Customized Real-Time Data Management for Automotive Systems: A Case Study

Simin Cai, Barbara Gallina, Dag Nyström, Cristina Seceleanu
School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
{simin.cai, barbara.gallina, dag.nystrom, cristina.seceleanu}@mdh.se

***Abstract*—Real-time DataBase Management Systems (RTDBMS) have been considered as a promising means to manage data for data-centric automotive systems. During the design of an RTDBMS, one must carefully trade off data consistency and timeliness, in order to achieve an acceptable level of both properties. Previously, we have proposed a design process called DAGGERS to facilitate a systematic customization of transaction models and decision on the run-time mechanisms. In this paper, we evaluate the applicability of DAGGERS via an industrially relevant case study that aims to design the transaction management for an on-board diagnostic system, which should guarantee both timeliness and data consistency under concurrent access. To achieve this, we apply the pattern-based approach of DAGGERS to formalize the transactions, and derive the appropriate isolation level and concurrency control algorithm guided by model checking. We show by simulation that the implementation of our designed system satisfies the desired timeliness and derived isolation, and demonstrate that DAGGERS helps to customize desired real-time transaction management prior to implementation.**

I. Introduction

Modern vehicles are equipped with dozens of sensors and Electronic Control Units (ECUs), which generate hundreds of signals [1]. These data usually reside in shared data structures or simple in-memory databases, and processed by the ECUs [2], [3]. With the increasing abundance of functionalities in automotive systems, the possibility of sharing data among ECUs is also increasing, leading to potential risks of data inconsistency and safety issues [4]. Real-Time DataBase Management Systems (RTDBMSs) have been advocated as a promising means to avoid inconsistent data in automotive systems, as they provide the so-called ACID (Atomicity, Consistency, Isolation, Durability) guarantees for transactions [4], [5]. Since full ACID assurance may introduce unbounded delays, and since timeliness is usually more crucial than data consistency in automotive systems, full ACID assurance is often relaxed in order to enhance time-predictability [6].

To aid the design of an RTDBMS with proper ACID and timeliness trade-offs, we have proposed the DAGGERS (Data AGGregation for Embedded Real-time Systems) process, which helps to systematically derive the transaction model with traded-off ACID and timeliness properties, and decide the appropriate run-time mechanisms for the RTDBMS [7]. To support this, it applies formal modeling and model-checking techniques to reason about transaction properties, and ensures that timeliness and an acceptable level of ACID properties are satisfied under selected mechanisms. Eventually, the process generates a customized RTDBMS based on the derived transaction models and run-time mechanisms.

The main objective of this paper is to illustrate the design capability of the DAGGERS process, and evaluate its applicability, by applying it to an industrially relevant case study. This case study aims to design an RTDBMS for an on-board diagnostic system [8] that monitors the operational conditions of vehicles. The target system collects high-frequency data from a number of sensors mounted on the vehicle for condition measurements. These data are transmitted to a microcontroller on-board, and are aggregated before further analysis. The computations in this system have timeliness requirements, and may access the data concurrently. To develop such a system, a common solution in industry is to apply coarse-grained semaphores on shared memory for mutual exclusion, and perform schedulability analysis for checking whether deadlines are met. If timeliness cannot be guaranteed, the designer adjusts the code for better timeliness guarantee, without means to analyze possible data inconsistency caused by concurrency.

An RTDBMS provides isolation of transactions, meaning that the system are guaranteed to be free from a set of concurrency-related inconsistencies (called anomalies [9] or phenomena [10]), but at a cost of timeliness. To systematically reason about the trade-offs between isolation and timeliness, and derive transaction models customized for this particular system, we apply the DAGGERS process to the design of the transaction management. We construct formal models of an abstracted version of transactions and the concurrency control algorithms, and select the appropriate algorithm by means of model checking. In the end, we implement the derived transaction model on a prototype system, and evaluate by simulation whether the transaction properties are satisfied.

The remainder of the paper is organized as follows. The DAGGERS process is recalled in Section II, followed by the application of DAGGERS to the case study in Section III. The evaluation results of the developed system is presented in Section IV. Section V discusses the related work, before we conclude the paper and outline the future work in Section VI.

II. The DAGGERS Process

The DAGGERS process [7] provides an engineering methodology for developing a customized transaction management for RTDBMSs based on analysis of trade-offs between ACID and timeliness properties. As shown in Fig. 1, the main steps of DAGGERS are presented as follows.

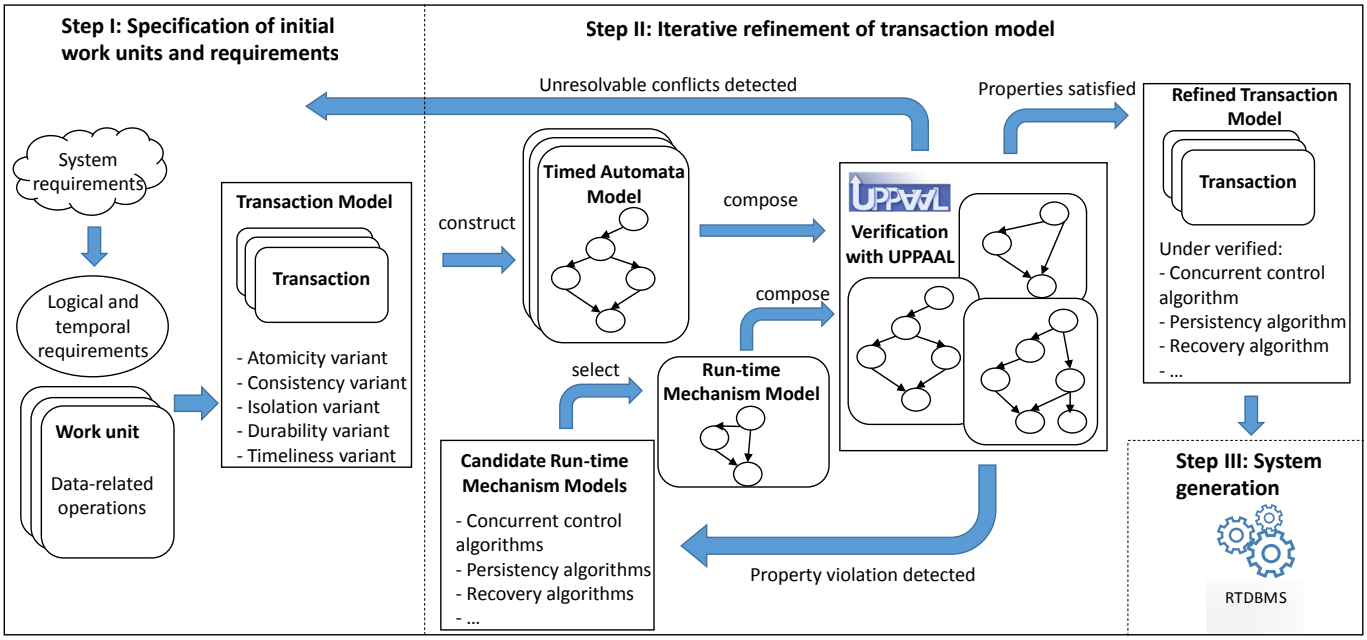

Fig. 1. The DAGGERS process

*a) **Step I: Specification of initial work units and requirements**:* The process starts with analyzing the work units, each of which is a set of operations that are logically related to achieve a business logic, as well as the logical and temporal constraints that need to be fulfilled by these work units. A work unit together with its associated constraints is called a transaction. Based on these constraints the system designer can propose the initial transaction models that can potentially achieve the requirements. A transaction model specifies the relationships between the transactions (for instance, a nested transaction model allows a parent-child relationship), as well as the ACID and timeliness variants to be ensured (for instance, SERIALIZABLE isolation and soft deadlines).

*b) **Step II: Iterative refinement of transaction model**:* We select a run-time mechanism, such as a concurrency control algorithm for isolation in the initial transaction model, from a set of candidate mechanisms. Together with the selected mechanism, we model the transactions as a set of timed automata [11], on which the ACID and timeliness properties are specified formally and can be checked by model-checking tools. If any property violation is detected, which indicates that the selected mechanism fails to meet the requirement, a new candidate mechanism is selected to replace the current one, and the model checking is restarted. This iterative process continues until all properties are satisfied by some selected mechanisms. In case that none of the run-time mechanisms in the repository can satisfy the specified properties, the designer needs to adjust the initial transaction models, by adjusting for instance the relationship between transactions, or the ACID and timeliness variants. When a new initial transaction model is decided, the refinement (Step (II)) is restarted. If the conflicts between properties cannot be resolved by any transaction model, the designer needs to adjust the requirements because they are infeasible under the current DBMS platform. When the requirements are adjusted, the entire DAGGERS process will be restarted.

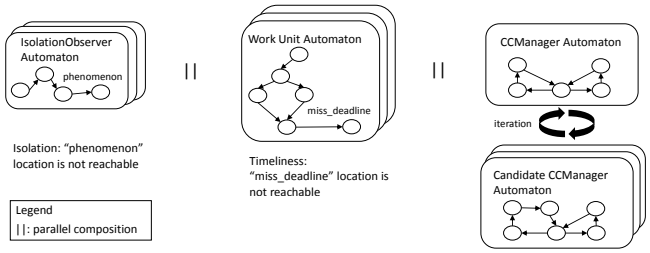

Fig. 2. Timed automata framework for analyzing isolation and timeliness

To facilitate the analysis of the transaction models in Step II, we have proposed a formal framework for modeling concurrent real-time transactions with various concurrency control mechanisms, and verifying isolation and timeliness [12].

An overview of the framework is presented in Fig. 2. We model a concurrent transaction system as a parallel composition of timed automata [11], which are finite automata extended with real-valued clock variables that measure elapsed time, and progress continuously at the same speed.

The composition consists of three types of automata: a set of **work unit** automata, a set of **IsolationObserver** automata, and a **CCManager** (Concurrency Control Manager) automaton. A work unit automaton models the work unit of a transaction as well as the interactions with the concurrency control manager. For each work unit automaton, a clock variable is defined to trace the time spent by the transaction, while a location *miss_deadline* is defined to represent the status of timeliness being breached, reached only if the clock

value exceeds a predefined deadline. An IsolationObserver automaton is created to monitor a concurrency phenomenon [10] that should be precluded by a particular isolation level [9]. If the monitored phenomenon, for instance a "lost update" occurs, the IsolationObserver will reach the location indicating the phenomenon, which also means that isolation is breached. The CCManager automaton models the concurrency control manager that applies a selected algorithm for preventing or resolving transaction conflicts. With this timed automata network, the designer can verify timeliness and isolation using model checkers such as UPPAAL [13], which is the state-of-the-art model checker for real-time systems. It supports a decidable subset of (Timed) Computation Tree Logic ((T)CTL) [14] as the specification language of properties, and provides verification of liveness and safety properties. We refer the interesting readers to literature [12].

The outcome of this step is the refined transaction models that are proved to achieve the appropriate ACID variants and timeliness under selected run-time mechanisms.

*c) **Step III: System generation**:* With the verified transaction models, the designer can implement the transactions in SQL or other programming languages, and generate the customized RTDBMS by composing or configuring the verified run-time mechanisms. In this paper, however, we do not intend to evaluate system generation.

While DAGGERS is valid for one or more ACID properties that one might need to trade off, in this paper we focus on trade-offs between isolation and timeliness, and selection of the appropriate concurrency control algorithm.

III. Case Study

In this section, we describe our case study on which we assess the applicability of the DAGGERS design process.

A. Case Study Description and Setup

Modern vehicles are equipped with dozens of sensors to monitor the vehicular and environmental states, which are analyzed by on-board diagnostic systems for on-line and off-line analysis. In our case study, these sensors are connected to a microcontroller, where the sensor data are temporarily stored, filtered and aggregated in its main memory. The sensor inputs are triggered periodically, and have strict deadlines. Since data may be accessed concurrently by updating, aggregation and other management tasks, concurrency control is needed to maintain data consistency.

Our studied system monitors 20 sensors, measuring velocity, temperature, vibration, etc, in a vehicle. The system is deployed on a AT32UC3A1512 microcontroller with a 66 MHz CPU, 512 kB flash and 65 KB data memory. The operating system is FreeRTOS, with a round-robin scheduler for tasks. All sensors have the same frequency of 5 Hz, meaning that they feed the microcontroller with data every 200 ms. A threshold is specified for each sensor to identify normal states. In case that a measurement exceeds its threshold, an error message is logged by the microcontroller, including the type of measurement and a timestamp. Every 2 seconds, the error logs are aggregated by the microcontroller, which are sent to a central computer for further analysis. The vehicle may be switched between different modes at run-time, triggered by external events. When a mode switch occurs, all thresholds should be updated for the new mode at once, without affecting the timeliness of other tasks in the microcontroller.

TABLE I
Timing measurements of individual operations

Operation	WCET	BCET
Update one SensorData record	414 μs	345 μs
Insert one ErrorLog record	2070 μs	2001 μs
Read one ErrorLog record	345 μs	276 μs
Update one Threshold record	207 μs	138 μs
Read one Threshold record	207 μs	138 μs

The architecture of the monitoring system is presented in Fig. 3. The data of each sensor is represented by a structure, *SensorData*. Thresholds are represented as an array, *Threshold*, of values. Error logs are stored in a circular list of log structures, *ErrorLog*. When a new error log should be inserted, the new data are stored in the next position of the list. The aggregated results are stored in a circular list, called *History*.

To maintain the logical data consistency, the designers decide to use an RTDBMS for managing the concurrent data access, with an appropriate lock-based concurrency control algorithm. We assume that we have a set of concurrency control algorithms to choose from, including Rigorous Two Phase Locking (R2PL), Conservative Two Phase Locking (C2PL) and short readlock algorithm [15], [16]. The first two algorithms ensure at least REPEATABLE READ isolation level, which guarantees that any data read by a transaction remains unchanged if the transaction reads the same data again before commit. The short readlock algorithm ensures a lower isolation level, READ COMMITTED, by releasing the read locks immediately after the read operations, which guarantees that only committed data are read by transactions.

The execution times of individual operations without any concurrent interference or transaction management can be measured directly on the platform, by a single task with a simple loop. These baseline measurements, including the Worst-Case Execution Time (WCET) and Best-Case Execution Time (BCET), are listed in Table I.

B. Applying the DAGGERS Process

In this subsection, we describe the design of the online monitoring subsystem following the DAGGERS process.

*1) **Step I Specification of initial work units and requirements**:* In this step we analyze the work units and their temporal and logical requirements, which allows us to propose the initial transaction models. As an example, we identify a work unit WU_n^1, which updates the sensor value in SensorData_n when new data have arrived from sensor n. Similarly, we identify others. All WU are listed in Table II.

The work units have to meet their temporal requirements, specified as follows:

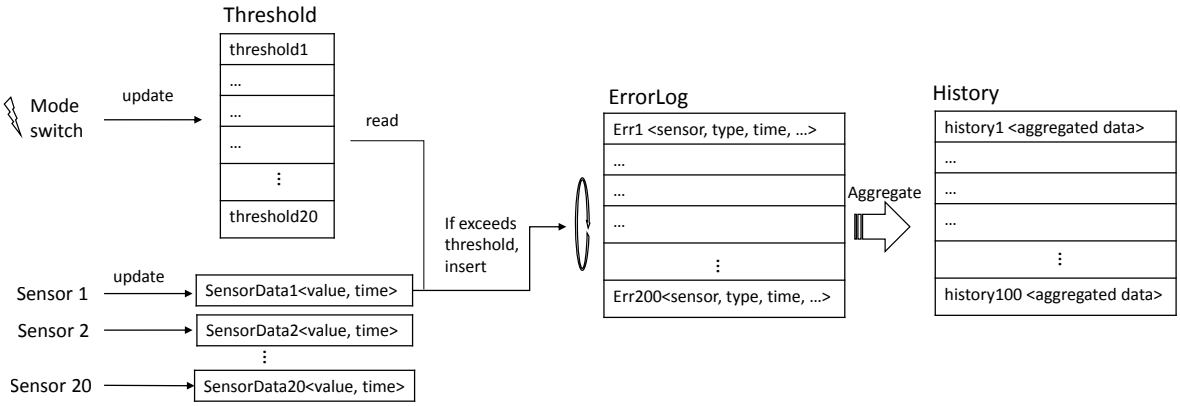

Fig. 3. Architecture of the sensor-based condition monitoring system

TABLE II
WORK UNITS IN THE ON-BOARD DIAGNOSTIC SYSTEM

WU	Description	Operations	Periods
WU_n^1	Update the SensorData_n, and compare with the threshold	update SensorData_n read Threshold_n	every 200 ms (if new data are within threshold)
WU_n^2	Update the SensorData_n, compare with the threshold, and insert a log into Error if it exceeds the threshold	update SensorData_n read Threshold_n insert a record into Error	every 200 ms (if new data exceed threshold)
WU_{reset}	Reset the values of thresholds	loop n from 1 to N update Threshold_n end loop	Triggered by a mode switch
WU_{agg}	Aggregate the Error table	loop each record in Error read record from Error end loop compute aggregation insert into History	every 2 s

TR For each sensor n, work units WU_n^1 and WU_n^2 have to meet their deadlines, which are equal to their respective periods. WU_{agg} and WU_{reset} do not have explicit deadline requirements. However, they should not cause deadline misses of the real-time work units.

Two requirements are imposed to maintain the logical data consistency. First, when a mode switch takes place, the thresholds should be updated at once. All monitored values should be compared with the thresholds from the same mode, in order to provide a consistent view of the system. Second, when the aggregation performs a sequential scan on the ErrorLog table, a sensor update work unit may overwrite a record that has not been aggregated, which may reduce the accuracy of further analysis. Therefore, it is desired that no updates should occur during the aggregation. With these considerations we conclude the following logical consistency requirements:

LR1 For each sensor n, WU_n^1 and WU_n^2 should not read uncommitted changes from WU_{reset}.

LR2 During the aggregation by WU_{agg}, the ErrorLog records should not be overwritten by WU_n^2.

TABLE III
THE INITIAL TRANSACTION MODEL

Transaction	WU	Isolation	Timeliness
T_n^1	WU_n^1	REPEATABLE READ	Deadline: $200ms$
T_n^2	WU_n^2		Deadline: $200ms$
T_{reset}	WU_{reset}		No explicit deadline
T_{agg}	WU_{agg}		No explicit deadline

In order to achieve the above temporal and logical requirements, we propose an initial transaction model to organize the work units. Our initial model is a flat transaction model, which considers each work unit in Table II as an individual transaction. The transaction model assumes totally relaxed atomicity, since the transactions in this system are not likely to be aborted. Durability is also totally relaxed as the microcontroller does not have persistent storage. The isolation level is decided to be REPEATABLE READ, since it not only precludes reading uncommitted data, but also enforces that all records in the ErrorLog remain unchanged during WU_{agg}. Table III lists the details of the initial transaction model.

*2) **Step II Iterative refinement of transaction model**:* In this following step we refine the initial transaction model via formal modeling and verification. In the studied case, we have performed three iterations for the refinement.

a) Iteration 1: We model the transactions specified in Step I as timed automata in the UPPAAL tool using the framework presented in Section II. For each transaction, a work unit automaton is created, using the automata skeletons and patterns proposed in DAGGERS [12].

As an example of the timed automata models, Fig 4 presents the automaton of transaction T_1^2, which updates data for sensor 1, reads its threshold, and inserts a new log to ErrorLog. The model is constructed by composing the instantiated locking, unlocking and data operation patterns. More specifically, this transaction starts with acquiring a write lock on SensorData_n via channel *lockwrite[SenTran1][SenData1]*. When it gets the notification of the granted lock from the transaction manager automaton via *grantwrite[SenTran1][SenData1]*, it moves to location *locked_s1*, and continues to the update operation,

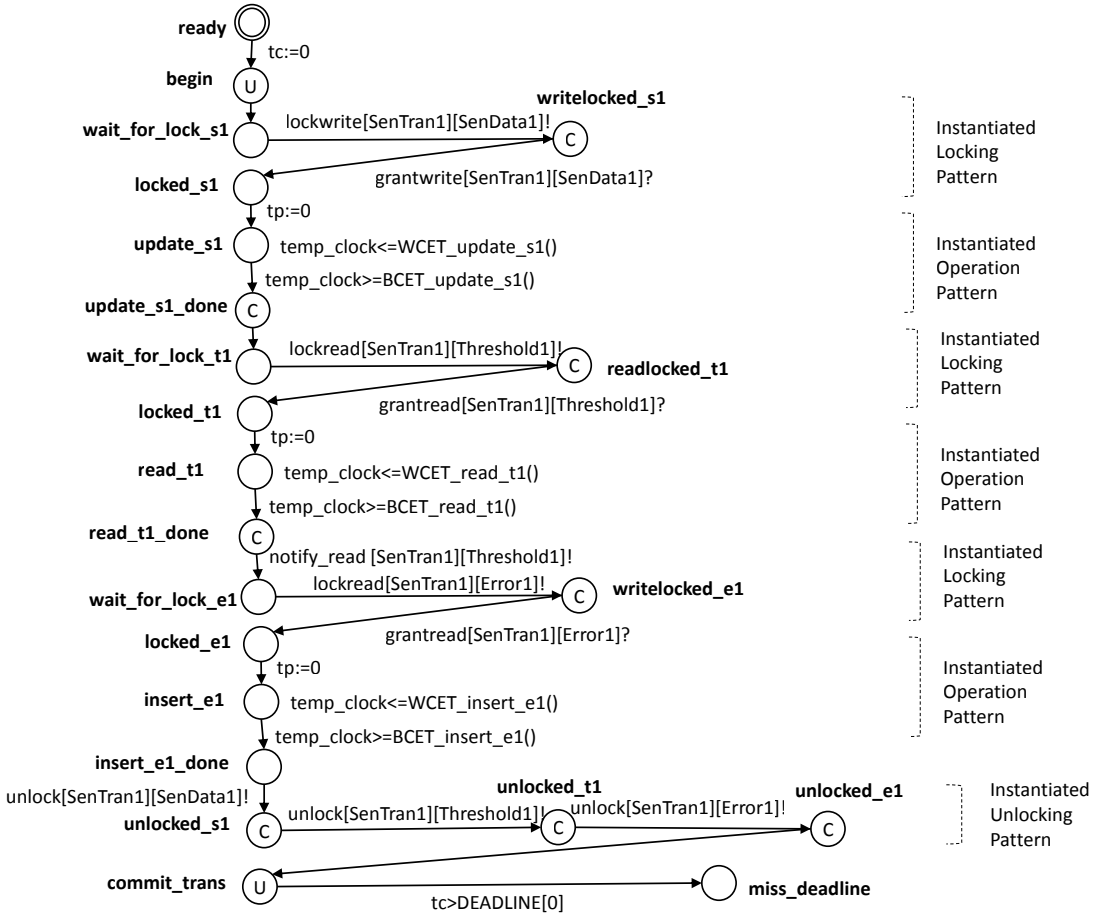

Fig. 4. Timed automaton model of T_1^2

modeled by the location *update_s1*. The automaton may stay at this location for at most *WCET_update_s1()*, representing the worst-case response time of this operation, and for at least *BCET_update_s1()*, representing the best-case response time. When this update operation is done, the automaton of T_1^2 continues to lock data Threshold_1, read Threshold_1, lock Error_1, and insert Error_1, modeled by the similar principle. Before committing, the automaton unlocks all granted locks, via the *unlock* channels. If the transaction's deadline is missed, the automaton will reach the *miss_deadline* location.

For each operation, the worst-case and best-case times are derived a priori from schedulability analysis, based on a chosen scheduling policy. For instance, in Fig 4, the function *WCET_insert_e1()* on location *insert_e1* denotes the worst case time that could be spent on inserting a row to the ErrorLog table. Since round-robin is applied as the scheduling policy, *WCET_insert_e1()* is calculated as:

WCET_WRITE_ERROR_ROW + QUANTUM *
OTHER_TRANSACTIONS *
(WCET_WRITE_ERROR_ROW / QUANTUM),

in which WCET_WRITE_ERROR_ROW is the WCET of writing one row without interference, QUANTUM is the time-slice assigned to each transaction by the operating system, and OTHER_TRANSACTIONS is the maximum number of other concurrent transactions in the system. Other timing parameters are calculated in a similar way.

A set of IsolationObservers are created to capture the phenomena that should be prevented according to **LR1** and **LR2**. For instance, Fig 5 shows the IsolationObserver for detecting a dirty read phenomenon, which appears in case that T_1^2 reads the threshold data updated by T_{reset} before T_{reset} commits. Since the IsolationObserver monitors the occurrence of the operations, it will reach the *DirtyRead* location when the particular sequence of operations of this phenomenon are performed.

These work unit and IsolationObserver models are composed with the model of a candidate concurrency control manager that ensures REPEATABLE READ. In the first iteration we choose R2PL as the concurrency control algorithm from the pool of existing algorithms. By R2PL, transactions acquire read and write locks in their growing phase before the read and write operations on data, respectively, and release all locks when they commit. If two concurrent transactions try to lock the same data, and one of them acquires a write lock, a conflict occurs and the later transaction needs to wait. The CCManager

TABLE IV
VERIFICATION RESULTS USING R2PL

ID	Specification	Verification Time	Explored States	Result
S1.1	$A[]\, not\, T\_1\_2.miss\_deadline$	0.515 s	55667	Not Satisfied
S2.1	$A[]\, not IsolationObserver1.$ $DirtyRead$	1.529 s	169210	Satisfied
S2.2	$A[]\, not IsolationObserver2.$ $InconsistentAggResult$	1.576 s	169210	Satisfied

TABLE V
VERIFICATION RESULTS USING SHORT READLOCK

ID	Specification	Verification Time	Explored States	Result
S1.1	$A[]\, not\, T\_1\_2.miss\_deadline$	4.524 s	480538	Satisfied
S2.1	$A[]\, not IsolationObserver1.$ $DirtyRead$	4.587 s	480538	Satisfied
S2.2	$A[]\, not IsolationObserver2.$ $InconsistentAggResult$	0.405 s	39105	Not Satisfied

TABLE VI
THE REFINED TRANSACTION MODEL

Transaction	WU	Isolation	Timeliness
T_n^1	WU_n^1	READ COMMITTED	Deadline: $200ms$
T_n^2	WU_n^2		Deadline: $200ms$
T_{reset}	WU_{reset}		No explicit deadline
T_{agg}	WU_{agg}		No explicit deadline

automaton models the lock resolution policy, as shown in Fig 6. The timing of lock-related actions is not modeled here since it is not significant compared with the reading and writing of data.

To verify the timeliness property imposed by **TR**, and the isolation property by **LR1** and **LR2**, we model check the timed automata network, against the specification S1 and S2 using UPPAAL model checker, respectively. The verification results are presented in Table IV. S1.1 specifies that transaction T_1^2 will never miss its deadline. For any other sensor n, the timeliness specification is basically the same. S2.1 and S2.2 specify the absence of dirty read and inconsistent aggregation result, respectively. The results show that while isolation is guaranteed, transactions may miss their deadlines. The traces given by the model checker show that the long blocking time caused by T_{agg} on table Error may result in the breached timeliness of T_1^2. In order to resolve this conflict, another iteration is necessary.

b) Iteration 2: We replace the Rigorous 2PL with another candidate concurrency control algorithm called Conservative 2PL that also ensures REPEATABLE READ isolation. The timed automata models built in Iteration 1 are adjusted to model Conservative 2PL, and the properties are checked in the same way. The results show, still, that transaction timeliness could be breached due to long blocking time.

c) Iteration 3: Since neither of the concurrency control algorithms provided for REPEATABLE READ is able to achieve the desired timeliness requirement, the proposed transaction model is not feasible under the current platform. In addition, we cannot find an alternative transaction model without changing the requirements. Therefore, we can conclude at this stage that the current requirements are not feasible.

In order to continue the system design, we revise the system requirements, and realize that LR2 can be relaxed. Even though a sensor value is overwritten before the aggregation finishes, this new value will soon be processed in the next aggregation, which will not affect the error trend analysis. Relaxing this requirement, on the other hand, allows us to relax the isolation guarantee, which may reduce the blocking on the ErrorLog table and improve the timeliness.

With LR2 being relaxed, the adjusted transaction model, instead of REPEATABLE READ, requires READ COMMITTED isolation, for which we select the short readlock concurrency control algorithm. The new models are easily adapted from the R2PL models, as shown in our previous work [12]. Verification results in Table V show that both READ COMMITTED isolation and timeliness are satisfied. As expected, inconsistent aggregation results may occur, but they are not regarded as anomaly.

As the outcome of this iteration, also the outcome of this step, we get the refined transaction model, presented in Table VI. The selected concurrency control algorithm is the short readlock algorithm.

*3) **Step III System generation**:* We have implemented the transaction models from the previous step in a prototype system on the aforementioned AT32UC3A1512 board. Every transaction in Table VI is implemented as one task, controlled by the short readlock algorithm verified in step II. The experiment results are presented in Section IV.

IV. RESULTS

We run the implemented prototype system for 10000 rounds, and record the worst-case response times, deadline miss ratio and the occurrence of dirty reads. For comparison, we have also implemented a common design with coarse-grained semaphores as our baseline system, and a system using R2PL concurrency control. The results, presented in Table VII, demonstrate that in the system implemented with the short readlock algorithm, all real-time transactions have met their deadlines, and no dirty reads precluded by READ UNCOMMITTED have occurred. This confirms our research hypothesis that the concurrency control algorithm selected by DAGGERS ensures the desired isolation level, while preserving the timeliness of the real-time transactions.

V. RELATED WORK

Customization of transaction management systems have attracted much attention from the research community. Some of these works have been applied in industrially relevant case studies. Gallina [17] proposes a software product line-oriented process called PRISMA for derivation and analysis of flexible transaction models, and applied it to a banking use case. Khachana et al. [18] propose an approach for

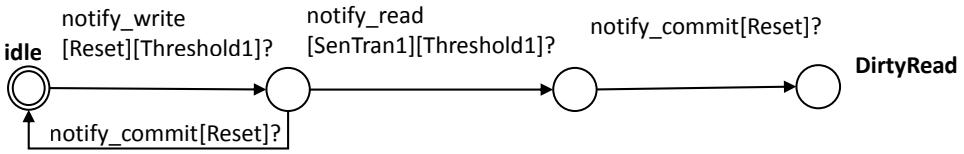

Fig. 5. Timed automaton model of IsolationObserver for DirtyRead phenomenon

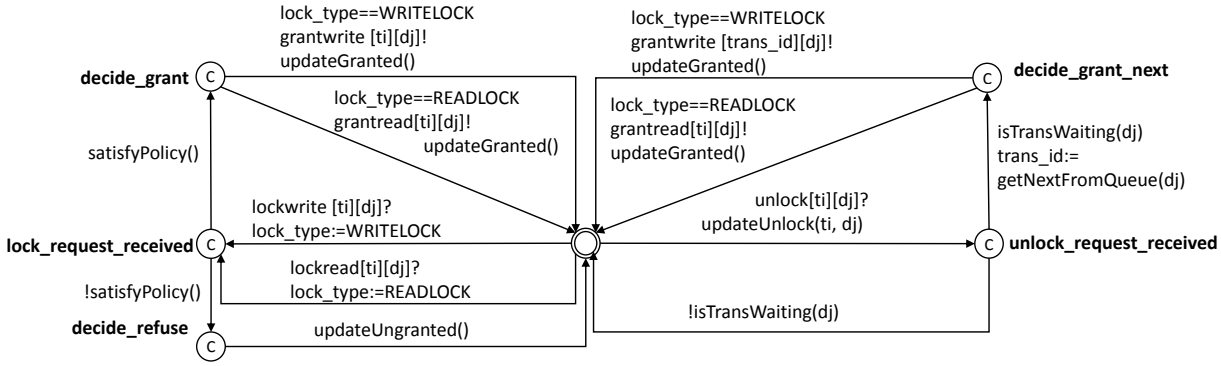

Fig. 6. Timed automaton model of the concurrency control manager for R2PL

TABLE VII
TIMING MEASUREMENTS OF THE IMPLEMENTED SYSTEM

Concurrency Control	WCRT	Deadline Miss Ratio	Number of Dirty Reads
Short readlock (selected by DAGGERS)	15.594 ms	0	0
Baseline	445.740 ms	0.08 %	0
Rigorous 2PL	1577.133 ms	1.37%	0

relaxation of ACID properties and tested it on a travel agency scenario. Various formalisms have been proposed to model transactions, such as first order logics in ACTA [19], real-time Maude by Liu et al. [20]. Our work models transactions as timed automata, and emphasizes on trading off isolation for timeliness, which is crucial to many embedded systems.

For customized embedded database systems, Thüm et al. [21] propose a feature-oriented approach for generating customized DBMSs for automotive systems and validated their platform on an automotive testing environment. Nyström et al. [22] combine a component-based approach and aspect-oriented programing to build customized RTDBMS, and verified the approach in an embedded automotive system. Both works base their transaction management design decisions on functional requirements analysis, code-size and performance. Our case study, as a contrast, performs the customization at the transaction model level, and bases the design decisions on the trade-offs between timeliness and isolation. Formal modeling and verification of transaction models, while not applied by these existing works, are the essence of our approach.

VI. CONCLUSION

In this paper we have presented a case study conducted to evaluate the applicability of the DAGGERS process in designing customized data management for automotive systems. In the case study, a customized RTDBMS for an on-board diagnostic system is designed following the exact steps of DAGGERS. Experiments on the prototype system show that both timeliness and the desired level of isolation are satisfied.

The results show that DAGGERS can help designers to systematically derive transaction models based on iteratively analyzing trade-offs between timeliness and isolation, and select the appropriate concurrency control algorithm to guarantee both properties. The transactional properties, as well as the work units, can be identified from the system requirements. The timing parameters necessary for the formal models can be measured on the real platform, together with schedulability analysis. The abstracted behaviors of the work units and the candidate concurrency control algorithms can be modeled as timed automata, and verified using UPPAAL. Therefore, based on this case study, it is unlikely to face technical obstacles when applying the process to other automotive systems, as long as they have well-defined requirements and platforms.

However, one major obstacle might still exist: the scalability issue of the automata modeling method due to the complexity of transaction management. At first we constructed a timed automata network modeling every transaction, together with concurrency control and round-robin scheduling policy explicitly, and tried to model check the entire system together. This however led to state explosion, and the verification failed to terminate. As an improvement, we have abstracted the scheduling policy away. Instead of modeling scheduling, we calculate the worst-case times of operations using schedulability analysis, and annotate these values directly to the models.

Although this replacement of worst-case estimation introduces pessimism, it still provides a safe upper bound, and reduces the state space. This abstraction also breaks the dependency at modeling level between transactions that do not share data directly, for instance two sensor transactions. Since the indirect impacts on timing from another sensor transaction has already been counted by the worst-case impact estimation, we only need to verify one sensor transaction automaton, together with the aggregation transaction automaton, threshold reset transaction automaton, and the CCManager automaton. As a result, the abstraction greatly improves the scalability of the analysis, and thus the applicability of the DAGGERS process.

For large systems where complex dependencies exist among many transactions, our experience tells that exhaustive verification may be possible only at a very high level of abstraction. As an alternative, one may consider applying statistical model checking, which only explores a bounded state space, and provides probabilistic verification results with a confidence. Although not a completely exhaustive and exact verification, this can still provide guidance for making design decisions, especially to eliminate infeasible ones.

Several future directions can be explored to improve DAGGERS. First, more timed-automata patterns for common runtime mechanisms need to be invented in order to enrich our repository. This includes not only models of other concurrency control algorithms, but also models of common logging, persistence and recovery algorithms, for customization involving all ACID properties. Second, as we have started in this paper, we plan to explore the integration of other formal methods, for instance statistical model checking, into the process, to achieve better scalability, effectiveness and efficiency.

REFERENCES

[1] R. Hegde, G. Mishra, and K. S. Gurumurthy, *An Insight into the Hardware and Software Complexity of ECUs in Vehicles*, 2011, pp. 99–106.

[2] C. Nitsche, S. Schroedl, and W. Weiss, "Onboard diagnostics concept for fuel cell vehicles using adaptive modelling," in *IEEE Intelligent Vehicles Symposium, 2004*, June 2004, pp. 127–132.

[3] G. R. Goud, N. Sharma, K. Ramamritham, and S. Malewar, "Efficient real-time support for automotive applications: A case study," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*, 2006, pp. 335–341.

[4] S. Schulze, M. Pukall, G. Saake, T. Hoppe, and J. Dittmann, "On the need of data management in automotive systems," in *Datenbanksysteme in Business, Technologie und Web (BTW)*, 2009, pp. 217–226.

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

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

[7] S. Cai, B. Gallina, D. Nyström, and C. Seceleanu, "Trading-off data consistency for timeliness in real-time database systems," in *Proceedings of the 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 13–16.

[8] O. Benedettini, T. S. Baines, H. Lightfoot, and R. Greenough, "State-of-the-art in integrated vehicle health management," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 223, no. 2, pp. 157–170, 2009.

[9] "ISO/IEC 9075:1992 Database Language SQL," International Organization for Standardization, Standard.

[10] A. Adya, B. Liskov, and P. O'Neil, "Generalized isolation level definitions," in *Proceedings of the 16th IEEE International Conference on Data Engineering*, 2000, pp. 67–78.

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

[12] 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 International Conference on Real-Time Networks and Systems*, 2016, pp. 3–12.

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

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

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

[16] J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger, "Readings in database systems." Morgan Kaufmann Publishers Inc., 1998, ch. Granularity of Locks and Degrees of Consistency in a Shared Data Base, pp. 175–193.

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

[18] R. T. Khachana, A. James, and R. Iqbal, "Relaxation of acid properties in autra, the adaptive user-defined transaction relaxing approach," *Future Generation Computer Systems*, vol. 27, no. 1, pp. 58–66, Jan. 2011.

[19] P. Chrysanthis and K. Ramamritham, "Synthesis of extended transaction models using acta," *ACM Transactions on Database Systems (TODS)*, vol. 19, no. 3, pp. 450–491, 1994.

[20] S. Liu, P. C. Ölveczky, M. R. Rahman, J. Ganhotra, I. Gupta, and J. Meseguer, "Formal modeling and analysis of ramp transaction systems," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 1700–1707.

[21] T. Thüm, S. Schulze, M. Pukall, G. Saake, and S. Günther, "Secure and customizable data management for automotive systems: A feasibility study," *ISRN Software Engineering*, 2012.

[22] D. Nyström, A. Tešanovic, M. Nolin, C. Norström, and J. Hansson, "Comet: a component-based real-time database for automotive systems," in *Proceedings of the Workshop on Software Engineering for Automotive Systems*, 2004, pp. 1–8.

[23] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.