Modeling Real-time Transactions in UPPAAL

Simin Cai
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
simin.cai@mdh.se

Abstract

During the development of an Real-time Database Management System (RTDBMS) one needs to trade-off between data consistency and timeliness. To achieve a systematic method for such trade-off, we must find a way to model the behaviors and properties of real-time transactions. In this report, we experiment with the modeling of transactions and verification of transaction properties in UPPAAL. We build a model for an exemplary transaction with the optimistic concurrency control mechanisms, and model-check the timeliness property.

1 Introduction

Traditionally, Database Management Systems (DBMSs) focus on maintaining data consistency, by ensuring the so-called ACID properties (Atomicity, Consistency, Isolation and Durability) during transaction execution [1]. A Real-Time Database Management System (RTDBMS), however, may need to relax the assurance of ACID properties, since the timely response of transactions is more important in a real-time system [2]. The RTDBMS designer must find an appropriate trade-off between the ACID and timeliness, and select and implement the appropriate run-time transaction management mechanisms. During the development of RTDBMS, however, there exists no systematic method to trade-off the ACID properties and the timeliness.

In the DAGGERS project we aim at introducing a process for developing a tailored RTDBMS that guarantees timeliness and desired data consistency for real-time systems by employing model-checking techniques during the process. As an initial step, we need to model the transactions together with the run-time transaction management mechanisms, and verify them against the desired ACID and timeliness properties.

In this report we explore the modeling of transaction behaviors in timed automata and verification of transaction properties in Timed Computation Tree Logic (TCTL), both implemented by the UPPAAL tool[3]. We identify the components of a transaction that need to be modeled, and experiment with a exemplary transaction and a concurrency control mechanism. Finally, we discuss the next step of our work.

2 Transactions and Transactional Properties

In a database, a transaction consists of a set of logically related operations on the data and ensures (a certain level of) data consistency. The collection of operations, including read, write and calculation on the data, is also called a work unit. Typically, a work unit reads some data from the database, performs some calculation, and writes the result into the database.

2.1 ACID Properties

Data consistency is usually achieved by ensuring the ACID properties, i.e., atomicity, consistency, isolation and durability [1]. Atomicity requires that a transaction either finishes all its operations completely, or does not make any changes at all. Consistency requires that a transaction executing by itself must not violate any logical constraints. Isolation requires that uncommitted changes of one transaction should not be seen by concurrent transactions. Durability requires that all committed changes must be made permanent, even after system failure.

In order to verify these properties, one possible way is to model the transaction behavior and verify the conformity of these properties when they should be met during transaction execution.

2.2 Timeliness

In a real-time system, each work unit usually has a set of real-time requirements in addition to data consistency requirements. More specifically, a work unit may be invoked in a particular pattern regarding to time, and are expected to complete its work before a deadline. When a work unit is encapsulated as a transaction, the transaction also inherits these real-time properties. A transaction may have a period (or Minimal INter-arrival Time (MINT)), and has a relative deadline. For a hard real-time system, the timeliness of a transaction requires that the transaction must completes before its deadline.

In order to verify the timeliness, we must model the time progress in the system as each operation in the transaction takes place.

2.3 Transaction Management Mechanisms

To ensure the desired ACID properties, a set of run-time transaction management mechanisms, such as concurrency control and recovery, are implemented in the DBMS and will be "plugged" into the work unit in order to achieve the transactional behavior. Therefore, extra steps are added in the transaction, before or after the work unit, or even in between the operations.

For example, to ensure atomicity, all operations, or a subset of operations, may be treated as a whole. The changes made by these operation may not be written into the database immediately after the write operation. These changes are written into the database as a whole, or not written into the database at all. This is achieved by different commit/abort/rollback mechanisms. Taking a flat transaction model as an example, which assumes full ACID assurance. Full atomicity is guaranteed by embracing the whole work unit within one begin/commit pair. The work unit starts its work after the begin command, but the actual changes towards the database are made after the commit command. If any failure occurs before commit, no changes will be made at all. In advanced transaction models, to support different variants of atomicity more mechanisms are adopted, for example, partial commit, checkpoint, selective rollback, etc.

In order to ensure isolation different concurrency control mechanisms can be applied. For example, using a lock-based concurrency control mechanism, acquiring and releasing locks are added to the transaction before or after particular operations of the work unit. Using a validation-based concurrency control, a validation phase is added before the work done by the work unit is committed.

For durability, changes made by the work unit are written into persistent storage after transaction commit. Logging and recovery mechanisms are often implemented to ensure durability even when failure occurs. Before transaction commit, the operations of the work unit that change the database are written into a log. When the system is restarted from a failure, the DBMS can recover all committed changes based on the log. These also add extra steps to the transaction.

Therefore, a transaction not only consists of the data access and manipulation operations of the work unit, which are decided at design time of the application to perform a logic; but also operations imposed by the transaction management mechanisms of the run-time platform (DBMS), which are intended for guarantee ACID properties.

To model the behavior of a transaction, we need to, first of all, model the basic work unit. We also need to model the run-time mechanisms and weave them into the work unit model.

3 Model-checking Transactions in UPPAAL

In this section, we briefly describe model-checking particular transactions against the desired transaction properties in UPPAAL [3], which is the state-of-the-art model-checker for real-time systems.

3.1 Modeling Transaction Behavior

We model transactional behavior as networks of timed automata [4] in UPPAAL. To model a real-time transaction's behavior we need to model three parts: the work unit, the run-time mechanisms, and the time.

In the following we use a transaction TR_0 guaranteeing full isolation with Optimistic Concurrency Control (OCC) [5] to illustrate the modeling of transaction behavior in UPPAAL. We assume that the transaction reads data D_0, performs some calculation, and writes the result back to D_0. The transaction is invoked continuously with a minimal interval, and must complete before a relative deadline. Transaction TR_0 is modeled as a template, as shown in Fig 1.

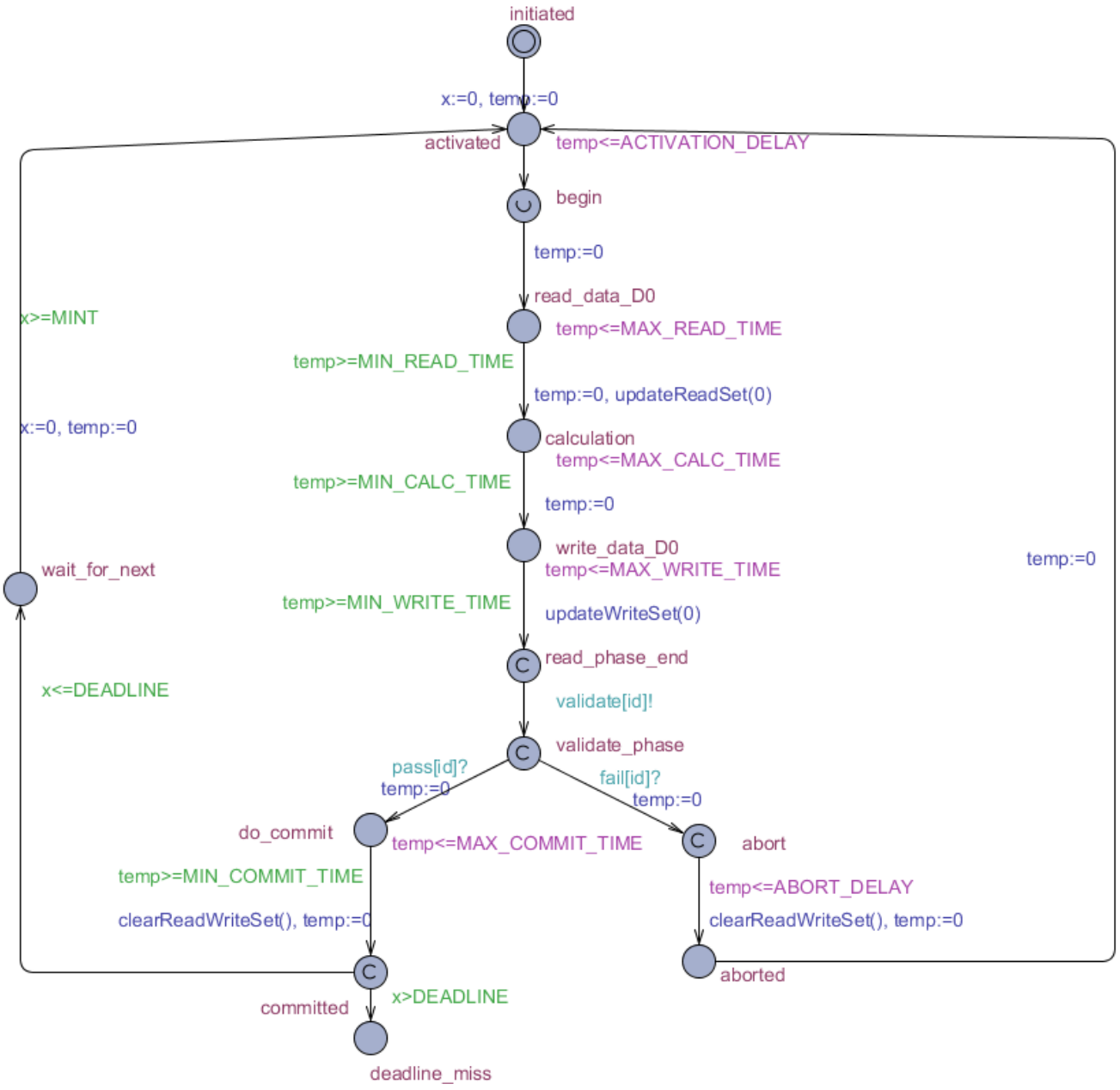

Figure 1. The UPPAAL model of transaction TR_0 using optimistic concurrency control

3.1.1 Modeling the Work Unit

A work unit, consisting of a set of read, write and calculation operations, can be modeled as a set of states, each of which corresponds to an operation. The edges represent the order of these operations. The work unit of TR_0 consists of three operations: reading D_0, calculation and writing D_0. They are modeled as states read_data_0, calculation and write_data_0, respectively.

3.1.2 Modeling the Run-time Mechanism

The run-time mechanism may require additional operations before, after or in between work unit operations. These operations can be modeled as individual states connected to work unit operations, or as side-effect free functions in update labels between

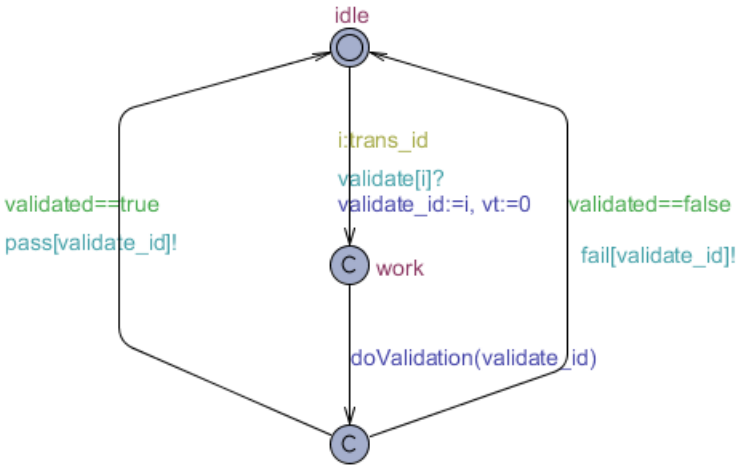

Figure 2. The UPPAAL model of the transaction manager

operations.

According to OCC, the data involved in a transaction are written into a read set and a write set when the work unit operations are performed. In Fig 1, functions updateReadSet() and updateWriteSet() are called in the update labels after the corresponding work unit operations. The transaction is validated before it is allowed to commit. If a transaction fails the validation, it will be aborted and restarted. These operations are modeled as states in In Fig 1.

A transaction manager is involved in many run-time mechanisms that provides services such as resource allocation and validation. The transaction manager can also be modeled as a template in UPPAAL, synchronizing with transactions through channels. In OCC, the validation is done by the transaction manager when a transaction steps into the validation phase. The transaction manager in our example is shown in Fig 2. It receives a channel signal from a transaction, performs validation, and replies to the transaction with a pass or fail signal.

3.1.3 Modeling the Time

Since UPPAAL internally models time using clocks, the real-time properties can be easily models as invariants or guards. The real-time properties include the execution times of operations, the period or MINT, and the deadline of the transaction.

We assume each individual operation has a bounded execution time. Each operation has a worst-case execution time, denoted as MAX_OPERATION_TIME, and a best-case execution time, denoted as MIN_OPERATION_TIME. In Fig 1, we use an invariant with MAX_OPERATION_TIME to model the behavior that an operation "at most" takes such amount of time, and a guard with MIN_OPERATION_TIME to model that an operation "at least" takes such amount of time. Similarly, we model the behavior of sporadic invocation using the guard together with the MINT. The total execution time of the transaction is modeled by a clock, x, which records time progress during the states transition.

3.2 Modeling and Verification of Transaction Properties

The transaction properties to be modeled and verified include both timeliness and the ACID properties. We use Timed Computation Tree Logic (TCTL) to verify these properties, which is integrated in the UPPAAL tool [3].

3.2.1 Timeliness

In order to model the timeliness property, we create a state representing a violation of timeliness, which is reached if the transaction misses its deadline when it completes. As shown in Fig 1, the deadline_miss state is such a state. The verification of timeliness is then transformed into a safety check that deadline_miss should never be reached. The TCTL query for verifying timeliness is: *A[] not TR0.deadline_miss*. If the verification fails, a counter-example is provided by the tool that exposes an execution path that leads to the violation of timeliness.

3.2.2 ACID properties

Similar to timeliness, the violation of ACID properties should be modeled as states. If any of such states is reachable, the corresponding property is violated. Modeling and verification of ACID properties are discussed further in this report, but they will be included in the next step of our work.

4 Related Work

Several existing works have modeled particular aspects of transaction or transaction management mechanisms in UPPAAL. Kot [6] modeled several priority assignment policies for transactions in an RTDBMS. Al-Bataineh et. al [7] modeled a two-phase commit protocol for real-time transactions in UPPAAL and verified the behavior of transactions using TCTL. The closest work to ours is Kot [8], which modeled several concurrency control protocols for an RTDBMS. These works all focus on particular protocols and show the possibility of model-checking with UPPAAL. None of them explicitly model and verify the timeliness and ACID properties.

5 Conclusion and Future Directions

In this report we explored how to model transactions and model-checking the desired timeliness and ACID properties in UPPAAL. We analyzed the structure of a transaction and pointed out the elements that need to be modeled. We also build an experimental model in UPPAAL for a transaction with the OCC mechanism, and verified the transaction timeliness property.

This report only presents the initial work on modeling real-time transactions. More work needs to be done in the future. Although we have discussed the possible way to model and verify the ACID properties, we have not done it in our exemplary UPPAAL model. The ACID properties will be included in the next step work and will be verified together with timeliness, in order to find conflicts between ensuring timeliness and data consistency.

In this report we only model one concurrency control mechanism, OCC. However, other concurrency control mechanisms may be significantly different from OCC, and how to model them accurately remains unclear. In addition, other run-time mechanisms such as logging and recovery should also be modeled. In the future we should create a library of common run-time mechanism models, and find a method to weave these models into the work unit model dynamically.

In this report we also assume a flat transaction model, which organizes work unit operations as a sequence and requires full ACID. In our next step we should experiment with various transaction models with various relaxation of ACID.

References

[1] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

[2] J. A. Stankovic, S. H. Son, and J. Hansson, "Misconceptions about real-time databases," *Computer*, vol. 32, no. 6, pp. 29–36, 1999.

[3] K. G. Larsen, P. Pettersson, and Y. Wang, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134–152, 1997.

[4] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[5] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Trans. Database Syst.*, vol. 6, no. 2, pp. 213–226, Jun. 1981.

[6] M. Kot, "Modeling and verification of priority assignment in real-time databases using uppaal." in *DATESO*. Citeseer, 2010, pp. 147–154.

[7] O. Al-Bataineh, T. French, and T. Woodings, "Formal modeling and analysis of a distributed transaction protocol in uppaal," in *Temporal Representation and Reasoning (TIME), 2012 19th International Symposium on*. IEEE, 2012, pp. 65–72.

[8] M. Kot, "Modeling selected real-time database concurrency control protocols in uppaal," *Innovations in Systems and Software Engineering*, vol. 5, no. 2, pp. 129–138, 2009.