transactions have usually a deadline. This can be hard (transaction exceeding deadline becomes nearly useless and can be aborted for the sake of other transactions meeting their deadlines) or soft (the value of transaction exceeding deadline decreases, the priority can be lowered and transaction is processed in the time when there are not any transaction possibly meeting deadlines). In this paper we consider hard deadlines.

Scheduling and computation time assignment is strongly connected with priorities. Hence we will discuss this also in the following sections. Other aspects as, e.g., concurrency control will be omitted in order that the suggested models are still manageable by Uppaal.

There are many possibilities how to model transaction arrival. For example, there can be special automaton serving as generator of transactions. The models described in this paper are designed for comparison of two different algorithms. Hence we have decided to define incoming transactions statically as an array `inc_trans` which elements are structures of `release_time` (representing incoming time of transaction since beginning), `deadline_time`(deadline since beginning), `operations` (number of database operations), `received_time` (initially equals zero, it represents computation time already used by this transaction). For simplicity we do not consider exact database records and we even consider that all database operations need the same computational time given by constant `OP_TIME`. Both model can be simulated over this array to compare them (number of aborted transactions etc.) We can also automatically check queries expressed

as formulae of temporal logic and become some other useful informations about modeled algorithms.

4 First Come First Serve

This policy assigns the highest priority to the transaction with the earliest release time. Often, release time equals arrival time. This algorithm is not very suitable for real-time database systems because it does not make use of deadline information. It can give more computation time to older transaction instead a newer transaction with more urgent deadline.

We consider that all computation time is assigned to the oldest transaction in a system until it finishes or reaches its deadline. Hence we will need just one copy of automaton depicted on Figure 1 atop. This automaton represents successively all transaction processed by a modeled system.

The state `Inactive` represents situation when there is not any processed transaction. Constant `TRANSACTIONS` contains the overall number of transactions defined in the input array, variable `act_trans` counts processed transactions. Clock variable `time` represents time from the beginning while clock variable `op_time` measures the time of performance of one database operation. If actual transaction ends successfully (number of performed operations `op_done` reaches the number of operations specified for this transaction), the automaton gets through the state `Done` to `Inactive` and it is prepared for representation of next transaction. If the transaction reaches its deadline before successful finish, the abort is modeled by the state `Abort`, it is counted and the automaton goes to the state `Inactive` once again.

The state `Waiting` is intended for the situation when there are more transactions in the input sequence but release time (representing time of arrival in the real system) is not passed yet. If all transactions from the input sequence are processed, automaton goes to the state `End` and a run is deadlocked. It is the only possible deadlock situation of this model (this has been checked using verification possibility of Uppaal).

5 Earliest Deadline

Earliest Deadline is algorithm which gives the highest priority to transaction with the earliest deadline. A disadvantage of this policy is that it can give high priority and hence a big amount of resources to a transaction which is about to miss its deadline anyway.

Assigned priorities can be used in several different ways for distribution of resources. One way is that a transaction with the highest priority gets all resources until it finishes or exceeds its deadline and it is aborted. To show some other general modeling possibilities, we have chosen some other way. We consider several concurrently processed transactions and scheduler distributes computation time between all of them. Transactions with higher priority get more time but no transaction is skipped. All automata representing concurrent transactions are

instances of the same template depicted on Figure 1 down. The number of those instances can be set by a constant `PAR_TRANS`.

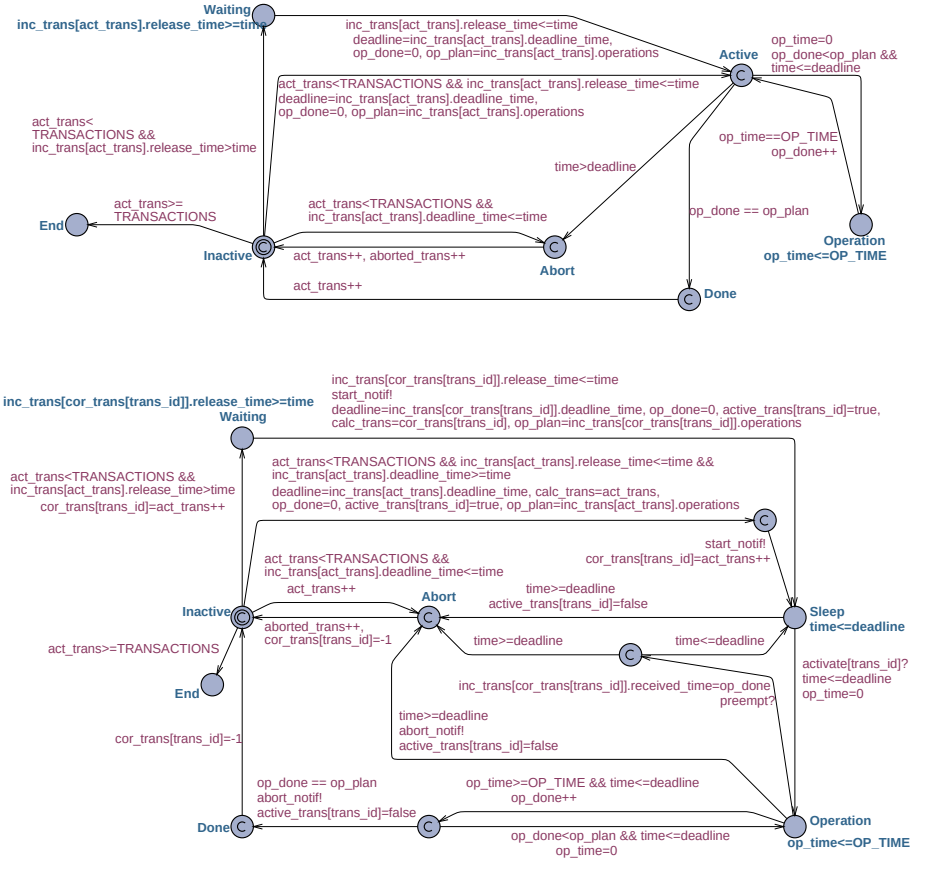

Fig. 1. Transaction automata for FCFS algorithm (atop) and Earliest Deadline algorithm (down)

Distribution of computation time is controlled by Scheduler Automaton depicted on Figure 2 atop. Priorities to transactions are assigned using Priority Assignment Automaton depicted on Figure 2 down.

The basic behavior of Transaction automata is the same as in the case of First Come First Serve algorithm. The main modification is the state `Sleep` and its adjacent edges. It represents the situation when this transaction is processed but actually has not assigned resources. An indication of assigned resources is received from Scheduler automaton through the channel `activate[trans_id]`. `trans_id` is unique identifier of each Transaction automaton and `activate` is

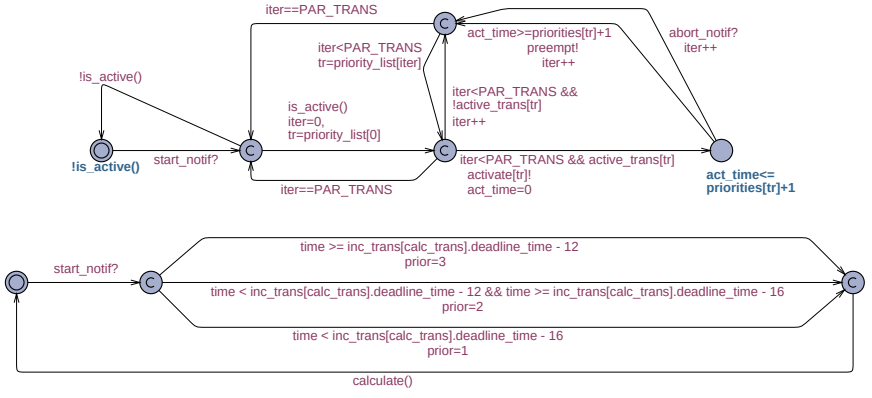

Fig. 2. Earliest Deadline algorithm - Scheduler automaton (atop) and Priority Assignment automaton (down)

array of channels. Taking the resources away is announced through the channel `preempt`. There are two more channels – `abort_notif` informs Scheduler automaton that this transaction is aborted and resources are free and `start_notif` is broadcast channel that informs Scheduler about active transaction and simultaneously asks Priority Assignment to compute priority for new transaction (identification of this transaction is shared using global variable `calc_trans`).

There are two axillary arrays. `active_trans` contains for each Transaction automaton a flag if it actually represents some transaction, `cor_trans` contains for each Transaction automaton identification of actually represented transaction from input array.

Scheduler automaton waits in the initial state until it is notified about new transaction. Then it takes iteratively identifications of Transaction automata representing transaction according the priority from the array `priority_list` which is maintained sorted by Priority Assignment automaton. Each transaction automaton is activated for the time corresponding to its priority (this can be chosen in different ways, in this paper it is directly priority, just for technical reasons increased by one). After all active Transaction automata take a turn, the whole process is repeated again from the automaton representing transaction with the highest priority. Function `is_active` is axillary, it returns true if there is at least one `true` in the array `active_trans`.

Priority Assignment automaton assigns priorities $1, 2$ or 3. The highest priority goes to transactions which have at most 12 time units until deadline, priority 2 to transactions with 12 to 16 time units before deadline and priority 1 to all other. Those values were chosen without any real meaning. It should be clear how to add more values of priority and change intervals for them. Function `calculate()` sorts identifications of Transaction automata in the array `priority_list` according to priorities of transactions they represent.

6 Conclusion

In the previous sections, several timed automata were shown. They form models of two variants of algorithms for priority assignment used in (real-time) database management systems. Of course, this were not the only possible models. The purpose was to show that some important aspects of the real-time database system, such as a priority assignment, can be modeled using such a relatively simple model as nets of timed automata are. The models can be extended in many different ways to capture more behavior of those policies and thus allow many properties to be described as a formula in the logic of Uppaal and then checked using its verification algorithms. Some properties even can not be expressed using Uppaal's modification of CTL. Automata can be modified to bypass this imperfection, but it can demand a special modification for each query and it also can increase reachable state space. Another possible solution to this problem is to try some other verification tool with other query language which can be our future work.

References

1. Abbott, R. K., Garcia-Molina, H.: Scheduling real-time transactions: a performance evaluation. ACM Transactions on Database Systems (TODS), Volume 17 , Issue 3, pages 513 – 560, ACM, September 1992.
2. Alur, R., Dill, D.L.: Automata for modeling real-time systems. Proc. of Int. Colloquium on Algorithms, Languages, and Programming, volume 443 of LNCS, pages 322-335, 1990.
3. Behrmann, G., David, A., Larsen, K. G.: A Tutorial on Uppaal. Available online at http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf (March 15, 2010)
4. Berard, B., Bidoit, M., Petit, A., Laroussinie, F., Petrucci, L., Schnoebelen, P.: Systems and Software Verification, Model-Checking Techniques and Tools. ISBN 978-3540415237, Springer, 2001.
5. Henzinger, T.A.: Symbolic model checking for real-time systems. Information and computation, 111:193-244, 1994.
6. Kao, B., Garcia-Molina, H.: An Overview of Real-Time Database Systems. Advances in Real-Time Systems, pages 463-486, Prentice-Hall, Inc., 1995.
7. Kot, M.: Modeling selected real-time database concurrency control protocols in Uppaal. Innovations in Systems and Software Engineering, Volume 5, Number 2, pages 129-138, ISSN 1614-5046 (Print), ISSN 1614-5054 (Online), Springer, June 2009.
8. Kot, M.: Modeling Real-Time Database Concurrency Control Protocol Two-Phase-Locking in Uppaal. Proceedings of the International Multiconference on Computer Science and Information Technology, Volume 3, pages 673-678, ISBN 978-83-60810-14-9, ISSN 1896-7094, IEEE Computer Society Press, 2008.
9. McMillan, K. L.: Symbolic Model Checking. ISBN 978-0792393801, Springer, 1993.
10. Nyström, D., Nolin, M., Tesanovic, A., Norström, Ch., Hansson, J.: Pessimistic Concurrency-Control and Versioning to Support Database Pointers in Real-Time Databases. Proc. of the 16^{th} Euromicro Conference on Real-Time Systems, pages 261-270, IEEE Computer Society, 2004.
11. ParaDiSe (Parallel & Distributed Systems Laboratory): Yahoda verification tools database. Available on-line at http://anna.fi.muni.cz/yahoda/ (March 15, 2010)