Towards the Verification of Temporal Data Consistency in Real-Time Data Management

Simin Cai, Barbara Gallina, Dag Nyström, Cristina Seceleanu
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
{simin.cai, barbara.gallina, dag.nystrom, cristina.seceleanu}@mdh.se

***Abstract*—Many Cyber-Physical Systems (CPSs) require both timeliness of computation and temporal consistency of their data. Therefore, when using real-time databases in a real-time CPS application, the Real-Time Database Management Systems (RTDBMSs) must ensure both transaction timeliness and temporal data consistency. RTDBMSs prevent unwanted interferences of concurrent transactions via concurrency control, which in turn has a significant impact on the timeliness and temporal consistency of data. Therefore it is important to verify, already at early design stages that these properties are not breached by the concurrency control. However, most often such early on guarantees of properties under concurrency control are missing. In this paper we show how to verify transaction timeliness and temporal data consistency using model checking. We model the transaction work units, the data and the concurrency control mechanism as a network of timed automata, and specify the properties in TCTL. The properties are then checked exhaustively and automatically using the UPPAAL model checker.**

I. Introduction

In a Cyber-Physical System (CPS), the control of physical working units is decided by the computational operations based on timely monitored environmental data [1]. Many CPS applications are real-time systems, which means that the results of the computation must be not only logically correct, but also temporally correct [1]. The temporal correctness of a result depends both on the time when the result is produced, and on the temporal consistency of the data used for the computation. For instance, consider a robot arm picking up objects from the conveyor of an assembling line. In order to pick up the object correctly, the robot arm must adjust its rotation angle according to the position of the approaching object. A computer in the arm calculates the rotation angle, based on current angle of the arm, and the position of the target. The computational result is useless, if either the calculation misses its specified deadline, or the position data are outdated.

One common way of managing the temporal environmental data and computational results is to store them in a Real-Time Database (RTDB) [2]. The temporal consistency of the data requires that the states of the RTDB must be consistent with the corresponding environmental states timely [3]. Since computations on data are implemented as transactions in the database, the Real-Time Database Management System (RTDBMS) must therefore ensure both the transaction timeliness and the temporal data consistency [4]. However, it is not trivial to verify these properties, partly due to the concurrency control mechanisms used by RTDBMSs to eliminate unwanted interferences from concurrent transactions. Transactions may be blocked or aborted by the concurrency control manager, which on the one hand may lead to breached timeliness and temporal consistency, and on the other hand increases the complexity of the analysis. Some of existing work towards analysis of temporal consistency (e.g., Song et al. [3]) are based on simulation, and thus lack formal guarantees. Other work either provide analysis of temporal consistency without considering concurrency control [5], or focus on other properties of concurrent transaction systems such as isolation [6] or absence of deadlock [7].

This work is funded by the Knowledge Foundation of Sweden (KK-stiftelsen) within the DAGGERS project.

In our recent work [6], we have proposed a formal approach based on timed automata [8], Temporal Computational Tree Logic (TCTL) [9] and UPPAAL [10] for verifying timeliness and isolation of transactions in a unified manner. Here, we develop our approach further, focusing on the tradeoff between timeliness and temporal data consistency instead. We consider the targeted system as a composition of the following constituents: the data accessed by either the sensors or the computational units of the CPS, the transaction work units [11], which are the logical operations in the transactions, and the concurrency control manager that coordinates concurrent transactions. We first transform these constituent parts into a formal model, which is a network of timed automata. Then we specify the timeliness and temporal consistency in a logic formalism called TCTL, using a set of specification patterns. Finally, we use the UPPAAL model checker [10] to check whether these formalized properties are satisfied by the model. The approach is exemplified on a concrete example in detail.

The remaining part of the paper is organized as follows. Section II introduces the background of the paper, consisting of the concepts of temporal data consistency in RTDBMS, and the needed knowledge on timed automata and UPPAAL. In Section III we present the assumed CPS system with exemplary transactions and relevant requirements. In Section IV we describe our modeling approach for transactions, data and the lock manager of the assumed system. The formal specification of the requirements, as well as the verification results, are presented in Section V. We compare our work to the related work in Section VI, after which we conclude the paper in Section VII.

II. Background

In this section, we first recall the concepts of temporal data consistency in real-time databases, followed by a brief introduction of timed automata and the UPPAAL model checker.

A. Temporal Data Consistency

Data in an RTDBMS can be classified into base data and derived data. In real-time applications, which often monitor the environment states and react accordingly, base data are the representations of the environment states in the database. A typical example of base data is the readings from sensors that monitor the speed of the conveyor in our example CPS. Derived data are the results of computations based on a set of base data objects. For instance, a transaction takes the conveyor speed and the position of the robot arm as inputs to compute the rotation angle. The rotation angle is a derived data. Each data object is associated with a timestamp. For a base data object, the timestamp indicates the time when it is collected, whereas for a derived data object, it refers to the time when it is derived.

As mentioned in Section I, RTDBMSs must guarantee the temporal data consistency, which includes two aspects: the **absolute validity** and the **relative validity** [3] of data. Absolute validity refers to the property that the data must always reflect the environment timely. If we define the age of a data object as the difference between the current time and its timestamp, a base data object is absolute valid if the age of the data is smaller than a specified interval, called **absolute validity interval**. A derived data object is absolute valid if all participating data are absolute valid.

In order to compute a valid derived data, the set of base data may have to be collected close enough to each other in time. For instance, the conveyor speed and the position of the robot arm must be collected within 50 milliseconds. A set of data objects are relative valid, if the difference between the ages of every object is within a specified **relative validity interval**.

The original absolute validity requires data to be absolute valid all the time. This however imposes restrictions on the database performance and the timeliness of other transactions, since the data may need to be updated frequently, even though it is not accessed by any other transaction. Therefore, Kao et al. [12] propose the **weak absolute validity** as a relaxation, which requires that, the age of the data accessed by a transaction should be smaller than its absolute validity interval only when the transaction accesses it. Similarly, one can define the **weak relative validity**, which requires that the age differences of the base data should be within the relative validity interval when they are accessed by a transaction.

B. *Timed Automata and* UPPAAL

UPPAAL [10] is the state-of-art model checker for real-time systems, based on timed automata [8]. Basically, a system is modeled as a network (a parallel composition) of timed automata in UPPAAL. A timed automaton is a finite-state automaton extended with real-valued clock variables and discrete variables. In UPPAAL, clock variables progress synchronously. The locations of all automata, together with the values of clock variables, define the state of a system.

The action to be taken at one location can either be a delay at the same location, or a transition to another location following an edge. An invariant, which is a predicate (boolean set of states) over clock variables, may be associated with a location setting an upper-bound on the delay. A guard, which is a predicate of clock or discrete variables, may be associated with an edge as the required condition to take the underlying transition. During the transition, discrete variables can be updated, while clock variables can be reset. An automaton can synchronize with another automaton via channels. Data can be shared by all automata by shared variables. A location marked as "U" is an "urgent" location, indicating that the next transition (not necessarily from this same location) should be taken without delay. A location marked as "C" is a committed location, indicating that the transitions from this location should be taken immediately.

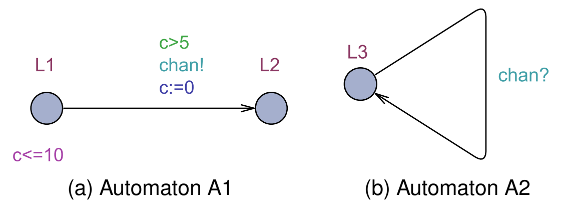

Fig. 1. Example of automata in UPPAAL

Fig. 1 shows two automata, A1 and A2, in UPPAAL notation. A1 has two locations, *L1* and *L2*, and has defined a clock variable c. *L1* has an invariant $c <= 10$, indicating that A1 may delay at *L1* at most until c equals 10 time units. The guard $c > 5$ requires that the value of c must be bigger than 5 in order to take the transition to *L2*. A1 synchronizes with A2 via channel *chan*. The "!" denotes sending the signal, and the "?" denotes receiving the signal. When A1 transits from *L1* to *L2*, it sends a signal via *chan* and resets c. When receiving the signal, A2 takes the transition from location *L3* back to *L3*, in turn.

UPPAAL uses a decidable subset of TCTL (Timed Computational Tree Logic) to formalize requirements that need to be proven as properties of the system by model-checking. These formalized specifications, called queries, can be verified exhaustively on the network of timed automata (e.g., A1 —— A2 in Fig. 1). In this paper we will use the following queries:

- $A\,[\,]\,p$: **Invariant property** (For all possible execution paths p always holds).
- $p \rightarrow q$: **Leads-to property** (Whenever p holds, q will eventually hold).

Property p is a logic expression that may contain logical operators such as "and", "or", "not" or "imply". In case an invariant property fails the verification, the model checker provides a counter-example, and step-by-step simulation of the counter-example. Readers can refer to the literature [10] for more information about UPPAAL.

III. ASSUMED SYSTEM

The assumed system is a CPS including sensors that monitor the environment, and control processes that control the actuators based on the sensor readings. The real-time data are stored in an RTDB. The access and manipulation of the data are managed by an RTDBMS as transactions. We identify the following transaction types in the RTDBMS.

*a) **Update transaction**:* An update transaction is a write-only transaction that updates a real-time data object with the sampled value in the database. It is triggered with a period that is decided by the sampling rate.

*b) **Control transaction**:* A control transaction reads data from the database, performs application-dependent computation based on the data, and may write data into the database. In a real-time application, control transactions are often periodic, or are triggered with a minimum inter-arrival time. They often have specified deadlines to meet.

Such update and control transactions may be executed concurrently. The RTDBMS applies a certain concurrency control mechanism to prevent unwanted interferences.

We consider the following computations in the system. Two update transactions, T0 and T1, update data D0 and D1, respectively. T0 has a period of 7ms, and T1 has a period of 8ms. A control transaction, T2, reads D0 and D1, and does some computation based on the values. T2 has a period of 15ms, and a deadline of 15ms. The worst-case time to read a data is 1ms, while the worst-case time to write is 2ms. In this paper, we assume that a lock-based concurrency control is applied. Before a transaction is able to read from or write to a data object, it needs to acquire the lock of that data. When a transaction commits, it releases the locks it holds so that the locked data become accessible to other transactions. The locking and unlocking are assumed to be instantaneous.

The following temporal consistency and timeliness requirements are specified. Among them, Requirement 1 and 2 refer to absolute validity, Requirement 3 refers to relative validity, and Requirement 4 refers to the transaction timeliness.

- **Requirement 1.1** D0 should never be older than its absolute validity interval, which is 15ms.
- **Requirement 2.1** D1 should never be older than its absolute validity interval, which is 16ms.
- **Requirement 3.1** The difference between the ages of D0 and D1 should never be larger than the relative validity interval, which is 18ms.
- **Requirement 4** T2 should not miss its deadline, which is 15ms.

The weak absolute validity and weak relative validity are specified as follows.

- **Requirement 1.2** D0 should not be older than its absolute validity interval, which is 15ms, when it is accessed by T2.
- **Requirement 2.2** D1 should not be older than its absolute validity interval, which is 16ms, when it is accessed by T2.
- **Requirement 3.2** The difference between the ages of D0 and D1 should not be larger than the relative validity interval, which is 18ms, when accessed by T2.

IV. Modeling Transaction Work Units and Data

In this section, we describe our approach for modeling the work units [11] of the transactions, the data and the transaction manager. The high-level description of the modeling approach is presented in Fig 2. The work units, data and the transaction manager are modeled as timed automata respectively. Similar to our previous work [6], a work unit automaton models the operations within a transaction with respect to timing, as well as the interactions with the data and the transaction manager. A transaction manager automaton models the concurrency control mechanism in this paper. In order to verify temporal data consistency, in this paper we extend the approach with a data automaton that models the updated time and the age of the data. A work unit automaton sends signals to the data automata, when the data is updated by the transaction. The data automaton will then update the timestamps of the data.

The assumed RTDBMS applies a lock-based concurrency control mechanism to manage concurrent transactions. Therefore, the transaction manager is called the lock manager in the rest of the paper. The work unit automata may send signals to require for locks from the lock manager. The lock manager either sends signals to the transactions and grant the locks, or lets the transactions wait if the data are already locked. In the remaining part of this section we will discuss how to model the work units, the data and the lock manager. To simplify the illustration, we omit error handling of the transactions, and assume that the lock manager makes decisions instantaneously.

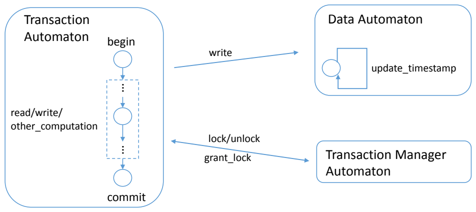

Fig. 2. High-level description of the approach

A. Modeling Transaction Work Units

Similar to our previously proposed approach [6], a transaction is composed by its work-unit operations that include the operations on data and other computation, and transaction management operations, including begin, commit, and synchronizing with the lock manager. We model the work unit, as well as the interactions with the lock manager imposed by the concurrency control, as an automaton in UPPAAL. In such a model, the operations are modeled as a set of locations. A transition from one location to another models the execution order of the operations. Since we especially target temporal data consistency, we explicitly model the interaction between transactions and data. When a transaction updates a data, it sends a corresponding signal to the data automaton via the designated channel.

Fig. 3 shows the timed automaton for the update transaction T0 described in Section III. In this automaton, the transaction starts from the initial location *begin*. A variable *cs* represents the CPU resource. If *cs* is 1, meaning the CPU is taken by another transaction, T0 must wait until CPU is free, which is modeled by the *cs_free* signal. When *cs* is 0, T0 tries to lock the data D0 via channel *lock_data_0[id]*, where *id* is the identifier of this transaction. It then waits until the lock is granted via channel *grant_lock_0[id]*, and proceeds to write the new value of D0 at location *write_d0*. Due to the timing constraints, we use a clock *temp* together with the invariant *WRITE_TIME* to model that it takes in worst case *WRITE_TIME* time units (in this case 2ms) to write the data. After this, the transaction immediately unlocks the data, which is modeled by an urgent location, and a consecutive *unlock_data_0[id]* channel. Then it notifies the observer that D0 is updated, via channel *update[data_id]*. The commit of the transaction is modeled as the *commit_work* location, with an invariant bounded by the commit time. To model the execution time of T0 we create a clock variable *t_u0*, which is reset when T0 is started. The periodic behavior of T0 is modeled such that T0 will be restarted if *t_u0* is equal to T0's period.

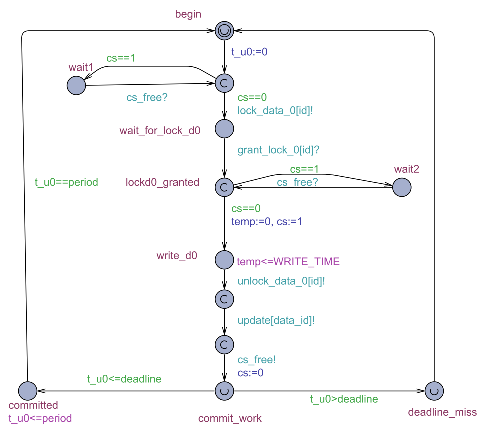

Fig. 3. The work unit automaton of the updater transaction T0

The model of T1 is very similar to the automaton of T0. The channels are defined as locking, unlocking and updating operations on D1 instead of D0, and the values of deadline and period are T1 specific.

The modeling of the control transaction T2 follows the same principles. We show the automaton of T2 in Fig 4. The reading D0 and D1 operations are modeled as locations *read_d0* and *read_d1* respectively, and between the locations channels are used to model the locking mechanism. Computational operations other than read and write are abstracted as a location *other_work*. The *temp* clock, the invariant *temp<=MAX_WORK_TIME* and the guard *temp >= MIN_WORK_TIME* together enforce the best and worst case execution time of the computation. A clock variable *t_tran* keeps track of the time of T2. If *t_tran* is bigger than the deadline, T2 will reach the *deadline_miss* location, indicating a deadline miss.

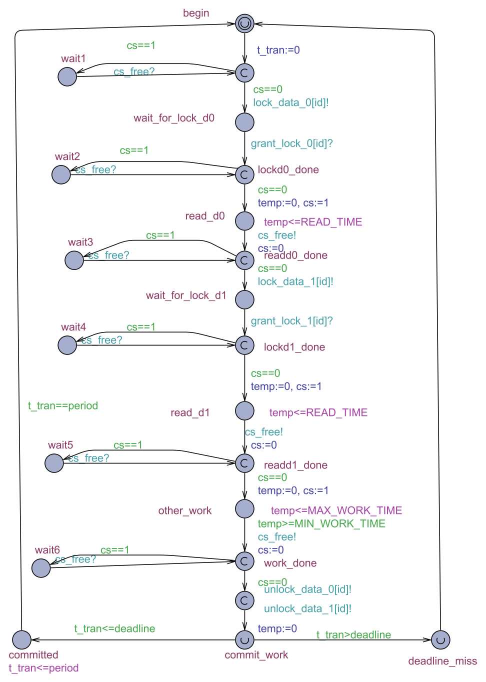

Fig. 4. The work unit automaton of the control transaction T2

B. Modeling the Age of Data

In order to model-check temporal data consistency, we need to model the age of the data. Our solution is to use observer automata that observe the write operations on the data, and update the age of the data accordingly. The observer automaton of D0 is shown in Fig 5. In this figure, the age of data D0 is modeled by a clock variable *age*. When data D0 is updated, the transaction sends a signal to the data automaton via channel *update[0]*, and the observer automaton then resets *age*. Therefore, at any given time point, *age* represents the age of D0.

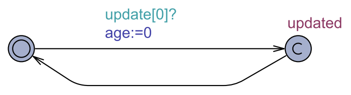

Fig. 5. UPPAAL model of data D0

C. Modeling the Lock Manager

The lock manager handles lock requests and releases from transactions, and grants available locks to the transactions. The modeling of the lock manager is heavily application dependent. In the assumed system, the lock manager grants locks to the transactions in a First-In-First-Serve manner. The lock manager implements a queue that holds the transactions requiring for a lock. New requiring transactions are appended to the end of the queue, while the head of the queue is the one that will be granted the lock.

The model of the lock manager is shown in Fig 6. The queue of waiting transaction ids is modeled by an array, whose first transactions' id is assigned to a variable *head*. An *enqueue()* function appends a new transaction id to the end of the array, while a *dequeue()* function removes the first transaction id from the array, and updates the index and the head of the queue. The *enqueue()* and *dequeue()* functions are shown in Listing 1. When the lock manager gets requests from a transaction with *tran_id* for locking D0 via channel *lock_data_0[tran_id]*, it moves to the *lock_data* location. Meanwhile, during the transition, the transaction is inserted into the queue of D0 by *enqueue()*. The lock manager then checks if the data is currently being locked. If the data is not locked, which means the requiring transaction is the head of the queue, the lock is granted via the channel *grant_lock_0[head]*. If the data is locked, the lock manager just returns to the initial location, and the requiring transaction has to wait. The location *lock_data* is a committed location, indicating the transition sequence from location *idle* to *lock_data* and then back to *idle* is instantaneous and atomic.

When a transaction unlocks data D0 via channel *unlock_data_0[tran_id]*, it is removed from the queue by function *dequeue()*. If the array is not empty, the lock will be granted to the head of the queue via channel *grant_lock_0[head]*.

V. Verification of Temporal Data Consistency and Timeliness

In this section we formulate the requirements of temporal consistency and timeliness as UPPAAL verification queries, and verify these properties for the assumed system.

Listing 1. The functions in the lock manager automaton

```
void enqueue(int tran_id) {
  queue[len]=tran_id;
  len++;
  head=queue[0];}

void dequeue() {
  if(len==1) {
    queue[len]=0;
    len--;
    head=0;
  } else {
    int i;
    for(i=0;i<len;i++)
      queue[i]=queue[i+1];
    queue[len]=0;
    len--;
    head=queue[0]; }}
```

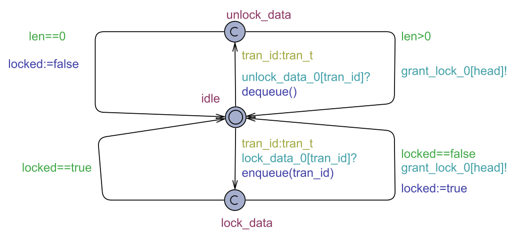

Fig. 6. UPPAAL model of the lock manager

A. Formalizing the Requirements

To model-check the requirements in Section III using UPPAAL, we have to first specify these requirements in TCTL. We propose a set of specification patterns to help the formulation of the temporal consistency and the timeliness of transactions.

The absolute validity requirements, i.e., Requirement 1.1 and 2.1 from Section III, can be specified as: the clock *age* of the automaton of Di must always be smaller than or equal to its absolute validity interval AVI(i). This is a property that must hold invariantly, and can be specified using the $A[]$ operators in TCTL as: $A[]\, Di.age <= AVI(i)$.

The weak absolute validity requirement can be specified as: whenever Tj reads Di, the age of Di must be smaller than or equal to its absolute validity interval AVI(i). This property is specified as $A[]\,(Tj.read\_di\, imply\, Di.age <= AVI(i))$.

The relative validity, as described in Requirement 3, requires the age differences of Di and Dj to be smaller than or equal to the relative validity interval RVI(i,j). Intuitively, this requirement can be specified as $A[]\,(Di.age - Dj.age <= RVI(i,j)\, and\, Dj.age - Di.age <= RVI(i,j))$. However, the verification of this query might not terminate, due to a large state-space during verification. .

In Fig 7 we illustrate the update of the age variables with respect to time, using D0 and D1 as examples. During the period shown in the figure, D0 is updated in t1 and t4, while D1 is updated in t2 and t6. Without loss of generality, we consider the relative validity at t3 and t5. At t3, the values of D0.age and D1.age are t3-t1 and t3-t2 respectively. The difference between the ages is hence t2-t1. This is actually the age of D0 when D1 is updated. Similarly, the age difference

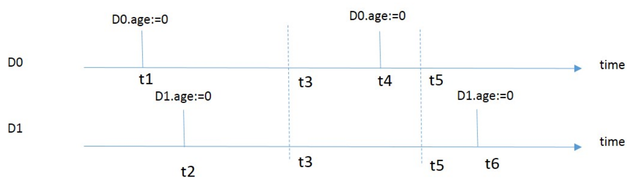

Fig. 7. Illustration of updates of data with respect to time

at t5 is equal to t4-t2, which is the age of D1 when D0 is updated. Therefore, we formulate relative validity of Di and Dj as the following query, which explores fewer states compared with the original one we mentioned in the previous paragraph: $A[]\,((Di.updated\, imply\, Dj.age <= RVI(i,j))$ $and\,(Dj.updated\, imply\, Di.age <= RVI(i,j)))$.

Similarly, the weak relative validity requirement requires that whenever Tk reads Di or Dj, the age differences of Di and Dj to be smaller than or equal to the relative validity interval RVI(i,j). This can be formulated as: $A[]\,((Tk.read\_di\, or\, Tk.read\_dj)\, imply$ $((Di.updated\, imply\, Dj.age <= RVI(i,j))\, and$ $(Dj.updated\, imply\, Di.age <= RVI(i,j))))$

The verification of timeliness equals to proving that location *Ti.deadline_miss* is not reachable. This requirement is equivalent to the following invariant property: $A[]\, not\, Ti.deadline\_miss$.

The proposed specification patterns are summarized in Table I. In each row, the table shows the informal description of the property, as well as the corresponding query patterns in UPPAAL TCTL.

B. Verification Results

We model the assumed system as described in Section IV and verify properties using UPPAAL 4.1.19. The properties are specified using the specification patterns from the previous subsection. The results are listed in Table II. All requirements have passed the verification. The table also lists the time it takes to verify each query, as well as the memory consumption. Since the system we have modeled is not complex, the time and memory costs look promising.

VI. RELATED WORK

Kung [13] applies pushed automata techniques to model and verify temporal constraints in a database. However, the temporal constraints he has verified do not include temporal data consistency. Song et al. [3] introduced the concept of temporal consistency. In their work, temporal consistency is evaluated via simulation, instead of formal verification. Han et al. [5] apply schedulability analysis on transactions to maintain temporal consistency. However, they do not consider concurrency control in the analysis. Both Lauer et al. [14] and Le Berre et al. [15] use formal methods to verify temporal data consistency in networked systems. Compared with our work, their work do not deal with database assumptions and do not model transaction behaviors.

Several researchers have used UPPAAL to model various aspects of database transactions and transaction management mechanisms. For example, Kot [7] models several selected transaction concurrency control mechanisms in UPPAAL and verify properties such as free of deadlock. Al-Bataineh et al.

TABLE I. SPECIFICATION PATTERNS

Property	Informal Specification	Query Pattern
Absolute validity	The age of Di must always be smaller than or equal to its absolute validity interval AVI(i)	$A[]\, Di.age <= AVI(i)$
Weak absolute validity	Whenever Tk reads Di, the age of Di must always be smaller than or equal to its absolute validity interval AVI(i)	$A[]\, (Tk.read\_di\, imply\, Di.age <= AVI(i))$
Relative validity	The age differences of Di and Dj to be smaller than or equal to the relative validity interval RVI(i,j)	$A[]\, ((Di.updated\, imply\, Dj.age <= RVI(i,j))$ $and\, (Dj.updated\, imply\, Di.age <= RVI(i,j)))$
Weak relative validity	Whenever Tk reads Di or Dj, the age differences of Di and Dj to be smaller than or equal to the relative validity interval RVI(i,j)	$A[]\, ((Tk.read\_di\, or\, Tk.read\_dj)\, imply$ $((Di.updated\, imply\, Dj.age <= RVI(i,j))\, and$ $(Dj.updated\, imply\, Di.age <= RVI(i,j))))$
Transaction timeliness	Transaction Tk will not miss its deadline	$A[]\, not\, Tk.deadline\_miss$

TABLE II. VERIFICATION RESULTS

Requirement	Query	Verification Time	Memory Consumption	Explored States	Status
Requirement 1.1	$A[]\, D0.age <= 15$	0.577s	9716KB	54996	Satisfied
Requirement 1.2	$A[]\, D1.age <= 16$	0.609s	9844KB	54643	Satisfied
Requirement 2.1	$A[]\, (T2.read\_d0\, imply\, D0.age <= 15)$	0.608s	9860KB	54996	Satisfied
Requirement 2.2	$A[]\, (T2.read\_d1\, imply\, D1.age <= 16)$	0.608s	9868KB	54643	Satisfied
Requirement 3.1	$A[]\, ((D0.updated\, imply\, D1.age <= 18)$ $and\, (D1.updated\, imply\, D0.age <= 18))$	0.765s	10008KB	57893	Satisfied
Requirement 3.2	$A[]\, ((T2.read\_d0\, or\, T2.read\_d1)\, imply$ $((D1.updated\, imply\, D0.age <= 18)\, and$ $(D0.updated\, imply\, D1.age <= 18)))$	0.780s	10044KB	57893	Satisfied
Requirement 4	$A[]\, not\, T2.deadline\_miss$	0.468s	9680KB	58729	Satisfied

[16] uses UPPAAL to model a two-phase commit protocol for an RTDBMS. In our previous work, we have proposed a flexible approach to verify transaction timeliness and isolation using UPPAAL [6]. Although these works also use model checking and UPPAAL, they focus on other aspects of the database than temporal data consistency.

VII. CONCLUSION

In this paper we have described a model-checking approach for verification of transaction timeliness and temporal data consistency in a real-time database within a cyber-physical system. We have modeled the transaction work units, the data and the concurrency control mechanisms as a network of automata. This work continues our previous work [6] in an attempt to create a framework for verification of RTDBMS, with respect to verifying transaction timeliness v.s. temporal data consistency. The properties are specified in TCTL using our proposed specification patterns. The formalized properties are model checked using UPPAAL. An example RTDBMS with update and control transactions has been used to demonstrate the model checking approach. All properties have been proved satisfied within short time and with low memory consumption.

One possible problem of this approach is the potential state explosion when the modeled system incorporates a large number of transactions and data. This can be mitigated by partitioning the transactions and data according to their dependencies. For example, many CPSs apply distributed data management. Instead of modeling the entire system directly, it may be possible to verify the properties in the local databases, and the model check, or deductively prove, the properties of the entire system.

REFERENCES

[1] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, "Cyber-physical systems: A new frontier," in *Machine Learning in Cyber Trust*. Springer US, 2009, pp. 3–13.

[2] K.-D. Kang and S. Son, "Real-time data services for cyber physical systems," in *Distributed Computing Systems Workshops, 2008. ICDCS '08. 28th International Conference on*, June 2008, pp. 483–488.

[3] X. Song and J. Liu, "Performance of multiversion concurrency control algorithms in maintaining temporal consistency," in *Proceedings of the Fourteenth Annual International Computer Software and Applications Conference*, 1990, pp. 132–139.

[4] K. Ramamritham, S. H. Son, and L. C. Dipippo, "Real-time databases and data services," *Real-Time Syst.*, vol. 28, no. 2-3, pp. 179–215, 2004.

[5] S. Han, K. yiu Lam, J. Wang, S. H. Son, and A. K. Mok, "Adaptive co-scheduling for periodic application and update transactions in real-time database systems," *Journal of Systems and Software*, vol. 85, no. 8, pp. 1729 – 1743, 2012.

[6] S. Cai, B. Gallina, D. Nyström, and C. Seceleanu, "Flexible verification of transaction timeliness and isolation," Tech. Rep., 2016. [Online]. Available: http://www.es.mdh.se/publications/4276-

[7] 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.

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

[9] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Information and Computation*, vol. 104, no. 1, pp. 2 – 34, 1993.

[10] 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.

[11] B. Gallina, "Prisma: a software product line-oriented process for the requirements engineering of flexible transaction models," Ph.D. dissertation, University of Luxembourg, 2010.

[12] B. Kao, K. Y. Lam, B. Adelberg, R. Cheng, and T. Lee, "Updates and view maintenance in soft real-time database systems," in *Proceedings of the Eighth International Conference on Information and Knowledge Management*, ser. CIKM '99, 1999, pp. 300–307.

[13] C. H. Kung, "On verification of database temporal constraints," in *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '85, 1985, pp. 169–179.

[14] M. Lauer, F. Boniol, C. Pagetti, and J. Ermont, "End-to-end latency and temporal consistency analysis in networked real-time systems," *International Journal of Critical Computer-Based Systems*, vol. 5, no. 3-4, pp. 172–196, 2014.

[15] T. Le Berre, P. Mauran, G. Padiou, and P. Quéinnec, "A Data Oriented Approach for Real-Time Systems," in *17th International Conference on Real-Time and Network Systems*, Paris, France, Oct. 2009, pp. 147–158.

[16] 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*, Sept 2012, pp. 65–72.