# Statistical Model Checking for Real-Time Database Management Systems: A Case Study

Simin Cai*, Barbara Gallina†, Dag Nyström‡, and Cristina Seceleanu§
Mälardalen University, Västerås, Sweden
Email: *simin.cai@mdh.se, †barbara.gallina@mdh.se, ‡dag.nystrom@mdh.se, §cristina.seceleanu@mdh.se

***Abstract*—Many industrial control systems manage critical data using Database Management Systems (DBMS). The correctness of transactions, especially their atomicity, isolation and temporal correctness, is essential for the dependability of the entire system. Existing methods and techniques, however, either lack the ability to analyze the interplay of these properties, or do not scale well for systems with large amounts of transactions and data, and complex transaction management mechanisms. In this paper, we propose to analyze large scale real-time database systems using statistical model checking. We propose a pattern-based framework, by extending our previous work, to model the real-time DBMS as a network of stochastic timed automata, which can be analyzed by UPPAAL Statistical Model Checker. We present an industrial case study, in which we design a collision avoidance system for multiple autonomous construction vehicles, via concurrency control of a real-time DBMS. The desired properties of the designed system are analyzed using our proposed framework.**

***Keywords*—Real-time Database Management System, Statistical Model Checking, Autonomous Vehicles, Case Study**

## I. Introduction

Database Management Systems (DBMS) have played an essential role in many industrial control systems [1], since they not only act as a central storage for critical data, but also provide a series of transaction management mechanisms to ensure the logical consistency of data. Among them, *Concurrency Control* (CC) prevents harmful manipulation of data by arbitrary concurrent transactions, which ensures the so-called *isolation* [2]. *Abort Recovery* (AR) restores the system into a consistent state when transactions fail, which achieves the *atomicity* of transactions [2].

Industrial control systems often also impose time constraints on their computations, which are inherited by the transactions managed by the Real-Time DBMS (RTDBMS). We refer to such time constraints as *temporal correctness*, which consists of two aspects [3]. On one hand, *timeliness* requires transactions to meet their specified deadlines. On the other hand, the data accessed by transactions must be fresh and relevant in time, which is referred to as *temporal validity*.

To achieve a dependable industrial control system, one needs to ensure that the DBMS it relies on can guarantee the desired atomicity, isolation and temporal correctness. Although much effort has been made on analyzing these properties separately, few works have provided means to analyze them together in a DBMS, such that trade offs between logical and temporal correctness can be reasoned about. In our previous work [4], [5], we have proposed a formal framework, called UPPCART (UPPaal for Concurrent Atomic Real-time Transactions), for the verification of these properties in transaction-based systems. In UPPCART, we have proposed timed-automata patterns for modeling database transactions together with the transaction management mechanisms, and verified the atomicity, isolation and temporal correctness using exhaustive model checking. However, as the amounts of transactions and data increase, and the complexity of the mechanisms rises, the state space to be explored by model checking may grow drastically. Consequently, the model checker may fail to reach any conclusion on whether or not the properties are satisfied.

In this paper, we propose to analyze atomicity, isolation and temporal correctness of complex DBMS with Statistical Model Checking (SMC) [6], which applies simulation-based techniques to avoid exhaustive state space exploration, and provides probabilistic guarantees for property satisfaction. The challenge lies in modeling the transactions together with various CC and AR mechanisms, such that these properties can be analyzed by the state-of-art tools. To achieve this, we propose a framework called UPPCART-SMC to model the transactions and the mechanisms as a network of stochastic timed automata, which can be analyzed by the UPPAAL SMC statistical model checker [7]. The satisfaction of these properties of the designed system can be evaluated as probability intervals, with an associated confidence, and compared with required probability thresholds. We also propose patterns to construct UPPCART-SMC models, which can reduce modeling effort and facilitate automated model construction.

We apply our proposed framework on a case study, suggested by the automotive industry, which aims to verify the design correctness of a collision avoidance system for multiple autonomous construction vehicles working in a mining quarry. We present a design that prevents vehicles from operating in critical areas simultaneously, via concurrency control of a common DBMS connected to the vehicles. To ensure safety and maintain productivity, a set of transactional properties regarding atomicity, isolation and temporal correctness should be verified during the design phase of the DBMS. We apply our UPPCART-SMC framework to model this system, and verify the desired properties using UPPAAL SMC.

The remainder of the paper is organized as follows. In Section II, we recall the background, including our previous UPPCART framework. In Section III, we introduce our extended UPPCART-SMC framework. We present the case study in Section IV, followed by the related work in Section V. In Section VI, we conclude the paper and outline our future work.

## II. BACKGROUND

### A. UPPAAL Timed Automata and UPPAAL Model Checker

An UPPAAL Timed Automaton (TA) [8] is defined as the following tuple:

$$TA ::= (L, l_0, X, V, I, Act, E), \tag{1}$$

in which: $L$ is a finite set of locations, $l_0$ is the initial location; $X$ is a finite set of clock variables; $V$ is a finite set of discrete variables; $I : L \rightarrow B(X)$ assigns invariants (predicates) to locations, where $B(X)$ denotes the set of clock constraints; $Act$ is a set of synchronization channels; $E \subset L \times B(X,V) \times Act \times R \times L$ is a set of edges, where $B(X,V)$ denotes the set of guards, $R$ denotes the set of assignments. We denote an edge $e \in E$ as $l \xrightarrow{g,a,r} l'$, where $g \in B(X,V)$, $a \in Act$, and $r \in R$. The state of a TA is defined as a pair $(l,u)$, where $l$ is the active location and $u$ is a valuation of all clock variables. At each state, the automaton may non-deterministically take a *delay transition*, denoted by $(l,u) \xrightarrow{d} (l, u+d)$, which only increases the clock values by $d$ time units, as long as the invariant associated with the current location is not violated. Alternatively, it may non-deterministically take a *discrete transition* $(l,u) \xrightarrow{a} (l',u')$ following an enabled edge $l \xrightarrow{g,a,r} l'$, update the discrete variables and reset clocks.

Multiple TA can form a Network of Timed Automata (NTA) via parallel composition ("||"), by which individual TA are allowed to carry out internal actions (i.e., interleaving), while pairs of TA can perform hand-shake synchronization via channels, or non-blocking communication via broadcast channels. For a communication via channel $c$, the sender is denoted by "c!", while the receiver by "c?". A location can be **urgent** (marked as "U") or **committed** (marked as "C"). An automaton reaching an urgent location must take the next transition without any delay in time, but another automaton may take transitions at this time, as long as the time does not progress. When an automaton is at a committed location, another automaton may NOT take any transitions, unless it is also at a committed location. The state of an NTA consists of the values of all clocks in the NTA, together with the currently visited locations of each TA, respectively.

Requirements can be specified in the UPPAAL query language, which is a decidable subset of CTL (Computational Tree Logic), and verified exhaustively and automatically by the UPPAAL model checker [8]. Among others, we focus on: (i) invariance properties, specified as $A\ [\,]\ P$, meaning that P always holds for all possible execution paths; and (ii) liveness properties, specified as $P \rightarrow Q$ ($P$ leads to $Q$), meaning that if P holds, Q will eventually hold.

### B. UPPAAL Stochastic Timed Automata and UPPAAL SMC

UPPAAL Stochastic Timed Automata (STA) [7] extends UPPAAL TA with stochastic interpretations. Formally, an STA is defined as

$$STA ::= (TA, \mu, \gamma), \tag{2}$$

where $\mu$ is a probability density function that assigns delays to the delay transitions in the TA. For time-bounded delays, $\mu$ gives a uniform distribution between the time bounds. For unbounded delays, $\mu$ follows an exponential distribution. $\gamma$ is a density function that assigns the probability distribution

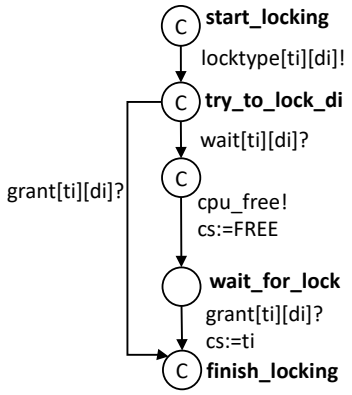


Fig. 1. UPPCART pattern for locking [4], [5]

defined by the user for multiple discrete transitions. Similar to NTA, multiple STA can form a Network of STA (NSTA), and communicate via broadcast channels.

UPPAAL SMC provides probability evaluation, hypothesis testing and probability comparison for STA models. It applies a simulation-based technique called statistical model checking [6], which avoids the state explosion problem of exhaustive model checking. Given $\star$ to denote either the eventually ($<>$) or the globally ($[\,]$) temporal operator, *probability evaluation* calculates $Pr(\star_{x \leq n} \phi)$, that is, the probability that property $\phi$ is eventually (or globally) satisfied within $n$ time units. This is specified as $Pr[\leq\ n](\star\phi)$ in the UPPAAL SMC query language. *Hypothesis testing* compares $Pr(\star_{x \leq n} \phi)$ with a given value *p*, specified as $Pr[\leq\ n](\star\phi)\ \sim\ p$, where $\sim \in \{<, \leq, >, \geq\}$. For more details about UPPAAL SMC, we refer to literature [7].

### C. The UPPCART Framework

In our previous work [4], [5], we have proposed a pattern-based framework, called UPPCART (UPPaal for Concurrent Atomic Real-time Transactions), for modeling transaction-based systems, which we verify by model-checking with respect to atomicity, isolation and temporal correctness. The transaction system is modeled as a timed automata network, which is made of automata representing the computational work to be executed, and automata observing the violation of the desired transactional properties. Formally, we define a real-time transaction-based system $N$ as follows:

$$\begin{aligned} N ::= & W_0 \,||\, ... \,||\, W_{n-1} \,||\, A_{CCManager} \,||\, A_{ATManager} \\ & ||\, O_0 \,||\, ... \,||\, O_{k-1} \,||\, D_0 \,||\, ... || \, D_{l-1}, \end{aligned} \tag{3}$$

where $W_0$, ..., $W_{n-1}$ are the TA of Work Units of transactions $T_0$, ..., $T_{n-1}$, respectively. These work unit TA also model the work units' interaction with the transaction manager with respect to concurrency control and abort recovery. $A_{CCManager}$ is the CCManager automaton that models the CC algorithm, and interacts with the work unit TA. $A_{ATManager}$ is the ATManager automaton that models the atomicity controller of recovery mechanisms. $O_0$, ..., $O_{k-1}$ are the TA of IsolationObservers that observe the unwanted transaction interleavings precluded by the desired isolation requirement, respectively. $D_0$, ..., $D_{l-1}$ are the TA that monitor the age of data.

To reduce the modeling efforts for various concurrency control algorithms, we have proposed a set of automata skeletons and parameterized patterns as modules to construct the

TABLE I. UPPAAL QUERY PATTERNS FOR VERIFYING TRANSACTIONAL PROPERTIES [4], [5]

| Property Type | Property Description | UPPAAL Query Pattern |
|---|---|---|
| Atomicity | $T_i$ aborted due to ERRORTYPE is eventually rolled back (compensated) | $(ATManager.abort\_id == i \,\&\&\, ATManager.error\_type == ERRORTYPE) \rightarrow W_i.trans\_rolledback\,(W_i.trans\_compensated)$ |
| Isolation | The specified isolation phenomena never occur | $A[]\ not\,(O_1.isolation\_phenomenon \,||\, ... \,||\, O_n.isolation\_phenomenon)$ |
| Timeliness | $T_i$ never misses its deadline | $A[]\ not\, W_i.miss\_deadline$ |
| Absolute Temporal Validity | When read by $T_i$, $D_j$ is never older than the absolute validity interval AVI(j) | $A[]\,(W_i.read\_di\_done\, imply\, Dj.age <= AVI(j))$ |
| Relative Temporal Validity | Whenever $T_i$ reads $D_j$ or $D_l$, the age differences of $D_j$ and $D_l$ is smaller than or equal to the relative validity interval RVI(j,l) | $A[]\,((W_i.read\_dj\_done \,||\, W_i.read\_dl\_done)\, imply\, ((Dj.age - Dl.age <= RVI(j,l)) \,\&\&\, (Dl.age - Dj.age <= RVI(j,l))))$ |

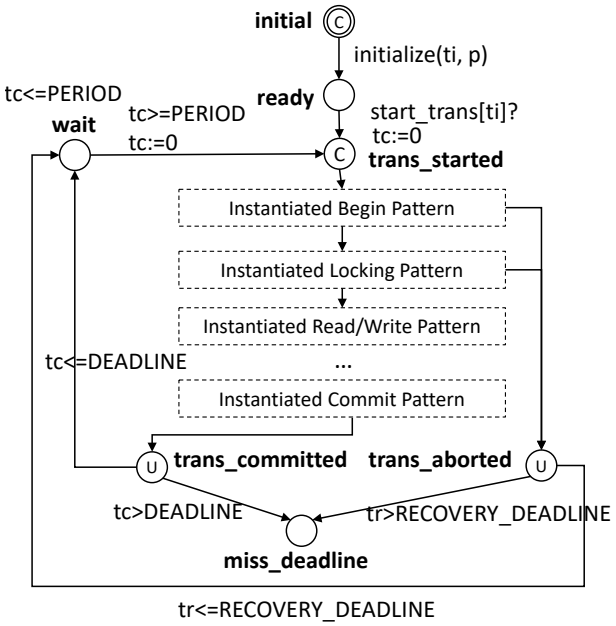


Fig. 2. Pattern-based construction of a work unit automaton [4], [5]

models of the system. For instance, Fig. 1 presents a pattern for modeling the locking operation used for concurrency control. This piece of a work unit automaton models the acquisition of a lock by sending a message via the *locktype[ti][di]* channel, and waits for the granting message from the CCManager automaton. Patterns can then be instantiated, and composed with other types of instantiated patterns, such as the ones for the read and write operations. As an example, Fig. 2 shows a TA of a transaction constructed from instantiated patterns. We have proposed patterns for modeling database operations, CCManager, ATManager, data objects, as well as IsolationObservers. Using them, basic structures of models can be reused to compose a complex model, and thus reduces the modeling efforts and allows for automated construction.

Atomicity, isolation and temporal correctness of the modeled system can be specified in UPPAAL queries (Table I), and verified by UPPAAL model checker. Among them, atomicity is formalized as a liveness property, stating that if an abortion occurs, a predefined recovery mechanism is eventually performed. Isolation is formalized as an invariance property that the locations representing the unwanted interleavings can never be reached. Temporal correctness is also specified as invariance properties, requiring that the states representing violated deadlines or temporal validity are never reachable.

Due to the increased amounts of transactions and data, or higher complexity of the selected CC and AR mechanisms, exhaustive model checking may not find an answer within the given time or resource limit. For the analysis of large systems that are beyond the analytical capability of exhaustive model

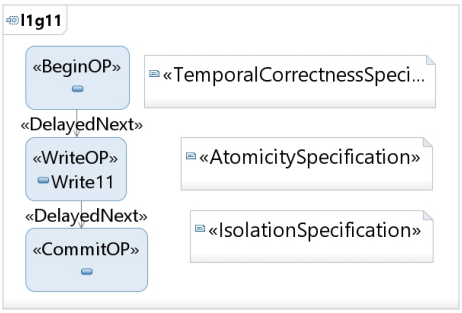


Fig. 3. An example of a transaction specified in UTRAN

checking, we resort to statistical model checking and UPPAAL SMC, which is presented in Section III.

### D. UTRAN

In this paper, we use a high-level language called UTRAN (UML for TRANsactions) [5] to describe the transactions in the case study. UTRAN is a UML profile that provides a series of stereotypes to specify transactions and their properties. In UTRAN, a transaction is specified as a UML activity composed of database operations. As an example, Fig. 3 shows a transaction consisting of a begin operation, a write operation that updates data with id 11, and a commit operation. The worst- and best- case execution times can be specified for each operation as annotated features. Delays between operations can also be annotated to the edges between the operation nodes. The stereotype «TemporalCorrectnessSpecification» specifies the temporal constraints, such as periods, deadlines and temporal data validities of transactions. «AtomicitySpecification» and «IsolationSpecification», specify the selected atomicity and isolation variants, as well as the recovery and concurrency control algorithms, respectively.

## III. THE UPPCART-SMC FRAMEWORK

In order to analyze atomicity, isolation and temporal correctness for large RTDBMS, we propose the UPPCART-SMC framework that models the transactions, together with the CC algorithm and the AR mechanisms, as a network of UPPAAL stochastic timed automata. Denoted by $N'$, the NSTA of the modeled real-time transactions is defined as follows:

$$N' ::= W'_0 \,||\, ... \,||\, W'_{n-1} \,||\, A'_{CCManager} \,||\, A'_{ATManager} \,||\, O'_0 \,||\, ... \,||\, O'_{k-1} \,||\, D'_0 \,||...||\, D'_{m-1}. \quad (4)$$

Since we are modeling the exact same RTDBMS as in Section II-C, each STA in Equation 4 corresponds to a TA in Equation 3. That is to say, $W'_0$, ..., $W'_{n-1}$ are the STA of work units of transactions $T_0$, ..., $T_{n-1}$, respectively. $A'_{CCManager}$ is the CCManager STA. $A'_{ATManager}$ is the ATManager STA. $O'_0$, ..., $O'_{k-1}$ are the STA of IsolationObservers. $D_0$, ..., $D_{l-1}$ are the STA that monitor the age of data.

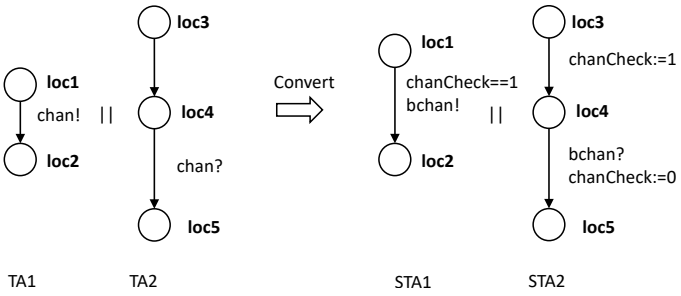


Fig. 4. Handshake synchronization modeling: converting from using a handshake synchronization channel to using a broadcast channel [7]

To construct $N'$, we apply a pattern-based approach, that is, each STA is constructed via the instantiation of a set of parametrized patterns, which are created such that they incorporate STA-related syntax and semantics. Therefore, these STA patterns are not a simple rewrite of the UPPCART TA patterns. A series of modifications are required on the UPPCART patterns, regarding the syntax and semantics. In addition, as the system grows more complex, constructing the models manually requires considerable effort, even if a pattern-based construction can alleviate some of the burden. Another challenge lies in the analysis using UPPAAL SMC, which not only adopts a different specification language than UPPAAL, but also lacks support for the "leads-to" property that is essential to atomicity specification. We address these challenges in the next subsections, respectively.

### A. UPPCART-SMC Patterns

In order to model synchronizations between the work unit automata, the CCManager, and the ATManager in UPPCART-SMC, we use broadcast channels together with shared variables, as suggested in literature [7].

Since in UPPCART we have used the handshake synchronization mechanism instead, as supported by the UPPAAL tool, we use a conversion pattern from handshake to broadcast synchronization, presented in Fig. 4. The left side of the conversion is an NTA composed of TA1 and TA2, synchronized via synchronization channel *chan* exactly when TA1 is at *loc1* and TA2 is at *loc4*. We perform the conversion by replacing the *chan* with a broadcast channel *bchan*, and control the handshake via an integer variable *chanCheck*. STA1 can send the signal via *bchan*, only when *chanCheck* equals 1. Before reaching *loc4*, STA2 sets *chanCheck* to 1, and waits at *loc4* for the broadcast signal. On receiving the signal, STA2 sets *chanCheck* to 0, such that the broadcast channel cannot be fired. By following this scheme, we achieve handshake synchronization in our model.

As an example, Fig. 5 shows the Locking-SMC Pattern, converted from the Locking Pattern in Fig. 1. For each synchronization channel (locktype[ti][di], wait[ti][di] and grant[ti][di]), we change them to broadcast channels, define an integer variable (locktypeCheck[ti][di], waitCheck[ti][di] and grantCheck[ti][di], respectively), and add the corresponding guard conditions and assignments according to the conversion pattern. The added guards and assignments are formatted in both bold and italic in Fig. 5. Since *cpu_free* is a broadcast channel, no changes are made for it during the conversion.

A significant semantic implication introduced by UPPAAL SMC, compared to UPPAAL, is that non-deterministic transitions in TA become probabilistic in STA. Since the outgoing edges of each location are either guarded by mutually-exclusive clock constraints, or by different synchronization channels, all discrete transitions in the framework are deterministic. Regarding the delay transitions, the locations in UPPCART-SMC fall into two categories: (i) Committed locations, in which case the automaton does not delay, but to take a discrete transition immediately; (ii) Invariant-associated locations, in which case the delays at such locations follow a uniformed distribution. We adopt this assumption when modeling our transaction systems. However, user-specified distribution can be modeled through adaptations in the STA [7].

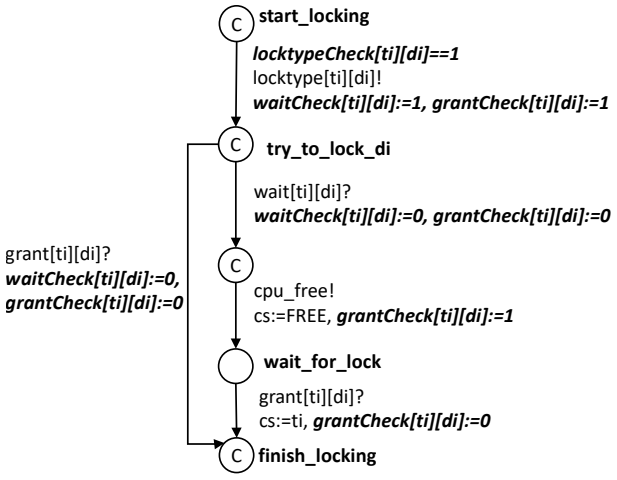


Fig. 5. Locking-SMC Pattern

*Automated Model Construction:* To ease the modeling effort, we have developed a tool to construct the UPPCART-SMC models from UTRAN specifications. Our tool implements a straightforward mapping between the elements in UTRAN, and the patterns in UPPCART-SMC. It accepts a description of transactions in UTRAN as input, and constructs models by instantiating and composing the patterns corresponding to the elements in the description. The source code of the tool can be accessed in our repository [9].

### B. Statistical Analysis

We use UPPAAL SMC to analyze the STA models, and check the transactional properties. The queries for checking isolation, timeliness, absolute validity and relative validity are listed as SQ2-SQ4 in Table II, and they encode the probability of their satisfaction, over a chosen simulation time (specified as $n$), and over a number of runs computed by the tool. For hypothesis testing, these queries can be extended with a comparison to a predefined probability value.

As the atomicity property specified with "leads-to" in Table I cannot be checked statistically using UPPAAL SMC, we choose to analyze time-bounded atomicity, a stricter version of atomicity, which can be specified as a checkable query. We analyze the probability that $T_i$ is rolled back (or compensated) within a time bound (e.g., $T_i$'s *RECOVERY_DEADLINE*) after its abortion, specified as the following time-bounded property:

$$(ATManager.abort\_id == i \quad \&\& \\ ATManager.error\_type == ERRORTYPE) \\ \Rightarrow_{\leq RECOVERY\_DEADLINE} Wi.trans\_rolledback.$$

TABLE II. UPPAAL SMC QUERY PATTERNS FOR ANALYZING TRANSACTIONAL PROPERTIES

| Property Type | Property Description | UPPAAL Query Pattern | ID |
|---|---|---|---|
| Time-bounded Atomicity | $T_i$ aborted due to ERRORTYPE is rolled back or compensated within *RECOVERY_DEADLINE* | $Pr\,[<= n]\,(<>\ W_i.b)$ | SQ1.1 |
| | | $Pr\,[<= n]\,([]\ (W_i.b\ \ imply\ W_i.ta <= RECOVERY\_DEADLINE))$ | SQ1.2 |
| Isolation | The probability of that the specified isolation phenomena never occur | $Pr\,[<= n]\,([]\ \ not\ \ (O_1.isolation\_phenomenon\,\|\|\ ...\|\|\,O_n.isolation\_phenomenon))$ | SQ2 |
| Timeliness | The probability of $T_i$ never missing its deadline | $Pr[<= n]([]\ not\ \ W_i.miss\_deadline)$ | SQ3 |
| Absolute Validity | The probability of that, when read by $T_i$, $D_j$ is never older than the absolute validity interval AVI(j) | $Pr[<= n]([]\,(W_i.read\_di\_done\,imply\ Dj.age <= AVI(j)))$ | SQ4 |
| Relative Validity | The probability of that, whenever $T_i$ reads $D_j$ or $D_l$, the age differences of $D_j$ and $D_l$ is smaller than or equal to the relative validity interval RVI(j,l) | $Pr[<= n]([]((W_i.read\_dj\_done\,\|\|\,W_i.read\_dl\_done)\ imply\,((Dj.age - Dl.age <= RVI(j,l))\ \&\&\ (Dl.age - Dj.age <= RVI(j,l)))))$ | SQ5 |

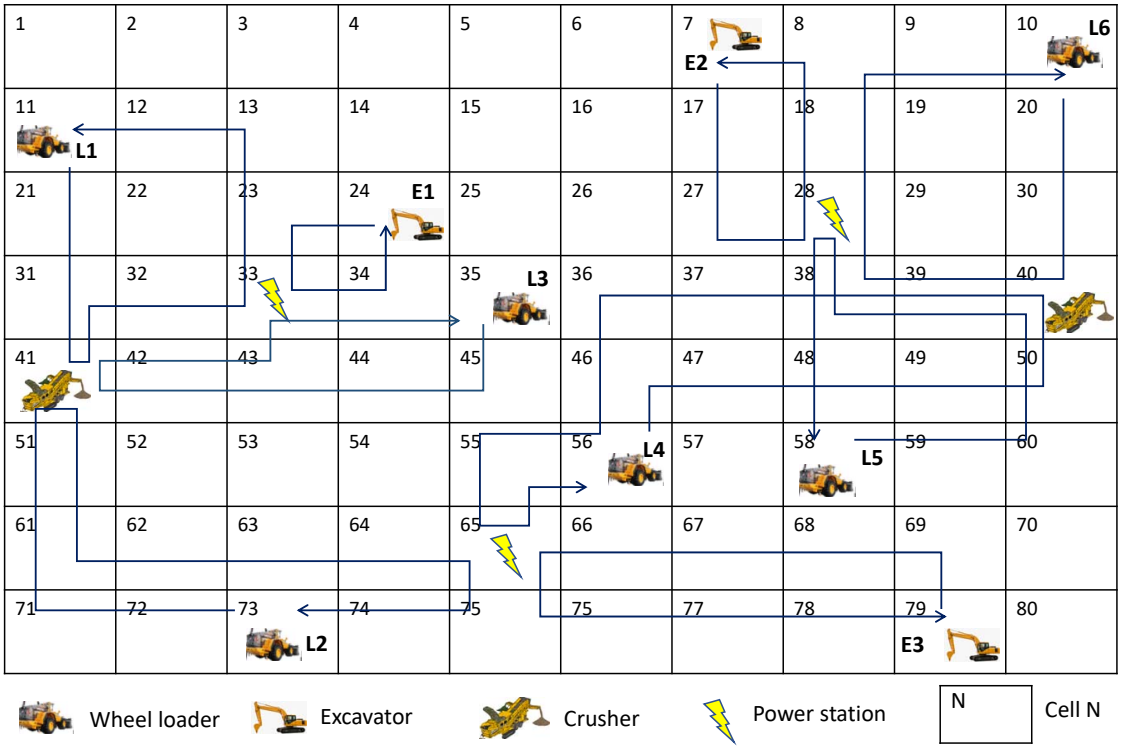


Fig. 6. Map and paths of vehicles in the AutoQuarry case study

This property can be checked by UPPAAL SMC with some modifications in the models, as suggested by literature [10]. We first introduce a clock variable *ta* in $W_i$, whose value is reset when $ATManager.abort\_id == i\ \&\&\ ATManager.error\_type == ERRORTYPE$; and a boolean variable $b$ in $W_i$, whose value is set to *true* when $ATManager.abort\_id == i \&\& ATManager.error\_type == ERRORTYPE$ holds, and to *false* as soon as $Wi.trans\_rolledback$ holds. The probability of the time-bounded leads-to property is then reduced to checking the probability of: $[]\,(Wi.b\ \ imply\ \ Wi.ta <= RECOVERY\_DEADLINE)$, which holds over the number of runs checked during analysis, and is specified as the UPPAAL SMC query SQ1.2 in Table II. To ensure that this query is not trivial, we also check that $Wi.b$ is reachable, via the query SQ1.1.

## IV. A CASE STUDY

In this section, we present a case study to show the applicability and usefulness of our UPPCART-SMC framework. We consider the design-time verification for a system model containing multiple construction vehicles that mine and transport iron ores in a quarry autonomously. The workflows of the vehicles are managed by a DBMS in the case study, as workflow management is a typical application scenario of database systems, and exemplifies complex real-time transaction management. Each vehicle is assigned a mission that consists of a series of activities. For instance, an excavator digs the ores at an assigned ore pit, and moves for charging from time to time. A wheel loader scoops the ore from a pit, transports them to a crusher, and travels to the power station before moving back to the pit. If we represent the map as a grid, a mission is actually a sequence of cells in the grid to be visited. Fig. 6 presents the map of the quarry in our case study. Six wheel loaders (L1, ..., L6) and three excavators (E1, E2 and E3) are deployed in the quarry with their assigned missions. For instance, L1 starts digging ores at at Cell 11, transports to the crusher at Cell 41, after which it refuels at Cell 33, before traveling back to Cell 11 and restarting its mission. As illustrated in the figure, the vehicles do not share only the crushers and the power stations, but also some cells as their paths may overlap.

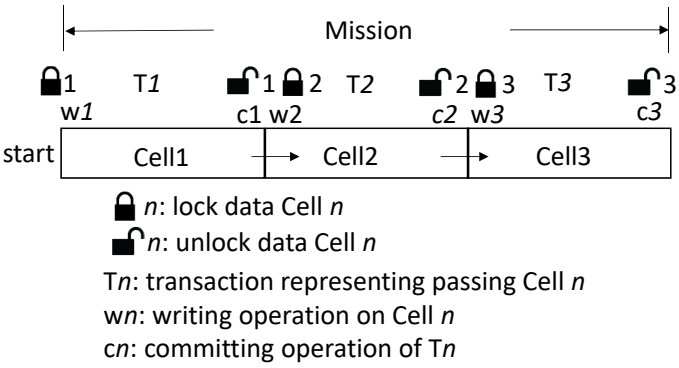


Fig. 7. Illustration of collision avoidance through transactions and CC

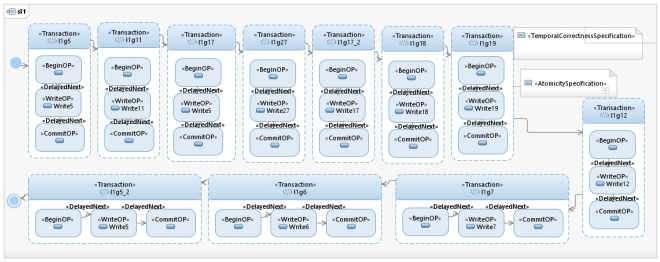

Fig. 8. Excerpt of the UTRAN specification for L1 in IBM RSA

TABLE III. TRANSACTIONS IN THE COLLISION AVOIDANCE DBMS

| Vehicle | Dead-line | Period | Trans-actions | Atomicity | Isolation |
|---|---|---|---|---|---|
| L1 | 1500s | 2000s | 11 | When G33 (charg-ing) gets aborted, redo G33 | Vehicles should not access the power stations and crushers simultaneously. In this case study, as an example, we ensure that transactions do not access Cells 33 (po-wer), and 41(crusher), simultaneously. |
| L2 | 2500s | 3000s | 15 | When G65 (charg-ing) gets aborted, redo G12 | |
| L3 | 1700s | 1700s | 11 | When G33 (charg-ing) gets aborted, redo G33 | |
| L4 | 2500s | 2800s | 17 | When G65 (charg-ing) gets aborted, redo G33 | |
| L5 | 1900s | 2000s | 11 | When G28 (charg-ing) gets aborted, redo G33 | |
| L6 | 1900s | 2000s | 11 | When G28 (charg-ing) gets aborted, redo G33 | |
| E1 | 2000s | 2100s | 5 | Not Applicable | |
| E2 | 2200s | 2400s | 7 | Not Applicable | |
| E3 | 1900s | 2000s | 11 | Not Applicable | |

Safety and productivity are the main concerns that need to be verified when multiple vehicles are working simultaneously and autonomously. First of all, it is crucial and challenging to ensure collision avoidance between moving vehicles, during their missions. In addition, to achieve desired productivity, each vehicle is scheduled to operate its mission with a specific period and by a given deadline, respectively. Since there are more wheel loaders than excavators, for better productivity, it is further required to allow the excavator to be charged whenever necessary. Therefore, we assume that the excavators have a higher priority than the loaders in their missions.

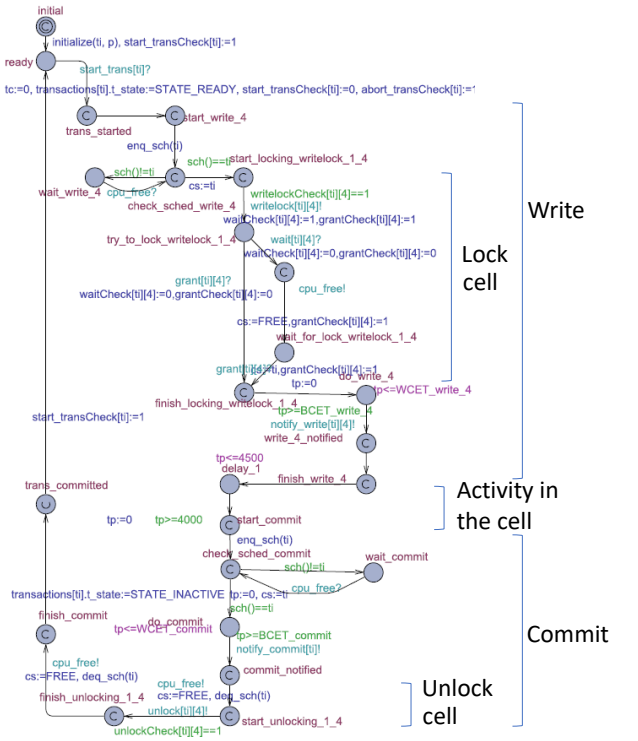


Fig. 9. Automaton for transaction $T_{L1G11}$

### A. System Description

Our collision-avoidance system is designed to prevent multiple vehicles from operating in the same cell simultaneously, via concurrency control provided by a central DBMS. The DBMS stores the map cells as data objects, and is connected to every vehicle. Fig. 7 illustrates the key idea of collision avoidance via concurrency control by a DBMS. In this figure, the operation of a vehicle within a cell is considered as a transaction, while each mission is treated as a sequence of transactions. Before entering a cell, the vehicle needs to lock the cell in the database by performing a write operation on the data, such that no other vehicles can enter. After the vehicle completes its work in the cell (digging, charging, or traveling through the cell), it commits the transaction before leaving the cell such that the cell is unlocked. Even if another vehicle enters this unlocked cell (before the committing vehicle entering the next cell), since the committing vehicle is in full stop, these two are not operating their tasks simultaneously and hence are considered safe. In this scheme, each transaction contains a write operation on the cell data, the moving and charging activity in the cell with its best- and worst- case execution times, and a commit operation before it commits. For instance, $T_{L1G11}$ identifies L1 accessing Cell 11, which is specified using UTRAN in Fig. 3. The mission of a vehicle contains a series of transactions. For instance, the mission of L1 is a series as follows: $< T_{L1G11}, T_{L1G21}, T_{L1G31}, T_{L1G41}, T_{L1G31_2}, T_{L1G32}, T_{L1G33}, T_{L1G23}, T_{L1G13}, T_{L1G12}, T_{L1G11_2} >$.

To ensure the immediate access of the high-priority vehicle, we resort to applying a priority-based CC algorithm, Two Phase Locking High Priority (2PL-HP) [11], which aborts the low-priority transaction when two transactions try to lock the same data. Since excavators E1, E2 and E3 have higher priorities, the wheel loaders may be aborted when an excavator enters the same cell, especially at the power stations. In this

situation, as a compensation, the wheel loader should restart the charging transaction when the excavator finishes charging.

In Table III we list the missions of all vehicles, including their end-to-end deadlines, periods, and number of contained transactions. In total, the system incorporates 9 periodic missions, including 99 transactions, 6 compensation transactions, and 60 data items. The atomicity and isolation requirements are also listed. Atomicity mainly focuses on the compensation of aborted charging transactions. Isolation emphasizes the exclusive access to the power stations, which are the most critical special cases of exclusive access to all cells. We describe the transactions and the DBMS using UTRAN, in the IBM Rational Software Architect (RSA) environment [1]. Fig. 8 presents the UTRAN specification for the aforementioned mission of L1, which contains 11 UML activities stereotyped as UTRAN transactions, and specifications for atomicity and temporal correctness of these transactions. The complete UTRAN descriptions are provided in the repository [9].

### B. UPPAAL SMC Models

We constructed the UPPAAL STA models from the UTRAN specifications using our tool. We obtain 9 STA for the 9 missions, 99 STA for the 99 transactions, 6 STA for the 6 compensations, respectively, as well as 1 STA for the CC manager, 1 STA for the AR manager, and 9 respective IsolationObservers for the isolation phenomena of two vehicles entering the same critical cell. Fig. 9 shows the STA of $T_{L1G11}$ that models wheel loader L1 working in Cell 11. The locking, unlocking, writing and committing pieces, as well as the basic structure of the transaction automaton, are constructed by instantiating the predefined patterns. The automaton first performs a write operation, including locking the cell that the vehicle should enter. It then moves to the location of the activity in the cell, which is associated with the time bound of the activity. In the end, it commits the transaction and sets the cell free. The CCManager STA receives locking and unlocking requests from the transaction work units, and performs the permission, rejection, or release of the locks. Fig. 10 shows an example of an IsolationObserver that monitors the phenomenon when L1 and E1 enter the same power station (Cell 33) simultaneously. It receives the messages when the write and commit operations are performed by the transactions. When the two transactions succeed in writing the same Cell 33, the observer moves to the *isolation_phenomenon* location. Due to lack of space, the complete models, including the CCManager, ATManagers, as well as other transactions and phenomena, are presented in the online repository [9].

### C. Analysis

The properties to be analyzed are formalized as UPPAAL SMC queries according to Table II, and the statistical model checking results are listed in Table IV. We select 300000 time units (30000 seconds) as the time bound $n$ in the queries, because in this amounts of time, each vehicle should complete at least 10 periods of its mission, which we consider adequate for one complete work batch. In this case study, we consider 99.99999% is a sufficiently high threshold for the satisfaction of the properties.

[1] https://www.ibm.com/developerworks/downloads/r/architect/index.html

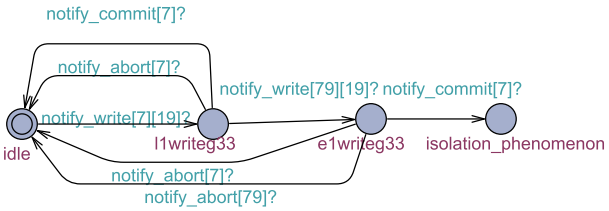


Fig. 10. IsolationObserver that observes conflicts between L1 and E1

We verify the queries using UPPAAL SMC (version 4.1.19), on a PC with an Intel i7-4800MQ CPU (2.70GHz, 8 cores), 16GB memory, and Ubuntu 16.04 (64-bit). The analysis results are also presented in Table IV, with a confidence level set to be 0.99. The results show that both isolation and timeliness of the transactions are satisfied with probabilities higher than or equal to 99.99999%. In terms of atomicity, L1, L2, L3, L5 and L6 are likely to get aborted during their missions (analyzed via SQ1.1), but once this happens, their respective compensation will be performed to maintain consistency with sufficiently high probability (analyzed via SQ1.2). As for L4, its mission is unlikely to be aborted. However, even if abortions happen, they will also be compensated with acceptable probability. Therefore, the current design is considered to satisfy the atomicity requirements.

*Discussion:* As a contrast to UPPCART-SMC, we also modeled the system in the case study with UPPCART, and tried to verify the properties using exhaustive model checking. The analysis failed to reach a conclusion due to memory exhaustion caused by state explosion, when the system reached 4 vehicles and 18 transactions. Compared to UPPCART, our UPPCART-SMC framework can deal with larger systems that contains more vehicles and transactions (9 vehicles and 99 transactions in this case). When the system scales up, longer analysis time is expected, but memory exhaustion will not occur with UPPCART-SMC.

## V. Related Work

A number of formal techniques have been developed for the verification of transactions. For instance, transaction atomicity of industrial workflow systems has been verified by Derks et al. [12] using Petri nets. Lanotte et al. [13] have proposed to verify atomicity and timing constraints of long-running transactions using his timed-automata-based language. Our previous work [5] has proposed an UPPAAL-based framework for the verification of atomicity, isolation and temporal correctness. These works exploit exhaustive verification, which is different from our work that applies statistical model checking for analysis while addressing complex transaction systems. Liu et al. [14] have recently proposed using statistical model checking and the tool PVESTA to analyze a distributed transaction protocol. They analyze read atomicity and lost update of transactions, which is a different focus if compared to our work.

Much work has been conducted to verify the correctness of mission plans for autonomous agents. Saha et al. [15] verify the collision avoidance of autonomous robots, whose movements are planned by a central controller. Their controller is not backed by a DBMS, and their verification approach is based on SMT solving, which is different from ours. Kamali et al. [16] and Saddem et al. [17] use UPPAAL to verify the correctness of the paths of autonomous vehicles and robotics, respectively.

TABLE IV. VERIFICATION RESULTS OF THE CASE STUDY

| Property | ID | UPPAAL SMC Query | Runs | Results | Time |
|---|---|---|---|---|---|
| Time-bounded Atomicity | SQ1.L1.1 | $Pr\,[<= 300000]\,(<>\ L1.b)$ | 24 | [0.8019,1] | 2126s |
| | SQ1.L1.2 | $Pr\,[<= 300000]\,([]\ (L1.b \quad imply \quad L1.ta <= 15000)) \geq 0.9999999$ | 458 | satisfied | 33657s |
| | SQ1.L2.1 | $Pr\,[<= 300000]\,(<>\ L2.b)$ | 74 | [0.7665, 0.9642] | 2166 |
| | SQ1.L2.2 | $Pr\,[<= 300000]\,([]\ (L2.b \quad imply \quad L2.ta <= 25000)) \geq 0.9999999$ | 458 | satisfied | 33363s |
| | SQ1.L3.1 | $Pr\,[<= 300000]\,(<>\ L3.b)$ | 24 | [0.8019,1] | 872s |
| | SQ1.L3.2 | $Pr\,[<= 300000]\,([]\ (L3.b \quad imply \quad L3.ta <= 12000)) \geq 0.9999999$ | 458 | satisfied | 33374s |
| | SQ1.L4.1 | $Pr\,[<= 300000]\,(<>\ L4.b)$ | 133 | [0.1648,0.3646] | 8670s |
| | SQ1.L4.2 | $Pr\,[<= 300000]\,([]\ (L4.b \quad imply \quad L4.ta <= 25000)) \geq 0.9999999$ | 458 | satisfied | 34172s |
| | SQ1.L5.1 | $Pr\,[<= 300000]\,(<>\ L5.b)$ | 166 | [0.4998,0.6991] | 8018s |
| | SQ1.L5.2 | $Pr\,[<= 300000]\,([]\ (L5.b \quad imply \quad L5.ta <= 12000)) \geq 0.9999999$ | 458 | satisfied | 34712s |
| | SQ1.L6.1 | $Pr\,[<= 300000]\,(<>\ L6.b)$ | 172 | [0.4285,0.6280] | 8984s |
| | SQ1.L6.2 | $Pr\,[<= 300000]\,([]\ (L6.b \quad imply \quad L6.ta <= 13000)) \geq 0.9999999$ | 458 | satisfied | 36402s |
| Isolation | SQ2 | $Pr\,[<= 300000]\,([]\ not\,(O1.isolation\_phenomenon\,\|\|...\|\|\,O6.isolation\_phenomenon)) \geq 0.9999999$ | 458 | satisfied | 33266s |
| Timeliness | SQ3 | $Pr\,[<= 300000]\,([]\ not\,(L1.miss\_deadline \quad ||... \quad || \quad E3.miss\_deadline) \geq 0.9999999$ | 458 | satisfied | 33924s |

The difference of our case study lies in the concurrency control provided by a DBMS, as well as that we apply statistical model checking to achieve better scalability.

## VI. Conclusions and Future Work

In this paper, we have proposed a framework for the analysis of large scale database management systems. The proposed UPPCART-SMC framework, built on top of our previous work, models the transactions and transaction management mechanisms as a network of stochastic timed automata, such that the atomicity, isolation and temporal correctness can be analyzed by UPPAAL SMC. We have demonstrated our framework via a case study, in which we analyze the correctness of a collision avoidance system for multiple autonomous construction vehicles. The case study involves a large number of transactions and data, which are formally modeled and analyzed using our framework. The verification proves that the desired properties are satisfied with acceptable confidence, in terms of probabilities.

Better tool automation, especially a full tool chain that covers the high-level specification of the DBMS and automated generation of formal models, as well as heuristic based selection between UPPCART and UPPCART-SMC, is one of the future works that can improve the applicability of our framework. Another future work is to integrate code verification for the C functions in the UPPAAL models. Currently, we assume that the functions encoding the algorithms are implemented correctly. In the future, such code can be verified with existing program verifiers.

**Acknowledgment:** The Swedish Research Council (VR) is gratefully acknowledged for supporting this research by the project "Adequacy-based Testing of Extra-Functional Properties of Embedded Systems".

## References

[1] S. Han, K.-Y. Lam, J. Wang, K. Ramamritham, and A. K. Mok, "On co-scheduling of update and control transactions in real-time sensing and control systems: Algorithms, analysis, and performance," *IEEE TKDE*, vol. 25, pp. 2325–2342, 2013.

[2] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 1992.

[3] K. Ramamritham, "Real-time databases," *Distributed and Parallel Databases*, vol. 1, no. 2, pp. 199–226, 1993.

[4] S. Cai, B. Gallina, D. Nyström, and C. Seceleanu, "A formal approach for flexible modeling and analysis of transaction timeliness and isolation," in *Proceedings of the 24th RTNS*. ACM, 2016, pp. 3–12.

[5] ——, "Specification and formal verification of atomic concurrent real-time transactions," in *Proceedings of the 23rd PRDC*. IEEE, 2018, pp. 104–114.

[6] H. L. Younes, "Verification and planning for stochastic processes with asynchronous events," Carnegie-Mellon University, Tech. Rep., 2005.

[7] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "UPPAAL SMC tutorial," *STTT*, vol. 17, no. 4, pp. 397–415, 2015.

[8] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *STTT*, vol. 1, no. 1, pp. 134–152, 1997.

[9] S. Cai. Statistical model checking for real-time database management systems: Repository. Password: etfa2019. [Online]. Available: https://www.idt.mdh.se/~sica/repo/

[10] M. Lindahl, P. Pettersson, and W. Yi, "Formal design and analysis of a gear controller," in *TACAS*. Springer, 1998, pp. 281–297.

[11] R. K. Abbott and H. Garcia-Molina, "Scheduling real-time transactions: A performance evaluation," *ACM TODS*, vol. 17, pp. 513–560, 1992.

[12] W. Derks, J. Dehnert, P. Grefen, and W. Jonker, "Customized atomicity specification for transactional workflows," in *The Proceedings of the 3rd CODAS*, 2001, pp. 140–147.

[13] R. Lanotte, A. Maggiolo-Schettini, P. Milazzo, and A. Troina, "Modeling long-running transactions with communicating hierarchical timed automata," in *Formal Methods for Open Object-Based Distributed Systems*. Springer, 2006, pp. 108–122.

[14] S. Liu, P. C. Ölveczky, K. Santhanam, Q. Wang, I. Gupta, and J. Meseguer, "Rola: A new distributed transaction protocol and its formal analysis," in *FASE*, 2018, pp. 77–93.

[15] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe ltl specifications," in *Proceedings of the 2014 IROS*. IEEE, 2014, pp. 1525–1532.

[16] M. Kamali, L. A. Dennis, O. McAree, M. Fisher, and S. M. Veres, "Formal verification of autonomous vehicle platooning," *Science of Computer Programming*, vol. 148, pp. 88–106, 2017.

[17] R. Saddem, O. Naud, K. G. Dejean, and D. Crestani, "Decomposing the model-checking of mobile robotics actions on a grid," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11 156–11 162, 2017.

[18] R. A. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., 2004.

[19] A. Adya, B. Liskov, and P. O'Neil, "Generalized isolation level definitions," in *Proceedings of the 16th ICDE*, 2000, pp. 67–78.