Notes on Modeling of Real-Time Database System V4DB in Verification Tool Uppaal

Martin Kot*

Centre for Applied Cybernetics, Dept. of Computer Science,
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 subject of an intensive research recent time. An experimental RTDBMS called V4DB has been created at the Department of measurement and control of Technical university of Ostrava and is still in further development. Now we want to use some formal verification techniques to verify used algorithms, especially concurrency control ones. In this paper I suggest some ideas, how to model concurrency control algorithms of V4DB using Uppaal's nets of timed automata originally supposed to model real-time reactive systems, not database ones. More precisely a simple real-time database system with optimistic concurrency control protocol called Sacrifice will be modeled.

Keywords: real-time database system, V4DB, timed automaton, modeling, verification tool, Uppaal

1 Introduction

Many applications working with a database management system (DBMS) require response in bounded time today. Traditional DBMS are not able to guarantee such bounds on response time. This is the reason why so-called real-time database management systems (RTDBMS) emerged.

Research in RTDBMS focused on evolution of transaction processing algorithms, priority assignment strategies and concurrency control techniques. But the research was based especially on simulation studies with many parameters defined. Hence at the Department of measurement and control of Technical university of Ostrava, Václav Król, Jindřich Černohorský and Jan Pokorný decided to design and implement an experimental real-time database system called V4DB suitable for study of real time transaction processing. The system is still in further development but some important results were obtained already.

Now we have decided to try to use some verification techniques on important parts and algorithms. To our best knowledge nobody has tried to verify real-time database algorithms using some verification tool yet. I have chosen the verification tool Uppaal. The reason is that this tool is designed for real-time systems.

* Supported by the Ministry of Education of the Czech Republic, Project 1M0567.

But it is supposed to be used on so-called reactive systems, not database ones. So I need to solve the problem of modeling data stored in the database or somehow do without such modeled data. Then I would like to check some important properties of V4DB and used algorithms. For example absence of deadlock when using an algorithm which should avoid deadlock in transaction processing, processing transaction with bigger priority instead of ones with smaller priority and so on.

In this paper I would like to aim at modeling of some parts of V4DB in Uppaal, not at verification itself. In the section 2 the RTDBMS V4DB will be described more closely. The section 3 is devoted to the verification tool Uppaal. And in the section 4 I will show some possibilities of modeling real-time database optimistic concurrency control protocol called Sacrifice without model of stored data.

2 Real-time database system V4DB

The main goal of the V4DB project ([4, 3]) was to design and implement real-time database system suitable for study of real-time transaction processing. Transaction processing is general concept in the area of databases. Its purpose is to ensure integrity of the system when changing from one consistent state into the other. Each transaction contains several database operations and only performing all operations of the transaction ensures consistency of the system.

The system V4DB is implemented as an integrated set of the most important functional parts of a veritable real-time database system. It enables testing and performance analysis of different algorithms for particular functional parts to understand the effect on system performance.

V4DB works under a real-time operating system VxWorks. The database is stored in memory to eliminate the influence of accesses to hard disk on the results of tests and analysis. The system consists of several parts working in parallel. The behavior of those parts could be changed easily by setting before start. Main executive parts are transaction generator, predispatcher, dispatcher and transaction execution including concurrency control. Those parts use data dictionary and database.

To study the database transaction processing, transactions should have known properties that can be set in advance. Hence, they are generated by an internal generator. There are two different generators - periodic and random. Periodic generator creates transactions with predefined period, random generator has defined minimal and maximal intervals between transactions. Generated transactions contain the following database operations: select, update, insert, delete.

From the generator, the transaction is passed to a predispatcher. This process avoids system overloading and creates the structure fully describing the transaction. This structure is used by all other functional blocks. The dispatcher then extracts the transaction parameters and dispatches transactions for execution according the priority assignment policy.

The transaction scheduled for an execution is parsed into particular commands and then the commands are processed by the command executor. To obtain reasonable performance, multiple transactions must be able to access data concurrently. Hence there is concurrency control component that synchronizes access to the stored data. There are many different concurrency control protocols. They divide into two basic groups - optimistic and pessimistic. Optimistic protocols are based on validation. All operations of a transaction are done and than the validation phase comes. If a conflict is detected the transaction may be restarted and changes are taken back. Pessimistic protocols use locks. At the beginning of transaction execution all respective parts of the database are locked and no other transaction has access to those parts. Then all operations are performed and at the end all locks are removed.

The database of V4DB is very simple. There are several tables. Each table has given number of records and each record is one value of given size.

3 Verification tool Uppaal

Uppaal ([1, 2]) is verification tool for real-time systems 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 time automaton is a finite-state automaton extended with clock variables. A dense-time model is used where clock variables have real number values and all clocks progress synchronously. In Uppaal, several such automata in parallel compose network of timed automata and communicate using channels and 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) and firing an edge of some automaton (possibly synchronizing with another automaton).

Automaton has locations and edges. Location has optional name and invariant. 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. 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. Synchronization label is in 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. Assignment is a comma separated list of expressions with a side-effect. It is used to reset clocks and set variable values.

On the Figure 1 is shown how the described notions are represented graphically in Uppaal. There are 3 locations named `A`, `B` and `C`. Location `A` is initial and `B` is committed. Moreover `A` has an invariant `x<=15` with the meaning that the automaton could be in this location only when the value of the clock variable `x` is less or equal 15. The edge between `A` and `B` has the guard `x>=5 && y==0`. It can be fired only when the value of the clock variable `x` is greater or equal 5 and the integer variable `y` has the value 0. The edge has also synchronization label `synchr!` and an assignment `x=0, y=1` reseting the clock variable `x` and setting the value 1 to the integer variable `y`.

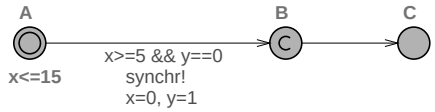

Fig. 1. Graphical representation of a timed automaton in Uppaal

Uppaal has some other useful features. Templates are automata with parameters. These parameters are substituted for a given argument in the process declaration. This enables easy construction of several alike automata. Besides 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 a simplified version of CTL. It enables to express common properties such as reachability, safety and liveness. This paper is focused on modeling hence I will not describe this language in detail.

4 Model of a database management system in Uppaal

In this section we will show some ideas on modeling real-time database management system in Uppaal. The model presented here consists of several timed automata. It represents important parts of V4DB with an optimistic protocol called *Sacrifice* chosen for concurrency control. Using this protocol, all operations of the transaction are executed. If a conflict is detected in validation phase and validating transaction has smaller priority, it is restarted. All changes of data should be taken back but not in V4DB because data are generated randomly and concrete values do not play any role.

First automaton on the Figure 2 represents random generator. The constants `MIN_INTERVAL` and `MAX_INTERVAL` determine minimal and maximal time

between two consecutive transactions. In the case that both constants have the same value, the automaton represents a periodic generator. Generated transaction should have given operations. But as they are chosen in random, we can afford in the model to choose particular operations randomly at execution. Hence, the generation of the transaction is only represented by synchronization with automaton representing dispatcher through the channel called `generate`.

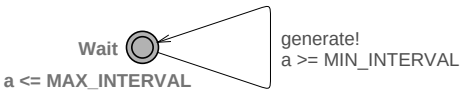

Fig. 2. Generator automaton

In V4DB, there is a maximal count of active transactions. Predispatcher avoids overload of the system so that transactions beyond the limit are put into buffer and executed when the execution of some other transactions terminates. Hence we can model each active transaction as one automaton. We use the template shown on Figure 3. In system declaration several copies of this automaton are created, each with an unique integer identification value stored in the variable `id`.

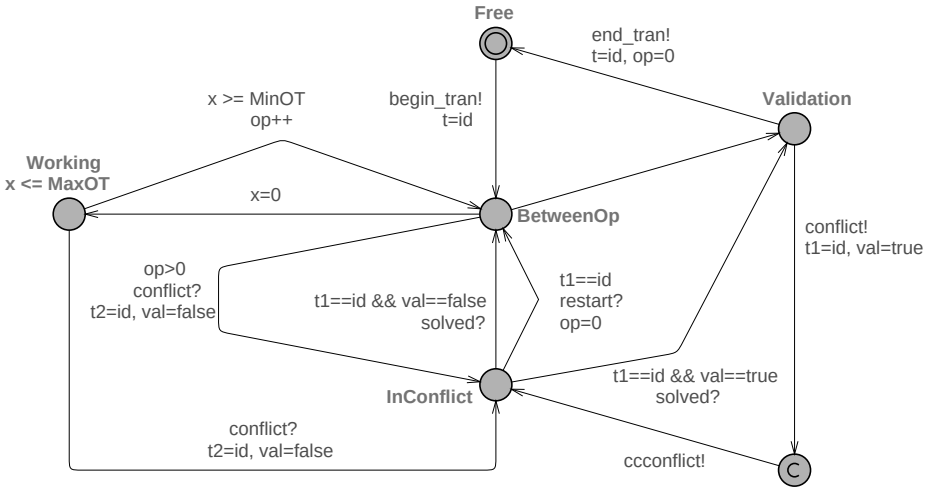

Fig. 3. Transaction automaton

An automaton representing transaction starts in the state `Free`. The edge to the state `BetweenOp` synchronizes with dispatcher automaton. This can ensure that only when some transaction is waiting for an execution the transaction automaton gets active. The assignment `t=id` passes identification of the automaton to dispatcher using the integer variable `t`.

The state `Working` represents execution of one database operation. We could have more states for different operations but from our actual point of view all operations are similar. We use constants `MinOT` and `MaxOT` as bounds for execution time of an operation. The integer variable `op` is used as a counter of executed operations.

After any number of executed operations the edge to the location `Validation` may be used. This corresponds to a fact that any number of operations could be generated into the transaction and after the execution of all of them validation phase comes. Database records accessed by operations of a transaction are chosen randomly by the generator. Hence it is in fact random situation that two transactions access the same record and a conflict occurs. In our automaton we have two edges to represent both situations. The edge to `Free` means that validation was successful. Synchronization through channel `end_tran` with dispatcher automaton is used to notify dispatcher of the fact that the transaction has terminated. The automaton may be used to represent next transaction.

The other edge from `Validation` represents conflict situation. Conflict occurs only when some other, not yet validated, transaction accessed the same record. Hence in our model there has to be some transaction automaton in the location `Working` or `BetweenOp` with nonzero count of executed operations. It is ensured using synchronization through the channel `conflict`. The validating transaction automaton passes its identification to concurrency control manager (CCMan) automaton using the variable `t1` and sets boolean variable `val` to true. The conflicting transaction passes its identification using the variable `t2` and sets boolean variable `val` to false. Only one synchronization is permitted for an edge and we need not only to synchronize two transaction automata but as well CCMan automaton. Therefore a committed location is used and the edge synchronizing with CCMan through the channel `ccconflict` is fired without any time delay.

CCMan sends its responses through channels `solved` and `restart`. In the variable `t1` is identification of the automaton for which the response is intended. Synchronization through `solved` means that conflict was successfully solved and transaction can continue validation or execution. Synchronization through `restart` means that transaction will be restarted. In this case `op` is reseted because all operations should be executed again.

Dispatcher automaton depicted on Figure 4 is quite simple. The edge with synchronization label `generate?` synchronizes with generator automaton. The variable `waiting` counts transactions in buffer waiting for an execution.

The edge with the label `begin_tran?` synchronizes with some transaction automaton. In the variable `trans_id` it gets transaction identifier. The function `register()` saves transaction identifier into an array and increases the value of

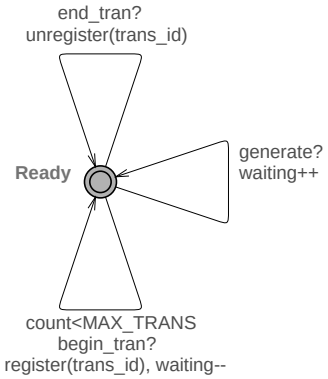

Fig. 4. Dispatcher automaton

the variable `count`. The number of waiting transactions is decremented. The edge with the label `end_tran?` synchronizes with transaction automaton too. It means that the transaction has terminated. Hence its identification is removed from the array and the variable `count` is decremented using the function `unregister`.

Last automaton of our simple model is concurrency control manager depicted on the Figure 5. We model sacrifice protocol, hence transaction with smaller priority is sacrificed (restarted) when conflict occurs. In V4DB, priorities are randomly set by generator. In model, we could choose priority randomly when transaction automaton begins simulation of the execution. But particular automaton is chosen nondeterministically and it has unique identification number. Hence we can consider identification to be the priority. Bigger identification number means bigger priority.

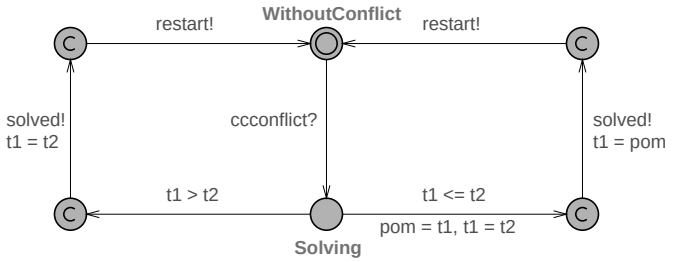

Fig. 5. Concurrency control manager automaton

CCMan automaton synchronizes with validating transaction through the channel `ccconflict`. When this synchronization occurs, in the variable `t1` is identification of validating transaction and in `t2` is identification of conflict-

ing transaction. CCMan sets the bigger of those identification to `t1` and fires `solved!` synchronization and then sets the smaller identification to `t1` and fires `restart!` synchronization. Hence one transaction is restarted and one continues in the execution. All three edges between locations `Solving` and `WithoutConflict` are fired without any time delay because of committed locations.

5 Conclusion

In the previous section, several timed automata were shown. They form a model of a simple real-time database management system inspired by V4DB with optimistic concurrency control protocol sacrifice. The purpose was to show that some important aspects of the database system such as concurrency control can be modeled without modeling stored data.

I have prepared more models of the V4DB. They capture different concurrency control protocols as well as different priority assignment algorithms. Now I am identifying crucial properties that could be expressed using query language of Uppaal and verified by this tool. This will be my future work in this area. The purpose is to identify problems of used algorithms and protocols and possibly suggest some improvements for V4DB.

References

1. 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 (September 7, 2007)
2. David, A., Amnell, T.: Uppaal2k: Small Tutorial. Available on-line at http://www.it.uu.se/research/group/darts/uppaal/tutorial.ps (September 7, 2007)
3. Król, V.: Metody ověřování vlastností real-time databázového systému s použitím jeho experimentálního modelu. Disertation thesis. VSB - Technical university of Ostrava, 2006.
4. Król, V., Pokorný, J., Černohorský, J.: The V4DB project - support platform for testing the algorithms used in real-time databases. WSEAS Transactions on Information Science & Applications, Issue 10, Volume 3, October 2006.