Systematic Design and Analysis of Customized Data Management for Real-Time Database Systems

Simin Cai
SYSTEMATIC DESIGN AND ANALYSIS OF CUSTOMIZED DATA MANAGEMENT FOR REAL-TIME DATABASE SYSTEMS

Simin Cai

2019
SYSTEMATIC DESIGN AND ANALYSIS OF CUSTOMIZED DATA MANAGEMENT FOR REAL-TIME DATABASE SYSTEMS

Simin Cai

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid Akademin för innovation, design och teknik kommer att offentligen försvaras måndagen den 4 november 2019, 13.30 i Gamma, Mälardalens högskola, Västerås.

Fakultetsopponent: Professor Marieke Huisman, University of Twente

MÄLARDALEN UNIVERSITY SWEDEN

Akademin för innovation, design och teknik
Abstract

Modern real-time data-intensive systems generate large amounts of data that are processed using complex data-related computations such as data aggregation. In order to maintain logical data consistency and temporal correctness of the computations, one solution is to model the latter as transactions and manage them using a Real-Time Database Management System (RTDBMS). Ideally, depending on the particular system, the transactions are customized with the desired logical and temporal correctness properties, which should be enforced by the customized RTDBMS via appropriate transaction management mechanisms. However, developing such a data management solution with high assurance is not easy, partly due to inadequate support for systematic specification and analysis during the design. Firstly, designers do not have means to identify the characteristics of the computations, especially data aggregation, and to reason about their implications. Design flaws might not be discovered early enough, and thus they may propagate to the implementation. Secondly, meeting more properties simultaneously might not be possible, so trading-off the less critical ones for the critical one, for instance, temporal correctness, is sometimes required. Nevertheless, trade-off analysis of conflicting properties, such as transaction atomicity, isolation and temporal correctness, is mainly performed ad-hoc, which increases the risk of unpredictable behavior.

In this thesis, we address the above problems by showing how to systematically design and provide assurance of transaction-based data management with data aggregation support, customized for real-time systems. We propose a design process as our methodology for the systematic design and analysis of the trade-offs between desired properties, which is facilitated by a series of modeling and analysis techniques. Our design process consists of three major steps as follows: (i) Specifying the data-related computations, as well as the logical data consistency and temporal correctness properties, from system requirements, (ii) Selecting the appropriate transaction models to model the computations, and deciding the corresponding transaction management mechanisms that can guarantee the properties, via formal analysis, and, (iii) Generating the customized RTDBMS with the proved transaction management mechanisms, via configuration or implementation. In order to support the first step of our process, we propose a taxonomy of data aggregation processes for identifying their common and variable characteristics, based on which their inter-dependencies can be captured, and the consequent design implications can be reasoned about. Tool support is provided to check the consistency of the data aggregation design specifications. To specify transaction atomicity, isolation and temporal correctness, as well as the transaction management mechanisms, we also propose a Unified Modeling Language (UML) profile with explicit support for these elements. The second step of our process relies on the systematic analysis of trade-offs between transaction atomicity, isolation and temporal correctness. To achieve this, we propose two formal frameworks for modeling transactions with abort recovery, concurrency control, and scheduling. The first framework UPPCART utilizes timed automata as the underlying formalism, based on which the desired properties can be verified by model checking. The second framework UPPCART-SMC models the system as stochastic timed automata, which allows for probabilistic analysis of the properties for large complex RTDBMS using statistical model checking. The encoding of high-level UTRAN specifications into corresponding formal models is supported by tool automation, which we also propose in this thesis. The applicability and usefulness of our proposed techniques are validated via several industrial use cases focusing on real-time data management.
Abstract

Modern real-time data-intensive systems generate large amounts of data that are processed using complex data-related computations such as data aggregation. In order to maintain logical data consistency and temporal correctness of the computations, one solution is to model the latter as transactions and manage them using a Real-Time Database Management System (RTDBMS). Ideally, depending on the particular system, the transactions are customized with the desired logical and temporal correctness properties, which should be enforced by the customized RTDBMS via appropriate transaction management mechanisms. However, developing such a data management solution with high assurance is not easy, partly due to inadequate support for systematic specification and analysis during the design. Firstly, designers do not have means to identify the characteristics of the computations, especially data aggregation, and to reason about their implications. Design flaws might not be discovered early enough, and thus they may propagate to the implementation. Secondly, meeting more properties simultaneously might not be possible, so trading-off the less critical ones for the critical one, for instance, temporal correctness, is sometimes required. Nevertheless, trade-off analysis of conflicting properties, such as transaction atomicity, isolation and temporal correctness, is mainly performed ad-hoc, which increases the risk of unpredictable behavior.

In this thesis, we address the above problems by showing how to systematically design and provide assurance of transaction-based data management with data aggregation support, customized for real-time systems. We propose a design process as our methodology for the systematic design and analysis of the trade-offs between desired properties, which is facilitated by a series of modeling and analysis techniques. Our design process consists of three major steps as follows: (i) Specifying the data-related computations, as well as the logical data consistency and temporal correctness properties, from system requirements, (ii) Selecting the appropriate transaction models to model the
computations, and deciding the corresponding transaction management mechanisms that can guarantee the properties, via formal analysis, and, (iii) Generating the customized RTDBMS with the proved transaction management mechanisms, via configuration or implementation. In order to support the first step of our process, we propose a taxonomy of data aggregation processes for identifying their common and variable characteristics, based on which their interdependencies can be captured, and the consequent design implications can be reasoned about. Tool support is provided to check the consistency of the data aggregation design specifications. To specify transaction atomicity, isolation and temporal correctness, as well as the transaction management mechanisms, we also propose a Unified Modeling Language (UML) profile with explicit support for these elements. The second step of our process relies on the systematic analysis of trade-offs between transaction atomicity, isolation and temporal correctness. To achieve this, we propose two formal frameworks for modeling transactions with abort recovery, concurrency control, and scheduling. The first framework UPPCART utilizes timed automata as the underlying formalism, based on which the desired properties can be verified by model checking. The second framework UPPCART-SMC models the system as stochastic timed automata, which allows for probabilistic analysis of the properties for large complex RTDBMS using statistical model checking. The encoding of high-level UTRAN specifications into corresponding formal models is supported by tool automation, which we also propose in this thesis. The applicability and usefulness of our proposed techniques are validated via several industrial use cases focusing on customized real-time data management.
致我的父亲母亲

To my parents
吾生也有涯，而知也无涯。
以有涯随无涯，何如？

— 庄子·内篇·养生主

My life has an end.
The universe of knowledge has no end.
How would it be,
to pursue the endless knowledge with a limited life?

— Zhuangzi
Acknowledgments

Many people have helped and accompanied me during this PhD study. First of all, I would like to thank my supervisors, Associate Professor Cristina Secleanu, Associate Professor Barbara Gallina, Dr. Dag Nyström, for your guidance and support during the entire journey. Besides the supervision on my research, which you have dedicated tremendous effort to, your care and your optimism play equally important roles in helping me to have come so far. I am also extremely happy that we have established a close friendship and witnessed important moments in each other’s lives during these years.

I would like to express my deep gratitude to the faculty examiner Professor Marieke Huisman, and the thesis grading committee members: Associate Professor Enrico Bini, Professor Magnus Jonsson, and Associate Professor Dragoș Truşcan. It is my honor to have you as the reviewers of this thesis.

My PhD study in MDH has been a great experience, thanks to the knowledgeable professors and lecturers, the helpful administrative staff, and the lovely fellow PhD students. Friendships in MDH have not only brought me with joy but also courage, such that I could continue in the darker nights when facing a seemingly endless tunnel. I also would like to thank my friends and colleagues in Mimer for the valuable support. Many thanks to my friends: Yemao, Xueming, Wei, Tengjiao, Nico, Anders, and more. Specially, I would like to thank Fredrik for your company, and for taking all my complaints during these years. Together, we have finished four theses.

Any acknowledgment notes without acknowledging my parents are incomplete. All my life, they have prioritized me in everything.

The Knowledge Foundation of Sweden and the Swedish Research Council are gratefully acknowledged for funding the work of this thesis.

Simin Cai
Västerås, September, 2019
List of Publications

Papers Included in the Thesis


1The included papers have been reformatted to comply with the thesis layout

**Paper F**  

**Papers Related to, but not Included in the Thesis**


Paper not Related to the Thesis


Licentiate Thesis

Contents

I  Thesis  1

1  Introduction  3
   1.1  Thesis Overview  ........................................  8

2  Preliminaries  13
   2.1  Data Aggregation  ....................................  13
   2.2  Transactions and Customized Transaction Management  14
      2.2.1  Relaxed ACID Properties  ............................  16
      2.2.2  Atomicity Variants and Abort Recovery Mechanisms  17
      2.2.3  Isolation Variants and Pessimistic Concurrency Control Algorithms  18
      2.2.4  Real-time Transactions and Temporal Correctness  20
      2.2.5  Customized Transaction Management in DBMS  21
   2.3  Formal Methods Applied in this Thesis  ......................  23
      2.3.1  Boolean Satisfiability Problem Solving Using Z3  24
      2.3.2  Model Checking Using UPPAAL  ..........................  24
      2.3.3  Statistical Model Checking Using UPPAAL SMC  28

3  Research Problem  31
   3.1  Problem Description  ......................................  31
   3.2  Research Goals  ..........................................  32

4  Research Methods  35

5  Thesis Contributions  37
   5.1  The DAGGERS Process: Data AGGregation for Embedded Real-time Systems  37
   5.2  DAGGTAX: Data AGGregation TAXonomy  ..........................  39

xiii
### Contents

5.3 SAFARE: SAt-based Feature-oriented Data Aggregation Design 41  
5.4 The UTRAN Profile: UML for TRANsactions ................. 43  
5.5 The UPPCART Framework: UPPaal for Concurrent Atomic  
Real-time Transactions ........................................ 45  
5.5.1 Modeling RTDBMS as a Network of Timed Automata 45  
5.5.2 Pattern-based Construction .............................. 46  
5.5.3 Verification .............................................. 52  
5.6 The UPPCART-SMC Framework .............................. 53  
5.7 Validation on Industrial Use Cases ......................... 56  
5.8 Research Goals Revisited .................................. 60  

6 Related Work .................................................. 63  
6.1 Design Methodologies for Customized Transaction Management 63  
6.2 Taxonomies of Data Aggregation ............................ 65  
6.3 Formal Specification and Analysis of Transactions .......... 66  

7 Conclusions and Future Work ................................ 69  
7.1 Limitations .................................................. 70  
7.2 Future Work ................................................ 71  

Bibliography ..................................................... 73  

II Included Papers .................................................. 85  

8 Paper A: 
Data Aggregation Processes: A Survey, A Taxonomy, and Design  
Guidelines ....................................................... 87  
8.1 Introduction .................................................. 89  
8.2 Background ................................................... 91  
8.2.1 Timeliness and Temporal Data Consistency .......... 91  
8.2.2 Feature Model and Feature Diagram ................. 93  
8.3 A Survey of Data Aggregation Processes ................. 94  
8.3.1 General-purpose Data Management Infrastructures .. 94  
8.3.2 Smart “X”, IoT and Big Data Applications ........... 97  
8.3.3 Ad Hoc Applications ................................. 98  
8.3.4 Survey Results .................................. 101  
8.4 Our Proposed Taxonomy ..................................... 104  
8.5 Design Constraints and Heuristics for Data Aggregation Processes ........................................ 112
9 Paper B:
Tool-Supported Design of Data Aggregation Processes in Cloud Monitoring Systems

9.1 Introduction ........................................... 135
9.2 Background .......................................... 137
  9.2.1 Feature Model .................................. 137
  9.2.2 The DAGGTAX Taxonomy ...................... 138
  9.2.3 Boolean Satisfiability and Z3 .................. 140
9.3 Formal Specification and Analysis of Data Aggregation Processes .................. 141
  9.3.1 Formalizing DAP Specifications ............... 141
  9.3.2 Formalizing Intra-DAP Constraints .......... 143
  9.3.3 Formalizing Inter-DAP Constraints .......... 144
  9.3.4 Formalizing Data Management Design Constraints ........................................... 144
9.4 The SAFARE Tool .................................... 146
  9.4.1 DAP Specification .............................. 148
  9.4.2 Consistency Check .............................. 149
  9.4.3 DBMS Specification .............................. 151
9.5 Case Study and Results ............................. 152
  9.5.1 Case Study Description ......................... 152
  9.5.2 Application of DAGGTAX and SAFARE ........ 154
  9.5.3 System Implementation ......................... 157
9.6 Discussion .......................................... 158
9.7 Related Work ........................................ 159
9.8 Conclusions and Future Work ....................... 161
Bibliography ............................................. 163
10 Paper C: Towards the Verification of Temporal Data Consistency in Real-Time Data Management 167
10.1 Introduction ............................................. 169
10.2 Background ............................................. 170
  10.2.1 Temporal Data Consistency ......................... 170
  10.2.2 Timed Automata and UPPAAL ....................... 171
10.3 Assumed System ........................................ 173
10.4 Modeling Transaction Work Units and Data ............. 174
  10.4.1 Modeling Transaction Work Units ................. 175
  10.4.2 Modeling the Age of Data ......................... 177
  10.4.3 Modeling the Lock Manager ......................... 177
10.5 Verification of Temporal Data Consistency and Timeliness ............................................. 179
  10.5.1 Formalizing the Requirements ....................... 180
  10.5.2 Verification Results ............................... 181
10.6 Related Work ......................................... 183
10.7 Conclusion ............................................. 183
Bibliography .................................................. 185

11 Paper D: A Formal Approach for Flexible Modeling and Analysis of Transaction Timeliness and Isolation 187
11.1 Introduction ............................................. 189
11.2 Preliminaries .......................................... 190
  11.2.1 The Concept of Transaction ......................... 190
  11.2.2 Isolation ............................................ 192
  11.2.3 Pessimistic Concurrency Control (PCC) ............. 193
  11.2.4 Timed Automata and UPPAAL ......................... 194
11.3 Our Approach .......................................... 196
  11.3.1 Work Unit Skeleton and Operation Patterns ........ 197
  11.3.2 Concurrency Control Skeleton and Patterns ........ 200
  11.3.3 IsolationObserver Skeleton ......................... 202
  11.3.4 Reference Algorithm: Rigorous 2PL ................ 203
11.4 Adjustments for Various PCC .......................... 205
  11.4.1 Concurrency Control for Relaxed Isolation .......... 205
  11.4.2 Real-time Concurrency Control ....................... 207
11.5 Verification ............................................. 210
11.6 Related Work .......................................... 214
11.7 Conclusion .................................. 216
Bibliography .................................. 219

12 Paper E: Specification and Automated Verification of Atomic Concurrent Real-Time Transactions 223
12.1 Introduction ................................. 225
12.2 Preliminaries ................................. 228
  12.2.1 Real-Time Transactions .......................... 228
  12.2.2 UML Profiles and MARTE ......................... 231
  12.2.3 UPPAAL Timed Automata and UPPAAL Model Checker .......................... 231
12.3 UTRAN ..................................... 233
  12.3.1 Domain View ............................... 234
  12.3.2 Profile .................................. 236
12.4 UPPCART framework ......................... 239
  12.4.1 Patterns and Connectors for Modeling Work Units .............. 242
  12.4.2 Patterns and Connectors for Modeling TransactionSequence .................. 253
  12.4.3 CCManager Skeleton (CCS) ...................... 256
  12.4.4 ATManager Skeleton (ATS) ...................... 258
  12.4.5 IsolationObserver Skeleton (IOS) .................. 259
  12.4.6 Data Skeleton (DS) ........................... 260
  12.4.7 Summary of Modeling ......................... 260
  12.4.8 Verification ............................... 261
12.5 From UTRAN to UPPCART .................... 262
  12.5.1 Translational Semantics of UTRAN .................. 262
  12.5.2 Automated Transformation ....................... 265
  12.5.3 Validation of U²Transformer ...................... 269
12.6 Case Study ................................. 270
  12.6.1 Global Collision Avoidance Layer .................. 271
  12.6.2 Local Collision Avoidance Layer .................. 275
12.7 Related Work ................................. 277
12.8 Conclusions and Future Work .................... 281
Bibliography .................................. 283
13.1 Introduction .......................................................... 291
13.2 Background ......................................................... 292
  13.2.1 UPPAAL Timed Automata and UPPAAL Model Checker .......................................................... 292
  13.2.2 UPPAAL Stochastic Timed Automata and UPPAAL SMC ......................................................... 293
  13.2.3 The UPPCART Framework ........................................ 294
  13.2.4 UTRAN ................................................................. 296
13.3 The UPPCART-SMC Framework ...................................... 298
  13.3.1 UPPCART-SMC Patterns ......................................... 299
  13.3.2 Statistical Analysis ................................................ 301
13.4 A Case Study ............................................................ 303
  13.4.1 System Description ................................................. 303
  13.4.2 UPPAAL SMC Models ............................................. 308
  13.4.3 Analysis .............................................................. 308
13.5 Related Work .......................................................... 309
13.6 Conclusions and Future Work ....................................... 311
Bibliography ................................................................. 313
I

Thesis
Chapter 1

Introduction

With the development of cyber-physical systems and Internet of Things in the last couple of decades, real-time systems are becoming more data-intensive than ever. For instance, modern automotive systems may process thousands of signals in real-time [1], and are evolving towards more adaptive software relying on large amounts of ambient and configuration data from various services [2]. In the popular Industry 4.0 paradigm [3], production processes are designed, executed and optimized, based on complex and timely analysis of sensor, production and configuration data from various layers. Not only the amounts of data are growing, but the delivery of correct services of real-time systems are also depending more on the effective and efficient management of data. On one hand, timely access to fresh data such as environmental status are essential for many safety-critical systems, in which the temporal correctness of computations is crucial for avoiding disastrous behaviors. On the other hand, as many decisions and operations involve processing multiple data items, maintaining logical data consistency plays a fundamental role for achieving the logical correctness of the decisions.

Amid the increasing importance of data and complexity of data-related computations, Real-Time DataBase Management Systems (RTDBMS) have been considered as a promising data management technique to enhance the safety and reliability of real-time data-intensive systems [4]. In addition to benefits such as common data access interface and separation of data from computation, an RTDBMS contributes to maintain logical data consistency, by managing the data-related computations as transactions, via a variety of mechanisms. As a goal, logical data consistency is provided by ensuring the so-called
ACID properties, which refer to, respectively: Atomicity (a transaction either runs completely or makes no changes at all), Consistency (a transaction executing by itself does not violate logical constraints), Isolation (uncommitted changes of one transaction should not be seen by concurrent transactions), and Durability (committed changes are made permanent) [5, 6]. As for the real-time aspect, an RTDBMS maintains temporal correctness by ensuring transaction timeliness and temporal data consistency. Timeliness refers to the property that the transaction should complete its computation by the specified deadline, while temporal data consistency requires that the data used for the computation should represent a fresh and consistent view of the system and the environment [4, 7]. Various transaction management mechanisms are employed by an RTDBMS to ensure logical data consistency and temporal correctness. For instance, abort recovery mechanisms are applied to achieve atomicity [8], concurrency control mechanisms are adopted for providing isolation [9], while various scheduling policies are combined with the former two to ensure temporal correctness [10].

Since real-time systems have exhibited large varieties in their resource limits, functionalities, as well as dependability requirements that entail logical and temporal correctness concerns, researchers have long discovered that a “one-size-fits-all” DBMS for all real-time applications is not feasible, especially due to the conflicts among the above constraints [11]. A dependable data management solution should be a DBMS customized with appropriate trade-offs between the conflicting constraints, stemming from the application requirements. In this thesis, we focus on the trade-offs between the assurance of ACID and temporal correctness imposed by the application semantics. Ideally, both ACID and temporal correctness should be guaranteed by the DBMS. However, conflicts among the desired properties could arise. A typical scenario occurs especially when temporal correctness is breached, due to the unpredictability introduced by the transaction management mechanisms. For instance, concurrency control may introduce long blocking time for concurrent transactions requiring access to the shared data [12]. Some concurrency control algorithms may even abort transactions in conflict, which triggers the recovery mechanism that further adds to unpredictability [13]. Therefore, for many real-time applications, ACID assurance is often relaxed in favor of temporal correctness [11], which entails a customized DBMS with selected transaction management mechanisms to provide the intended assurance.

While most existing DBMS products are indeed customizable, the real challenge lies in how to make the right choice from a number of transaction management customization options. For instance, customization of isolation
assurance is often provided by most commercial DBMS as a configuration of isolation levels, backed by a series of concurrency control algorithms. Atomicity variations can be achieved by selecting corresponding recovery options, which can alternatively be explicitly encoded in the transaction code by the developer. Software-product-line-based techniques also allow a DBMS to be constructed with selected transaction management components and aspects, hence embracing larger freedom of customization. However, these products and proposals do not answer the critical questions: How should the system designer select the customized transaction management mechanisms, such that the desired ACID and temporal correctness trade-offs are ensured? Despite the existence of a number of DBMS design methodologies [14, 15, 16, 17, 18, 19, 20], few of them provide sufficient support for the systematic analysis of trade-offs between ACID and temporal correctness, which should guide the decisions on the appropriate mechanisms. As common practice, the decisions are made by designers upon experience or pilot experiments, while no guarantee can be provided on whether or not the traded-off properties are satisfied by the selected mechanisms. Consequently, breached temporal correctness or unnecessary relaxation of ACID could arise, which may lead to severe safety risks.

This thesis investigates how to design customized transaction management of DBMS solutions for real-time database-centric systems systematically. We believe the foundation to this is to describe and reason about the high-level characteristics of the application’s data-related computations, especially those related to logical data consistency and temporal correctness, in a methodical manner. We are particularly interested in one type of such computations, called data aggregation, which is the process of producing a synthesized form of data from multiple data items using an aggregate function [21], and is commonly applied in most real-time systems [22, 23, 24]. In many data-intensive systems, the aggregated data of one aggregation process could serve as the raw data of another, hence forming a multiple-level aggregation design. Each level of aggregation may have its unique characteristics, not only in the functional aspects of specific aggregate functions, but also in their logical and temporal correctness constraints. Despite the importance and popularity of data aggregation, knowledge of the characteristics of this computation is inadequate for the systematic analysis of the mentioned properties and customized transaction management design.

With this knowledge of the application semantics, the next step is to reason about the trade-offs between ACID and temporal correctness, and select the appropriate transaction management mechanisms. We focus on analyzing trade-offs between atomicity, isolation and temporal correctness, and the
selection of abort recovery mechanisms, concurrency control algorithms and scheduling policies. We select these properties because their mechanisms may heavily influence the assurance of each other, while existing techniques do not provide adequate support for all these properties and mechanisms [25, 26]. A challenge arises in how to formulate the transactions’ behaviors under various mechanisms and the desired properties, such that they can be rigorously analyzed with the state-of-art methods and tools.

Our work addresses the above issues, by showing how to systematically design a customized and dependable RTDBMS for real-time systems with data aggregation, with assured trade-offs between ACID and temporal correctness. We achieve this by proposing a design process as our methodology, called DAGGERS (Data AGGregation for Embedded Real-time Systems) [27], which guides the selection of ACID and temporal correctness trade-offs, as well as the concrete mechanisms to provide property assurance within the process. The DAGGERS process consists of the following steps: (i) Specifying the data aggregation processes and other data-related computations, as well as the logical data consistency and temporal correctness properties, from system requirements, (ii) Selecting the appropriate transaction models to model the computations, and deciding the corresponding transaction management mechanisms that can guarantee the properties, via formal analysis, and, (iii) Realizing the customized RTDBMS, using the decided transaction models and mechanisms that are proven to assure the desired properties, and the customization primitives provided by the particular RTDBMS platform.

We propose a set of modeling and analysis techniques to facilitate the DAGGERS process, with emphasis on the first and second steps. Our proposed techniques rely on a set of formal methods, which create mathematical representations of the studied systems and properties, and provide a high degree of assurance via rigorous analysis. For the first step, to better understand and analyze the application semantics, we propose a taxonomy called DAGGTAX (Data AGGregation TAXonomy) to characterize the features of data aggregation processes at high level [28, 29], and a technique to formally detect the violations among the features at early design stages [30]. DAGGTAX-based specification of data aggregation processes, as well as the automatic detection of violations, are implemented in our SAFARE (SAt-based Feature-oriented dAta aggREgation design) tool [30].

To enable explicit specification of logical data consistency and temporal correctness properties, and facilitate intuitive analysis for designers in a high level, we propose a Unified Modeling Language (UML) profile, called UTRAN (UML for TRANsactions) [26]. This profile offers elements for specifying
transactions and transactional properties in the first step of DAGGERS, focusing on atomicity, isolation and temporal correctness, as well as their corresponding RTDBMS mechanisms.

During the second step of DAGGERS, one needs to derive the appropriate trade-offs between ACID and temporal correctness, and select the corresponding transaction management mechanisms from a set of candidate variants. To support this, we propose a formal framework based on UPPAAL Timed Automata (TA) [31], called UPPCART (UPPAAL for Concurrent Atomic Real-time Transactions), to specify and model the transactions with various abort recovery, concurrency control, and scheduling mechanisms [32, 33, 26]. The desired atomicity, isolation and temporal correctness properties of the UPPCART models can then be verified by model checking with the state-of-the-art UPPAAL model checker [31]. We provide a set of automata patterns as basic units for modeling the transactions and the mechanisms, which not only tame the complexity of the models, but also allow them to be reused during the iterative selection and analysis of various mechanisms. We also propose a tool, called $U^2$Transformer, for the translation from high-level UTRAN specifications into UPPCART models. The tool shields the designers from the formal underpinning, and reduces the efforts on the formal modeling of the system.

In order to analyze larger scale DBMS, we propose a formal framework UPPCART-SMC [34], which extends UPPCART with the capability of statistical model checking. In this framework, we model the transactions and the transaction management mechanisms as Stochastic Timed Automata (STA) [35], and verify the probabilities of atomicity, isolation and temporal correctness using UPPAAL SMC [35], the extension of UPPAAL for statistical model checking. UPPAAL SMC can answer questions on whether the probability of property satisfaction meets a provided threshold, for instance, whether the probability of ensuring isolation is higher than 99.9%. It can also perform probability estimation (e.g., what is the probability that some property holds?) that returns an approximation interval for that probability, with a given confidence. Such analysis helps designers to evaluate the appropriateness of the customization choices.

We validate our proposed techniques and tools by demonstrating their applicability via a series of industrial use cases, in which customized real-time data management plays an essential role. Among the validations, DAGGTX and SAFARE are demonstrated via the analysis of the Hardware Assisted Trace (HAT) framework [29], and the design of a cloud monitoring system for auto-scaling [30]. These use cases demonstrate that our proposed taxonomy can help to ease the effort in designing data aggregation for real-time data-intensive sys-
tems, and prevent software design flaws prior to implementation. We demonstrate the applicability and usefulness of UTRAN, UPPCART, UPPCART-SMC, as well as U²Transformer, via the design of a collision avoidance system for autonomous construction vehicles. The validation demonstrates that the transactions and their desired properties can be easily specified, automatically formalized, and formally analyzed using the state-of-art (statistical) model checking tools, which guides the designers towards appropriate customizations.

## 1.1 Thesis Overview

This thesis is divided into two parts. The first part is a summary of our research, including the preliminaries of this thesis (Chapter 2), the problem formulation and our research goals (Chapter 3), the research methods applied in this thesis (Chapter 4), a brief overview of our contributions (Chapter 5), a discussion on the related work (Chapter 6), as well as our conclusions, limitations and future work (Chapter 7).

The second part is a collection of papers included in this thesis, listed as follows:


**Abstract:** Data aggregation processes are essential constituents for data management in modern computer systems, such as decision support systems and Internet of Things (IoT) systems. Understanding the common and variable features of data aggregation processes, especially their implications, is key to improving the quality of the designed system and reduce design effort. In this paper, we present a survey of data aggregation processes in a variety of application domains from literature. We investigate their common and variable features, which serves as the basis of our previously proposed taxonomy called DAGGTAX. By studying the implications of the DAGGTAX features, we formulate a set of constraints to be satisfied during design, which helps to reduce the design space. We also provide a set of design heuristics that could help designers to decide the appropriate mechanisms for achieving the selected features. We apply DAGGTAX on industrial case studies, showing that DAGGTAX not only strengthens the understanding, but also serves as the
foundation of a design tool which facilitates the model-driven design of data aggregation processes.

**My contribution:** I was the main driver of the paper. I performed the survey on data aggregation processes, proposed the taxonomy and formulated the constraints and heuristics. I also conducted the industrial case study and wrote the major part of the paper. The other authors have contributed with important ideas and comments.


**Abstract:** Efficient monitoring of a cloud system involves multiple aggregation processes and large amounts of data with various and interdependent requirements. A thorough understanding and analysis of the characteristics of data aggregation processes can help to improve the software quality and reduce development cost. In this paper, we propose a systematic approach for designing data aggregation processes in cloud monitoring systems. Our approach applies a feature-oriented taxonomy called DAGGTAX (Data AGGRe-gation TAXonomy) to systematically specify the features of the designed system, and SAT-based analysis to check the consistency of the specifications. Following our approach, designers first specify the data aggregation processes by selecting and composing the features from DAGGTAX. These specified features, as well as design constraints, are then formalized as propositional formulas, whose consistency is checked by the Z3 SAT solver. To support our approach, we propose a design tool called SAFARE (SAAt-based Feature-oriented dAta aggREgation design), which implements DAGGTAX-based specification of data aggregation processes and design constraints, and integrates the state-of-the-art solver Z3 for automated analysis. We also propose a set of general design constraints, which are integrated by default in SAFARE. The effectiveness of our approach is demonstrated via a case study provided by industry, which aims to design a cloud monitoring system for video streaming. The case study shows that DAGGTAX and SAFARE can help designers to identify reusable features, eliminate infeasible design decisions, and derive crucial system parameters.

**My contribution:** I was the main driver of the paper. I proposed the formal
analysis of the taxonomy-based specification, and developed the tool. I also conducted the industrial case study and wrote the major part of the paper. The other authors have contributed with important ideas and comments.


**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.

**My contribution:** I was the main driver of the paper. I proposed the modeling and verification approach presented in the paper and wrote the major part of the text. The other authors have contributed with important ideas and comments.


**Abstract:** Concurrency Control (CC) ensures absence of undesired interference in transaction-based systems, e.g. by guaranteeing isolation, thus contributing to their dependability. However, CC may introduce unpredictable
delays and breach timeliness, which is unwanted for real-time transactions. To avoid deadline misses, some CC algorithms relax isolation in favor of timeliness, whereas other alternatives limit possible interleavings by leveraging real-time constraints and preserve isolation. Selecting an appropriate CC algorithm that can guarantee timeliness at an acceptable level of isolation thus becomes an essential concern for system designers. However, trading-off isolation for timeliness is not easy with existing analysis techniques in database and real-time communities. In this paper, we propose to use model checking of a timed automata model of the transaction system, in order to check the traded-off timeliness and isolation. Our solution provides modularized modeling for the basic transactional constituents, which reduces the efforts for selecting the appropriate CC algorithm.

My contribution: I was the main driver of the paper. I proposed the modeling and verification approach presented in the paper and wrote the major part of the text. The other authors have contributed with important ideas and comments.


Abstract: Unambiguous specification and rigorous analysis of transaction atomicity, isolation and temporal correctness is important for the dependability of many database-centric systems. While both high-level specifications and verifiable formal representations for transactions have been proposed, transformation between these two remains undefined and subject to manual work, which can require considerable effort and is prone to error. In this paper, we bridge the gap between our previously proposed high-level specification language UTRAN for transactions, and our previous timed-automata-based UPPCART framework, which allows transaction properties to be verified by the UPPAAL model checker. We provide a translational semantics between these two languages, and facilitate automated transformation from UTRAN specifications to the UPPCART models with tool automation. We also extend the expressiveness of UTRAN and UPPCART to incorporate transaction sequences and their timing properties. The applicability of our tool chain is demonstrated via an industrial case study.
My contribution: I was the main driver of the paper. I proposed the UML profile and the extended modeling and verification framework presented in the paper, and performed the case study. I also wrote the major part of the text. The other authors contributed with ideas and comments.


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 the statistical model checker UPPAAL SMC. To demonstrate the applicability and usefulness, 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 correctness of the designed system is verified using our proposed framework.

My contribution: I was the main driver of the paper. I proposed the framework in the paper and conducted the case study. I also wrote the major part of the text. The other authors contributed with ideas and comments.
Chapter 2

Preliminaries

2.1 Data Aggregation

Data aggregation is the process of producing a synthesized form of data from multiple data items using an aggregate function [21]. It is applied extensively in information systems [21, 36, 37, 24]. For instance, in database systems, data tuples are aggregated to compute statistical values, such as the average salary of a department; in resource-constrained systems, large amounts of data are aggregated as minimum and maximum values over time in order to save storage or transmission resources; in systems concerning privacy and security, aggregating user details into general profiles prevents information exposure.

In complex information systems, the aggregated data of one aggregation process could serve as the raw data of another process, forming a multi-level aggregation architecture. For instance, a cooperative autonomous robot aggregates the states of its companions to make a decision, which could again be transmitted to other robots as the raw data for their aggregation [38]. VigilNet [39], an integrated multi-layer sensor network system, exploits four levels of aggregation to perform real-time surveillance. As shown in Figure 2.1, the first-level aggregation takes place in the sensor layer, in which surveillance sensor data are aggregated to form detection confidence vectors. These confidence vectors are used as raw data for the second-level aggregation in the node layer, to produce a report of the tracked targets. Using the reports from the nodes, the third-level aggregation creates aggregated reports for each group of nodes. At last, in the base layer, reports from the groups of nodes are aggregated together with historical data to make the estimation of the targets. Both the data
and the aggregation processes in each level could have their unique characteristics. In VigilNet, the values of the sensor data become obsolete much faster than the historical data. Each type of sensor data also differs in when and how the data are collected and used. The aggregation processes also have different characteristics. In these four levels, the processes are triggered by different conditions, and apply various functions to perform aggregation. Although all these processes have to meet real-time requirements, the aggregation processes in the first and second levels have more strict time constraints than the other aggregation processes.

In this thesis, we have surveyed how Data Aggregation Processes (DAP) are designed in modern information systems, and studied their common and variable characteristics. Based on these studies we propose our taxonomy of data aggregation processes in Chapter 8, and tool-supported analysis of DAP specifications in Chapter 9.

### 2.2 Transactions and Customized Transaction Management

A transaction is a partially-ordered set of logically-related operations on the database, which as a whole guarantees logical data consistency [5], that is, satisfying a set of integrity constraints imposed on the database [40]. The partially-ordered set of operations is called a work unit [14], which may include read operations that read data from the database, write operations that modify the data in the database, and be extended with other operations that do not interact with the database directly. Initially, a transaction maintains logical data consistency by ensuring the so-called ACID (Atomicity, Consistency, Isolation, and Durability) properties during the execution [5, 6]. Atomicity
refers to the “all-or-nothing” semantics, meaning that if a transaction fails before completion, all its changes should be undone. Consistency requires that a transaction executed alone should not violate any logical constraints. Isolation refers to the property that no uncommitted changes within a transaction should be seen by any other, in a concurrent execution. If a transaction is committed, durability requires that its changes should be permanent and able to survive system failures.

Example 1. Let us consider two bank accounts, A and B, each having an initial balance of 150. Program 2.1 shows a transaction $T$ that transfers 100 from account A to B. In this transaction, “Begin” indicates the start of the transaction, while “Commit” and “Abort” indicate the successful termination and failure, respectively. The transaction first reads the value of A from the database to a local variable, subtracts 100 from A, and writes the new value back to the database. Similarly, it then adds 100 to B in the database. When $T$ is executed alone and commits, the values of A and B are 50 and 250, respectively. Let us assume that full ACID is ensured by the DBMS. In case $T$ is executed alone and commits, the values of A and B are exactly the same as the values when $T$ is started, that is, 150 for both (atomicity assurance). An integrity constraint is implemented to make sure that the balance of A is never negative, if $T$ is executed alone (consistency assurance). If two instances of $T$ are executed concurrently, their effects on the database are as if they are executed one after another, that is, one transaction changes A to 50 and B to 250, while the other one gets aborted (isolation assurance). Once $T$ has committed, the new values are permanently stored in the disk, and can be recovered if the DBMS crashes (durability assurance).

A database management system enforces transaction properties by applying various mechanisms in transaction management [12]. For instance, recover-
ery techniques, such as *rollback* that undoes all partial results of an aborted transaction, can be applied to restore the consistency of database and achieve atomicity. Concurrency control techniques are applied by DBMS to prevent unwanted interleavings of concurrent transactions from occurring, in order to guarantee isolation. One concurrency control example to provide full isolation is to lock the shared data such that the conflicting transactions are executed serially.

### 2.2.1 Relaxed ACID Properties

Although full ACID assurance achieves a high level of logical data consistency and has thus witnessed success in many applications, it is not a “one-size-fits-all” solution for all applications [41, 14]. First, full ACID assurance might not be necessary, or desired, depending on the application semantics. For instance, in Computer Supported Cooperation Work (CSCW) systems, a transaction may need to access partial results of another concurrent transaction, which is however prohibited by full isolation [16]. Second, full ACID assurance may not be possible under the particular system constraints. As stated in the CAP theorem [42], in distributed database systems with network Partitions (P), trade-offs always occur between logical data Consistency (C) and Availability (A). Further, the PACELC theorem [43] states that even when partitions do not exist, the database system always needs to trade off between logical data consistency and latency. Therefore, in scenarios such as cloud computing and high-volume data stream management, full ACID is relaxed for availability and low latency.

**Example 2.** Let us consider a travel agency that provides reservation services. A typical trip reservation transaction could involve the following series of activities: booking a flight, booking a hotel, and paying the bill. If full ACID is ensured, when a customer books a trip, the updated information of available flight tickets is not visible to another customer until the payment has succeeded due to full isolation. This results in long waiting time for the other customer to get updated information. In addition, due to full atomicity, failing to complete the hotel reservation will lead to the rollback of the entire transaction, including the flight reservation. The customer will need to book the flight again, but the tickets may have already been sold out by that time. In order to provide better service, the traveling agency may desire another transaction model, with relaxed ACID properties.

The relaxation could be carried out in one or several of the ACID properties, depending on the requirements of the developed system. Decades of re-
search have proposed a rich spectrum of transaction models, each consisting of a particular level of A, C, I, and D [14]. For instance, in the nested transaction model [44], if a sub-transaction fails, its parent can decide whether to ignore the failure, or to restart the failed sub-transaction, rather than to abort the entire transaction as required by full atomicity. By applying the nested transaction model in the travel agency example, the trip reservation transaction can choose to continue when a failure occurs in the hotel booking sub-transaction. Another example of transaction models with relaxed ACID is the SAGAS model [45], in which a long-running transaction can be divided into steps. As a relaxation of full isolation, the results of these internal steps are visible to other transactions before the long-running transaction is committed. By using this model in the travel agency system, the updated tickets information is allowed to be seen by other customers before the payment is finalized.

In the following subsections, we recall the atomicity and isolation variants, as well as their supporting mechanisms, within the focus of this thesis. For more information about transaction models and relaxation variants of ACID, we refer the readers to literature [14].

### 2.2.2 Atomicity Variants and Abort Recovery Mechanisms

We consider two atomicity variants in this thesis, *failure atomicity* and *relaxed atomicity*, differentiated by how the database consistency should be restored when a transaction is terminated with partial completion due to errors. Failure atomicity requires that the database is recovered to the exact state before the aborted transaction was started. This also complies to the “nothing” semantics of full atomicity. Relaxed atomicity, on the contrary, only requires that the database is restored into such a state that is considered as consistent according to the application semantics, which is not necessarily the exact one before the transaction. It is a weaker variant of atomicity compared to failure atomicity.

DBMS applies abort recovery mechanisms to realize its supported atomicity. The most common mechanism for failure atomicity is *rollback*, which restores database consistency by undoing all changes made by the to-be-aborted transaction [5]. For instance, the DBMS rolls back transaction $T$ in Program 2.1 by rewriting the values of accounts A and B with their old values, which may be cached by the DBMS in a transaction log.

We also consider two recovery mechanisms that support relaxed atomicity in this thesis, as follows.

- **Immediate compensation** executes a sequence of operations immediately upon abortion, in order to update the database into a consistent state.
Table 2.1: Atomicity variants and the supporting recovery mechanisms

<table>
<thead>
<tr>
<th>Atomicity Variants</th>
<th>Abort Recovery Mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure Atomicity</td>
<td>Rollback</td>
</tr>
<tr>
<td>Relaxed Atomicity</td>
<td>Immediate Compensation</td>
</tr>
<tr>
<td></td>
<td>Deferred Compensation</td>
</tr>
</tbody>
</table>

For instance, when a payment step with debit account fails in a booking transaction, a compensation can be initiated immediately by the DBMS, which tries to continue the booking using the client’s credits.

- **Deferred compensation** also executes an extra sequence of operations to restore consistency. In contrast to immediate compensation, such operations are executed as an ordinary transaction, which is scheduled with other transactions in the DBMS.

Using the compensation mechanisms, the recovery operations are designed flexibly, depending on the application semantics.

To summarize, the recovery mechanisms and their supported atomicity variants are listed in Table 2.1.

### 2.2.3 Isolation Variants and Pessimistic Concurrency Control Algorithms

In this thesis, we are particularly interested in a set of isolation variants, called *Isolation levels*, which represent a well-accepted framework for relaxing isolation, and are implemented by most commercial DBMS. The isolation levels are introduced in the ANSI/ISO SQL-92 standard [46], and later extended and generalized by Berenson et al. [47] and Adya et al. [48]. An isolation level is defined as the property of avoiding a particular subset of phenomena (or anomalies), that is, a particular series of interleaved transaction executions that can lead to inconsistent data [48, 46]. Assuming that $T_1$ and $T_2$ are two transactions as defined previously, we describe the phenomena introduced by the SQL-92 standard as follows:

- **Dirty Read**. Transaction $T_2$ reads a data item that was modified by transaction $T_1$ before $T_1$ commits. If $T_1$ is rolled back, the data read by $T_2$ is not valid.
2.2 Transactions and Customized Transaction Management

Table 2.2: Isolation levels in the ANSI/ISO SQL-92 standard [46]

<table>
<thead>
<tr>
<th>Isolation level</th>
<th>Dirty read</th>
<th>Non-repeatable Read</th>
<th>Phantom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Uncommitted</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Read Committed</td>
<td>Not Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Repeatable Reads</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Serializability</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Not Possible</td>
</tr>
</tbody>
</table>

- **Non-repeatable Read.** Transaction T1 reads a data item. Before T1 commits, T2 modifies this data item and commits. If T1 reads the same data again, it will receive a different value, and thus the data used by T1 becomes inconsistent.

- **Phantom.** Transaction T1 reads a set of data items that satisfy a search condition. Before T1 commits, T2 modifies a data item that affects the result of the search condition and commits. If T1 reads data with the same condition again, it will receive a different set of items, and thus the data used by T1 become inconsistent.

Four isolation levels are defined in the SQL-92 standards, which are *Read Uncommitted* (the most relaxed isolation), *Read Committed, Repeatable Reads*, and *Serializability* (the most strict isolation). As listed in Table 2.2, the Serializability level precludes all types of phenomena, the Read Uncommitted level allows all these phenomena, whereas other levels are defined to preclude a selected set of the phenomena.

Among various types of concurrency control applied in DBMS, in this thesis we focus on one of the most common types called *Pessimistic Concurrency Control (PCC)*, which employs locking techniques to prevent interferences [12]. In PCC, a transaction needs to acquire a lock before it accesses the data, and release the lock after using the data. The DBMS decides which transactions should be granted with the lock, wait, or be aborted, when lock conflicts occur [12].

A wide range of PCC algorithms have been proposed in literature [12]. They differ from each other in the types of locks, the locking durations, as well as the conflict resolution policies. As a result, these algorithms rule out various types of phenomena, and achieve different levels of isolation. For instance, as explained by Gray et al. [49] and Berenson et al. [47], one can achieve the different SQL-92 isolation levels by adjusting the lock types and locking durations (Table 2.3). In this table, a lock on a data item refers to the fact that a
Table 2.3: SQL-92 isolation levels achieved by adjusting locks [49, 47]

<table>
<thead>
<tr>
<th>Isolation level</th>
<th>Read locks on data</th>
<th>Write locks on data</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ UNCOMMITTED</td>
<td>no locks</td>
<td>long write locks</td>
</tr>
<tr>
<td>READ COMMITTED</td>
<td>short read locks on data item</td>
<td>long write locks on data item</td>
</tr>
<tr>
<td>REPEATABLE READS</td>
<td>long read locks on data item, short read locks on phantom</td>
<td>long write locks on data item</td>
</tr>
<tr>
<td>SERIALIZABILITY</td>
<td>long read locks on both data item and phantom</td>
<td>long write locks on data item</td>
</tr>
</tbody>
</table>

lock is required before reading/writing the data item, while a lock on phantom refers to the fact that a lock on the set of data items satisfying a search condition is required. A short read/write lock means that the lock is released immediately after the read/write is performed, while a long read/write lock means that the lock is released only when the transaction is committed.

### 2.2.4 Real-time Transactions and Temporal Correctness

In real-time database systems, a *real-time transaction* is one whose correctness depends not only on the logical data consistency, but also on the temporal correctness, which is imposed from both the transaction computation and the data [4]. As any other real-time computation, a real-time transaction should complete its work by its specified deadline. This property is referred to as *timeliness*. In addition, the data involved in the computation should be temporally consistent, including two aspects: *absolute temporal validity* and *relative temporal validity* [7]. A data instance is absolutely valid if its age from being sampled is less than a specified absolute validity interval. A data instance derived from a group of other real-time data (base data) is relatively valid, if these base data are sampled within a specified relative validity interval.

In order to satisfy temporal correctness, time-cognizant scheduling techniques are often adopted by real-time DBMS, by taking priorities and timing constraints of transactions and data into consideration when producing transaction schedules [7]. For instance, transactions may be scheduled according to the earliest deadlines among them, or based on the temporal validity of data.

In RTDBMS with real-time constraints, since the mechanisms for full ACID may introduce unacceptable latency and unpredictability, ACID may need to be relaxed in order to ensure temporal correctness [11]. For instance, failure atomicity by rolling back all changes may not be affordable as deadlines could
be missed, while relaxed atomicity with compensation can be an alternative to satisfy the logical and temporal requirements. Concurrency control algorithms ensuring full isolation have also been considered as a bottleneck to achieve temporal correctness, as they may cause unpredictable delays introduced by long blocking, arbitrary aborting and restarting, which could lead to deadline misses. In fact, recent research reveals that, waiting for the acquired lock alone may contribute to 37.5% of the latency variance in the selected DBMS [50]. Therefore, RTDBMS may choose a concurrency control algorithm that achieves relaxed isolation, and improves timeliness [10, 51, 52].

### 2.2.5 Customized Transaction Management in DBMS

The widely accepted DBMS development methodology [12] describes two parallel processes for the development of database systems, as presented in Figure 2.2. The left process explains the phases for the development of data content and structures. Starting from the data requirements, one should analyze the data concepts and their relationships, which are later consolidated as DBMS-dependent database schema, during the logical design, physical design and system implementation phases. The right process depicts the development of application logic, and to a large extent regards the design of transactions. Between the conceptual design and logical design phases, the developer must decide the DBMS that provides transaction management. In principle, the selected DBMS must ensure the required properties of transactions, such as performance and other constraints, for which customization of the DBMS mechanisms is often unavoidable.

Most commercial DBMS products, for instance, Oracle ¹, Mimer ² and eXtremeDB ³ allow users to customize transaction management via configurations in run-time. Although failure atomicity and rollback are often the default settings, developers can encode compensations flexibly through transaction error handling API provided by the DBMS. Various isolation levels and concurrency control algorithms can also be selected by developers as optimization options.

Apart from adopting an off-the-shelf DBMS with configuration options, system designers can also develop a DBMS with the liberty of selecting transaction management features. Researchers have proposed DBMS platforms with design customization and system generation in various granularity. The

---

¹[https://www.oracle.com/database/](https://www.oracle.com/database/)
²[https://www.mimer.com/](https://www.mimer.com/)
³[https://www.mcobject.com/extremedbfamily/](https://www.mcobject.com/extremedbfamily/)
Figure 2.2: DBMS development methodology [12]
essence of this trend is to abandon the monolithic “universal” database engines, and to advocate for creating customized application-dependent solutions with only a proper set of data management functionalities [53]. For instance, COMET [19] uses component-based software engineering technology to construct a DBMS for real-time systems, in which different data access methods and concurrency control algorithms can be selected for composition. FAME-DBMS [20] considers the data and transaction management mechanisms as selectable features of a DBMS product line, and creates particular DBMS products via composition of the features. AspectOptima [17] decomposes common transaction management mechanisms as aspects, and builds customized DBMS by weaving the selected aspects. Some commercial DBMS products, such as Berkeley DB ⁴, also allow designers to perform design-time configurations in the source code from provided options, in order to create a customized DBMS.

Although these aforementioned products and platforms provide means to customize DBMS, they do not provide any methodology on how to make the customization choice, such that the desired transactional properties can be satisfied. In practice, the customization decisions are decided upon the provided features (indexing, concurrency control, storage, etc.) and the resource requirements (e.g., memory footprint), while temporal correctness is often not the concern. Transaction throughput experiments on standard benchmarks (such as TPC benchmarks ⁵) are also often used as criteria. However, these benchmarks cannot reflect the actual transaction workload of the designed real-time system, whose temporal correctness highly depends on the activation and data access patterns. In this thesis, we focus on proposing a methodology to guide the customization, based on formal analysis of the trade-offs between ACID, and temporal correctness. The proposed formal analysis framework is presented in Chapters 10, 11, 12 and 13.

2.3 Formal Methods Applied in this Thesis

In this section, we provide an overview of the formal techniques and tools applied in the thesis.

⁵http://www.tpc.org/information/benchmarks.asp
2.3.1 Boolean Satisfiability Problem Solving Using Z3

The Boolean Satisfiability Problem (SAT) refers to determining whether there exists an assignment of the variables in a propositional formula, such that the formula evaluates to true [54]. Usually, the considered propositional formula is given in the conjunctive normal form, that is, a conjunction (AND, ∧) of clauses, where each clause is a disjunction (OR, ∨) of literals, and each literal is either a variable or its negation (NOT, ¬). A SAT solver can automatically find an assignment that satisfies the formula, or otherwise return the contradictory clauses.

Z3 [55] is one of the state-of-the-art Satisfiability Modulo Theories (SMT) solvers, developed by Microsoft. SMT generalizes the satisfiability check from boolean logics to other background theories, such as equality reasoning, arithmetic, fixed-size bit-vectors, arrays, etc. [54]. Z3 provides “black-box” satisfiability checks with well-defined programming interfaces in several programming languages. This makes Z3 an ideal choice to be integrated as part of many software verification and design tools. If the given propositional formula is satisfiable, Z3 gives “yes” as an answer; otherwise, in the case the formula is not satisfiable, Z3 can also return an unsatisfiable core, which is a conjunction of unsatisfiable clauses, to help find the inconsistency in the specification.

In our work, we formulate the specifications of data aggregation processes and the constraints as propositional formulas, and check if there is any contradiction by employing SAT solving with Z3 in our SAFARE tool [30].

2.3.2 Model Checking Using UPPAAL

Model checking is a formal analysis technique that rigorously checks the correctness of a given model of the analyzed system, by exploring all possible states of the model, exhaustively and automatically [56]. An overview of the model-checking technique is shown in Figure 2.3. The formal model of the system is described in a language such as UPPAAL Timed Automata [31] presented in Chapter 2.3.2. The properties to be verified are formalized in some logic, in our case as temporal logic (Timed Computation Tree Logic [57]) formulas. The model checker implementing a model-checking algorithm can then automatically verify whether the properties are satisfied by the system model.

The properties verified by a model checker are of two main types: (i) invariance properties, of the form “something (bad) will never happen”, and (ii) liveness properties, of the form “something (good) will eventually happen”. The result of the verification given by the model checker is a “yes/no” answer,
2.3 Formal Methods Applied in this Thesis

Formal model of the system

Formalized Requirement

Model Checker

Yes

No

Counter-Example
(invariance properties)

Figure 2.3: Model-checking technique

indicating that the verified property is satisfied/violated, respectively. For invariance properties, if a “no” answer is given, a model execution trace could be returned that acts as a counterexample to the invariance property, as shown in Fig 2.3.

UPPAAL Timed Automata

In this thesis, we use the Timed Automata (TA) formal framework [58] to model real-time transactions, and the UPPAAL model checker [31] to verify their correctness (the relaxed ACID properties and temporal correctness). Our choice is justified by the fact that timed automata is an expressive formalism intended to describe the behavior of timed systems in a continuous-time domain. Moreover, the framework is supported by the UPPAAL tool, the state-of-the-art model checker for real-time systems. UPPAAL uses an extended version of TA for modeling, called UPPAAL TA in this thesis.

Timed automata [58] are finite-state automata extended with real-valued clock variables. As mentioned, UPPAAL TA [31] extends TA with discrete variables as well as other modeling features, like urgent and committed locations, synchronization channels, etc.

Formally, an UPPAAL timed automaton [31] is defined as the following tuple:

\[ TA := (L, l_0, X, V, I, Act, E), \]  \hspace{1cm} (2.1)

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 to locations, where $B(X)$ denotes
the set of clock constraints, which are conjunctions of atomic constraints
in the form of $x_i \equiv n$ or $x_i - x_j \equiv n$, $x_i, x_j \in X$, $n \in \mathbb{N}$, and
\(\Downarrow \in \{<, \leq, =\};
• $Act = \sigma \cup \{\tau\}$ is a finite set of actions, where $\sigma$ is a set of synchronization actions, and $\tau \notin \sigma$ represents internal actions without synchronization, or empty actions;
• $E \subset L \times B(X,V) \times Act \times R \times L$ is a set of edges. $B(X,V)$ denotes the
set of guards, as a conjunction of the mentioned clock constraints $B(X)$,
with non-clock constraints $B(V)$, which are conjunctions of atomic constraints
in the form of $v_i \sim m$ or $v_i \sim v_j$, $v_i, v_j \in V$, $m \in \mathbb{Z}$, and
\(\sim \in \{<, \leq, >, \geq, =, \neq\}$. $R$ denotes the set of assignments that update
the values of discrete variables, or reset clocks.

We denote an edge $e \in E$ as $l \xrightarrow{g,a,r} l'$, where $g \in B(X,V)$, $a \in Act$, and
$r \in R$. An edge is enabled if its associated guard evaluates to true, and the
source location is active.

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 $l \xrightarrow{g,a,r} l'$, while performing the action $a$ and the assignment $r$.

Multiple TA can form a Network of Timed Automata (NTA) via CCS parallel composition ("||") [59], by which individual TA are allowed to carry out internal actions (i.e., interleaving), while pairs of TA can communicate via
channels, or shared variables. The state of an NTA consists of the values of all
clocks in the NTA, together with the currently active locations of each TA.
For a communication via channel $c$, the sender is denoted by "c!", while the
receiver by "c?". There are two types of channels. A binary channel allows
the sender and its receiver to perform hand-shake synchronization. An edge
with the sending (or receiving) mark is enabled, only if the receiver (or sender)
can perform a complementary receiving (or sending) action. Therefore, both
the sender and the receiver may be blocked, if its counterpart automaton is not
ready. A broadcast channel allows one sender to send messages to multiple receivers. The sender is not blocked, if any of the receivers is not able to perform the receiving action.

We illustrate the basics of UPPAAL TA via a simple example. For more details, we refer the readers to the literature [31]. Figure 2.4 shows a simple network of UPPAAL TA composed of automata A1 and A2. In the figure, a clock variable \( cl \) is defined in A1 to measure the elapse of time, and progresses continuously. A discrete variable \( a \) is defined globally, and shared by A1 and A2. A1 consists of locations L1, L2 and L3, out of which L1 is the initial location. At each location, an automaton may non-deterministically choose to take a delay transition or a discrete transition. For instance, A1 may stay at L2 until the value of \( cl \) reaches 3, or move to L3 when the value of \( cl \) is greater than or equal to 1. While moving from L2 to L3, A1 synchronizes with automaton A2 by sending a message through the channel \( ch \), and increments the value of \( a \) by the function \( inc(a) \). Meanwhile, upon receiving the message from channel \( ch \), A2 moves from location L4 to location L5.

A location can be urgent or committed. When an automaton reaches an urgent location, marked as “U”, it must take the next transition without any delay in time. Another automaton may take transitions at the time, as long as the time does not progress. In our example, L5 is an urgent location. A committed location, marked as “C”, indicates that no delay occurs on this location and the following transitions from this location will be taken immediately. When an automaton is at a committed location, another automaton may NOT take any transitions, unless it is also at a committed location. L3 is a committed location.
Chapter 2. Preliminaries

UPPAAL Model Checker

Properties of the analyzed system, modeled as a network of UPPAAL TA, can be verified by the UPPAAL model checker [31]. UPPAAL supports verification of invariance and liveness properties, specified in a decidable subset of (Timed) Computation Tree Logic (TCTL) [57]. An invariance property is often specified as $A \left[ P \right]$, meaning that predicate $P$ always holds for all possible execution paths, where $A$ is the universal path quantifier, and $\left[ \right]$ is the “always” temporal operator. If an invariance property is violated, UPPAAL can provide a trace as a counterexample to the property. In this thesis, we also focus on the liveness properties specified as $P \rightarrow Q$ ($P$ “leads to” $Q$), meaning that if $P$ holds, $Q$ will eventually hold. This property is equivalent to $A \left[ P \implies A < > Q \right]$, where $< >$ is the “eventually” temporal operator.

2.3.3 Statistical Model Checking Using UPPAAL SMC

As the model grows complex, traditional exhaustive model checking may suffer from the state explosion problem, that is, the state space to be explored becomes so large that the model checking process cannot reach a conclusion within the given memory and computation resources. Statistical Model Checking (SMC) is proposed as an alternative to analyze large models without exhaustive exploration of the entire state space, at the expense of a full guarantee. SMC is a formal technique that estimates the probability of property satisfaction or violation, based on a finite number of monitored simulations of the model [60]. Each simulation generates a result on whether the checked property is satisfied or not during this run. Using the simulation statistics, SMC infers the probability of satisfaction associated with a confidence level. Compared to exhaustive model checking, SMC cannot provide a guarantee of the result, but the probability of errors can be bounded. As an advantage, SMC is far less memory and time intensive, making it suitable for large models, which may lie beyond the analytical capability of exhaustive model checking.

UPPAAL Stochastic Timed Automata

We choose UPPAAL Stochastic Timed Automata (STA) [61, 35] as our formal representation, which can be analyzed by the state-of-art UPPAAL SMC statistical model checker [35], the extension of UPPAAL for analyzing STA models.

UPPAAL Stochastic Timed Automata (STA) [61, 35] extends UPPAAL TA...
with stochastic interpretations. Formally, an STA is defined as

\[ STA ::= (TA, \mu, \gamma), \]

where \( TA \) is a UPPAAL timed automata defined in Equation 2.1. \( \mu \) is a set of probability density functions that assign delays to the delay transitions in the \( TA \), for each state \( s = (l, v) \). Let \( E_l \) denote the disjunction of guards \( g \) such that \( e = l \xrightarrow{g,a,-} - \), \( e \in E \), for some action \( a \). Let \( d(l, v) \) denote the infimum delay before the action is enabled, i.e., \( d(l, v) \equiv \inf \{ d \in \mathbb{R}_{\geq 0} : v + \Delta d \models E \} \), and \( D(l, v) \) denote the supremum delay at location \( l \), i.e., \( D(l, v) = \sup \{ d \in \mathbb{R}_{\geq 0} : v + d \models I(l) \} \). If \( D(l, v) < \infty \), then the delay density function \( \mu_s \) for state \( s \) is a uniform distribution on \([d(l, v), D(l, v)]\). Otherwise, when the upper bound on the possible delays does not exist, \( \mu_s \) is an exponential distribution with rate \( P(l) \), where \( P : L \to \mathbb{R}_{\geq 0} \) is an additional distribution rate specified for the automaton.

\( \gamma \) is a set of probability density functions that assign the probability distribution for multiple discrete transitions from a state. For each state \( s = (l, v) \), \( \gamma_s \) is a uniform distribution over the set \( \{ a : l \xrightarrow{g,a,-} - \in E \land v \models g \} \).

Multiple STA can form a Network of STA (NSTA). Component STA performs internal actions independently, based on its probability density functions, and communicate via broadcast channels.

Let us consider the model in Figure 2.4, which is now a network of STA. Similarly to the NTA example, STA A1 repeatedly increments the value of \( a \), while STA A2 repeatedly moves between locations L4 and L5, controlled by the message sent by A1 via channel \( ch \). The only syntactical change is that, the previous binary channel \( ch \) in NTA is now a broadcast channel. Semantically, the transitions are taken randomly, following the probabilistic functions defined in the NSTA. For instance, the delay of A1 at location L1 follows an exponential distribution with rate 1, while the delay at L2 follows a uniform distribution on \([1, 3]\).

**UPPAAL SMC**

UPPAAL SMC provides probability evaluation, hypothesis testing and probability comparison for STA models. In this thesis, we utilize the first two features for our analysis. The specification language of UPPAAL SMC is an extension of the Weighted Metric Temporal Logic [62]. Given \( \star \) to denote either the eventually (<> ) or the globally ([ ]) temporal operator, **probability evaluation** calculates \( Pr(*_{x \leq n} \phi) \), that is, the probability that property \( \phi \) is eventually
(or globally) satisfied within $n$ time units. This is specified as $Pr[\leq n](\cdot \phi)$ in the UPPAAL SMC query language. **Hypothesis testing** compares $Pr(\star \leq n \phi)$ with a given value $p$, specified as $Pr[\leq n](\cdot \phi) \sim p$, where $\sim \in \{<, \leq, >, \geq\}$. For more details about UPPAAL SMC, we refer to literature [35].
Chapter 3

Research Problem

In this chapter, we formulate our research problem and research goals addressed in this thesis.

3.1 Problem Description

Designing an RTDBMS for a real-time system with complex data-related computations may require customization of the transaction management mechanisms, in order to achieve the appropriate trade-offs between ACID and temporal correctness. Due to a lack of support for systematic analysis, the customization decisions are often made by designers in an ad-hoc manner, without thorough consideration on the impact of the actual transaction workload, and their interplay with the transaction management mechanisms. This could lead to an inappropriate design that fails to satisfy the desired properties.

Several issues exist in the design of customized RTDBMS. First, a methodology is needed, which guides the designer to systematically decide an appropriate ACID and temporal correctness trade-off, together with the concrete transaction management mechanisms, from a rich spectrum of possible choices. Such methodology is missing from the common DBMS development research and practice. Second, the characteristics of data aggregation processes, as well as their implications with respect to logical data consistency and temporal correctness, are not well-understood. This information is crucial for the analysis of customization at an early stage, and provides required properties and constraints for the further design. Third, systematic specification of transactions,
the ACID and temporal correctness variants, and the transaction management mechanisms in a high-level language can benefit system designer with a more intuitive way to reason about the trade-offs. However, such specification capability is missing in existing design languages. Last but not least, existing techniques cannot provide assurance that the selected mechanisms for the RT-DBMS can achieve the decided trade-offs.

3.2 Research Goals

Given the above issues, we formulate our overall research goal as follows:

**Overall goal.** Facilitate the systematic design and analysis of customized trans-action-based data management with data aggregation support for real-time systems, such that the desired ACID properties and temporal correctness are assured.

In order to address the overall research goal, we define concrete subgoals that need to be tackled in order to fulfill the former. Designing appropriate real-time data management with data aggregation support requires a profound understanding of data aggregation, as well as means to organize the characteristics and reason about their implications with respect to logical data consistency and temporal correctness. Therefore, we formulate the first subgoal as follows:

**Subgoal 1.** Classify and analyze the common and variable characteristics of data aggregation, such that the consequences of design choices can be reasoned about.

When designing the customized RTDBMS, the challenge is how to derive a transaction model with the appropriate transactional properties, based on application semantics, and decide the appropriate mechanisms from a set of candidates. In this thesis, we focus on analyzing trade-offs between atomicity, isolation and temporal correctness, and the selection of abort recovery mechanisms, concurrency control algorithms and scheduling policies. We select these properties because their mechanisms may heavily impact the assurance of each other, while the specification and analysis techniques for all these properties and mechanisms are lacking [26]. Therefore, our next two subgoals address the specification, and the analysis support, for these properties and mechanisms, respectively. Our second research subgoal is formulated as follows:
Subgoal 2. Specify atomicity, isolation and temporal correctness of transactions, as well as the abort recovery, concurrency control and scheduling mechanisms, in a systematic manner.

As mentioned, our next subgoal addresses the analysis of the properties and mechanisms selected from a set of candidates. The challenge lies in how to formally model these complex mechanisms woven with the transactional work units, such that one can analyze our desired properties rigorously with the state-of-art tools. Due to the large number of variations in the transaction management mechanisms, we also need to provide flexibility in the modeling and the analysis, to facilitate easier iterations of the candidate mechanisms and reduce modeling effort. Based on this, we formulate our third research subgoal as follows:

Subgoal 3. Enable flexible formal modeling of real-time transactions with abort recovery, concurrency control, and scheduling, as well as formal analysis of atomicity, isolation and temporal correctness.

To ensure the applicability and usefulness of our proposed methods on real-world applications, we need to apply them on industrial use cases that expose the mentioned problems. Therefore, our fourth subgoal is presented as follows:

Subgoal 4. Validate the applicability and usefulness of the proposed methods via industrial use cases.
Chapter 4

Research Methods

In this chapter, we introduce the methods used to conduct our research in order to address the research goals. We first describe the general process that we follow in our research, after which we explain the concrete methods used in this thesis.

Our research process is shown in Figure 4.1. This research is initiated by the industrial problems that have not been solved by industrial solutions nor thoroughly studied by academic researchers. In today’s industrial practices, software engineers lack means to develop customized provable data management solutions for emerging data-intensive time-critical applications. This gap could lead to reduced software quality in logical and temporal correctness. Based on the industrial problems, the state of practice and the state of the art, we formulate our research goals, as introduced in the previous chapter. To address these goals, we propose a systematic methodology as our approach, and develop a series of techniques to facilitate this methodology, such that it can be applied to industrial applications. Finally, we validate the approach by applying it to the development of several industrial applications. Our proposed approach, as well as the validation process and results, are documented in a series of research papers and reports.

We apply a set of research methods during the activities of the aforementioned process. For the purpose of identifying the gaps between the problems and existing approaches and formulating the research goals, we apply the “critical analysis of literature” method [63] to study the state of the art and the state of practice of the researched area. In Paper A [29], we survey the literature in data aggregation applications, and propose our taxonomy based on the anal-
ysis on the common and variable characteristics of the surveyed works. We also collect literature in customized real-time data management, as well as in transaction modeling and analysis. By analyzing the strengths and weaknesses of existing approaches, we identify the challenges to be addressed related to our research goals, and propose our solutions.

During the implementation of our approach, we apply the “proof of concepts” method [63] to show the correctness and applicability of our proposed approach. In Papers C [32] and D [33], we show that our UPPCART framework is capable of analyzing transaction isolation and temporal correctness via small yet representative examples. In Papers E [26] and F [34], we apply our UPPCART and UPPCART-SMC frameworks in industrial use cases, respectively, with more concrete and realistic settings. We also develop tool prototypes that implement our solutions as proofs of concepts. In Paper B [30], we develop the SAFARE tool as a proof of concept for our analytical method, which checks the consistency of DAGGTAX specifications using SAT solving techniques. In Paper E [26], we implement the U²-Transformer tool as a proof of concept, to demonstrate our automated conversion from UTRAN specifications to UPPCART models.

When validating the research with industry in the loop, we apply the “proof by demonstration” method [63], by developing demonstrators in industrial use cases. The developed demonstrators, as well as the development process, are eventually evaluated with respect to our research goals by both the researchers and the industrial partners. In Paper A [29], we demonstrate the benefit of DAGGTAX via an joint analysis of the HAT system together with its developers. In Paper B [30], we develop a cloud monitoring system with the engineers in the loop, and demonstrate the strength of DAGGTAX and SAFARE.
Chapter 5

Thesis Contributions

In this chapter, we present the contributions of this thesis that address the aforementioned research goals. We first introduce a development process as our methodology to address the overall research goal, followed by a series of methods and tools to facilitate the process, which address the subgoals, respectively.

5.1 The DAGGERS Process: Data AGGregation for Embedded Real-time Systems

As a first step towards reaching our research goals defined previously, we propose, at a conceptual level, a development process called DAGGERS (Data AGGregation for Embedded Real-time Systems), as the methodology to design customized real-time data management solutions in a systematic manner. This process allows designers to identify work units of data aggregation and other data-related computations, as well as the desired properties from system requirements, based on which to derive the appropriate transaction models and the transaction management mechanisms via model checking techniques.

The DAGGERS process, presented in Figure 5.1, includes three major steps, as follows.

Step 1: Specification of initial work units and requirements. The process starts with analyzing the data-related computations, including data aggregations, in the system requirements. The analysis should identify the work units as well as the logical and temporal constraints that need to be fulfilled. Based
on these work units and constraints, the system designer can propose the initial transaction models, including the specification of the relationships between the transactions, as well as the ACID and temporal correctness variants to be ensured.

**Step II: Iterative refinement of transaction model.** In this step, we apply formal modeling and model-checking techniques to derive the refined transaction models, and select the appropriate run-time mechanisms that ensure the desired ACID and temporal correctness properties. We construct a formal model of the work units, on which the transactional properties are specified formally, and can be checked by the state-of-the-art model checkers. We also assume that a repository of models of commonly used run-time mechanisms, in the same formal representation, has been prepared, which can be reused and composed with the work unit models.

The refinement is the iterative “select-check” process as follows. First, we select one candidate mechanism from the repository and compose its model with the work unit models. Second, we check the composed model against the specified properties. 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 en-
sure the specified properties, the designer needs to adjust the initial transaction models, that is, by adjusting the ACID and temporal correctness variants. If the conflicts cannot be resolved by any transaction model, the designer needs to adjust the requirements as they are proven infeasible under the assumed DBMS platform. As soon as the requirements are adjusted, the entire DAGGERS process is restarted.

The outcome of DAGGERS is the refined transaction models and a set of transaction management mechanisms that are proved to achieve the appropriate ACID and temporal correctness variants. A customized RTDBMS can then be implemented using the transaction models and mechanisms.

**Step III: System generation.** The third step is to generate the RTDBMS, by implementing the verified transaction models and mechanisms, or via customization configurations provided by the DBMS platform.

In this thesis, we focus on providing design support for customized RTDBMS, via a series of methods and tools to facilitate the specification, modeling and analysis of transactions and transactional properties during Step I and Step II of DAGGERS. These proposed methods and tools are formulated as contributions that address the related subgoals as follows.

## 5.2 DAGGTAX: Data AGGregation TAXonomy

In order to identify the data aggregation computations and their required properties, we have extensively surveyed data aggregation processes as proposed in theory and used in practice, and investigated their common and variable characteristics. Based on a survey on data aggregation applications and industrial case studies, we investigate the high-level characteristics relevant to logical data consistency and temporal correctness in Data Aggregation Processes (DAP), and propose a taxonomy, called **DAGGTAX (Data AGGregation TAXonomy)**, in Paper A [29]. It is presented as a feature diagram [64], in which each characteristic is modeled as a feature that can be selected to form a software design. The features in DAGGTAX, extracted from the survey, cover the common and variable characteristics of the main constituents of a data aggregation process, which are the raw data, the aggregate function and the aggregated data, as well as the triggering patterns and real-time properties of the process.

Figure 5.2 presents the feature diagram of DAGGTAX. In this diagram, features presented with solid dots are mandatory features. For instance, “aggregate function” is mandatory for any data aggregation process. Optional
Figure 5.2: Data AGGregation TAXonomy (DAGGTAX)[29]
features are denoted by circles, such as “real-time (P)”, which means that a data aggregation may have real-time constraints. Several features associated with a spanning curve form a group of alternative features, from which one feature must be selected by a particular aggregation process. As an example, the “triggering pattern” of a data aggregation process must be one of the following: “aperiodic”, “periodic”, or “sporadic”. The cardinality $[n_m..n_n]$ ($n \geq m \geq 0$) annotating a feature denotes how many instances of the feature, including the entire sub-tree, can be contained as children of the feature’s parent. We use a star symbol “*” to denote if the bounds of the cardinality are undecided. For instance, in Figure 5.2, a data aggregation process may have at least one “raw data type”.

DAGGTAX provides a comprehensive view of data aggregation processes for designers. A data aggregation process can be constructed via the selection of desired features and their combination. Based on the taxonomy, we have introduced design constraints that eliminate some of the infeasible combinations of features during the design, such as conflicts between real-time data and non-real-time processes. For instance, “persistently stored” data may lead to unpredictability in its accessing time, thus contradicting with “hard real-time” features. Since timing properties such as validity intervals and deadlines are expressed by the taxonomy, we can also calculate the propagation of real-time properties along the aggregation processes in multiple levels of aggregation.

We have also proposed a set of design heuristics to help the designer decide the necessary mechanisms for achieving the selected features and other system properties. An example of such heuristics is that, if the data has a “shared” feature, the designer may need to consider concurrency control in the system design in order to maintain logical data consistency.

5.3 SAFARE: SAT-based Feature-oriented Data Aggregation Design

In order to enable automated reasoning about the correctness of data aggregation design, we propose to transform the feature diagram, as well as the constraints between features, into propositional formulas, which can be checked automatically by existing SAT solvers.

Given a DAGGTAX specification $s$ consisting of a set of selected features $\{f_1, ..., f_m\}$ and a set of de-selected features $\{f_{m+1}, ..., f_n\}$, and a system consisting of a set of DAP $\{s_1, ..., s_k\}$, we define the following rules to transform the specifications and constraints into propositional formulas.
1. For each feature \( f \) in a specification, create a variable \( f \).

2. The selection of feature \( f \) is formalized in Boolean logic, as \( f \), meaning that the boolean variable \( f \) is true. The de-selection of \( f \) is formalized as \( \neg f \).

3. The specification \( s \) is a conjunction of propositions: \( s := f_1 \land f_2 \land \ldots \land f_m \land \neg f_{m+1} \land \ldots \land \neg f_n \).

4. A constraint \( c \) between features is specified as a formula \( c \), which represents the feature variables connected by logical operators \( \land, \lor, \neg \), and \( \Rightarrow \) (implication). For instance, if feature \( f_1 \) requires feature \( f_2 \), this can be formalized as: \( f_1 \Rightarrow f_2 \). If feature \( f_1 \) excludes feature \( f_2 \), this can be formalized as: \( f_1 \Rightarrow (\neg f_2) \).

5. The specification of the system is inconsistent, if the propositional formula \( \Psi \Rightarrow false \) is satisfied, where \( \Psi := s_1 \land \ldots \land s_k \land c_1 \land \ldots \land c_l \), where \( l \) is the number of constraints.

For instance, according to our proposed rules, the real-time strictness level of the raw data must be higher than or equal to the real-time strictness level of the aggregated data. In a concrete specification of a single DAP called \textit{DecisionProcess}, if the aggregated data \textit{Decision} is required to be hard real-time, then its real-time raw data \textit{Sensor} must also be hard-real-time. This constraint can be formalized as the following formula (using our naming convention in Paper B):

\[
\text{DecisionProcess\_Decision\_RealTime\_Hard} \land \\
\text{DecisionProcess\_Sensor\_RealTime} \\
\Rightarrow \text{DecisionProcess\_Sensor\_RealTime\_Hard}
\]

If the specification formulated as a conjunction of such formulas is proven inconsistent, it indicates that there exists conflict among the selected features. The designer needs to resolve the conflict by selecting another feature combination.

The automated formalization and analysis of DAGGTAX-based specifications are realized by our design tool called \textbf{SAFARE} (SAat-based Feature-oriented dAta aggREgation design) (Figure 5.3), proposed in Paper B[30]. This tool provides a graphical interface for the specification using DAGGTAX, and automatically checks for inconsistency using the integrated Microsoft Z3...
To address the lack of design-time support for transactions and transactional properties, we propose a UML profile in Paper E[26], called UTRAN (UML for TRANsactions), which provides explicit specification capability for atomicity, isolation and temporal correctness.

The overview of UTRAN [26] is presented in Figure 5.4. UTRAN specifies a transaction as an activity, and includes modeling elements to explicitly express the three properties as well as the corresponding transaction management mechanisms. Using UTRAN, one can specify different atomicity vari-

5.4 The UTRAN Profile: UML for TRANsactions

Theorem Prover [55]. Designers specify the DAP in section A, and the dependencies between DAP in section B, in Figure 5.3. Tab C opens up the consistency check window. If the specification violates a design constraint, for instance, trying to obtain a hard real-time aggregated data from soft real-time raw data, SAFARE will detect the violation, and pin-point the contradictory specification for the designer in the consistency check window. Tab D provides high-level recommendation of DBMS from the registered DBMS options, based on the selected DAP features. For instance, if “durable” is a selected feature, then a DBMS with durability functionality will be recommended by the tool.
Figure 5.4: The UTRAN profile[26]
ants and their supporting recovery mechanisms in the activity diagram, with
the corresponding stereotypes provided by UTRAN. The customized isolation
properties can be specified as a set of transaction interleavings, stereotyped as
IsolationPhenomenon, that should be prevented by the selected CC algorithm.
Time-related information such as deadlines and periods, which are reused from
the UML-MARTE (Modeling and Analysis of Real-Time Embedded systems)
profile [65] for expressing timing information in real-time systems, can be an-
notated to transactions and operations. We also propose a set of syntactic con-
straints for UTRAN specifications, formulated in the Object Constraint Lan-
guage (OCL).

5.5 The UPPCART Framework: UPPaal for Concurrent Atomic Real-time Transactions

One challenge of reasoning about the trade-offs between atomicity, isolation
and temporal correctness lies in the analysis of these properties in a uniformed
framework, together with different abort recovery, concurrency control and
scheduling mechanisms. To address this, we propose a formal framework,
called UPPCART (UPPAAL for Concurrent Atomic Real-time Transactions)
[33, 32, 26], that facilitates flexible modeling of real-time transactions with
these mechanisms, and verification of the specified trade-offs between the men-
tioned properties.

5.5.1 Modeling RTDBMS as a Network of Timed Automata

In UPPCART, the RTDBMS is modeled as a network of UPPAAL TA, which
is composed of automata representing the computational work to be executed,
and automata serving as monitors for the desired properties. An illustration of
UPPCART is presented in Figure 5.5. Formally, we define an RTDBMS \( N \) as
follows:

\[
N ::= W_1 || \ldots || W_n || A_{CCManager} || A_{ATManager} || O_1 || \ldots || O_k || D_1 || \ldots || D_m || S_1 || \ldots || S_l,
\]

(5.1)

where \( W_1, \ldots, W_n \) are work unit automata of transactions \( T_1, \ldots, T_n \), respectively. They also model the WU’s interaction with the transaction manager with
respect to concurrency control and abort recovery. \( A_{CCManager} \) is the CC-
Manager 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 upon abort of transactions. $O_1, ..., O_k$ are IsolationObserver automata that observe the phenomena to be precluded by isolation, respectively. $D_1, ..., D_m$ are data automata for the data with temporal validity constraints, respectively. $S_1, ..., S_l$ are automata for transaction sequences, respectively.

### 5.5.2 Pattern-based Construction

For each type of the aforementioned TA in $N$, we propose a set of parameterized patterns and connectors for the pattern-based construction. A parameterized pattern of TA is a reusable structure that models a repetitive behavior or property. Formally, we define a parameterized pattern as follows:

$$PP(\text{Para}) := (L_{pp}, L_{pinit}, X_{pp}, V_{pp}, I_{pp}, Act_{pp}, E_{pp}) \cup \text{Function}, \quad (5.2)$$

where, aligned with the definition in Equation 2.1, $L_{pp}$ is a finite set of locations, $L_{pinit}$ is a finite set of initial locations, $X_{pp}$ is a finite set of clock variables, $V_{pp}$ is a finite set of discrete variables, $I_{pp}$ assigns invariants to locations, $Act_{pp}$ is a finite set of actions, and $E_{pp}$ is a set of edges. Para is a set of parameters ($para1$, $para2$, ...) that appears in the tuple, and Function is a set of function signatures that appear in $E_{pp}$.

A parameterized connector is a set of edges that connect two parameterized patterns. Formally, a parameterized connector connecting parameterized
patterns $PP_i$ and $PP_j$ is defined as follows:

$$PCon(PP_i, PP_j, \text{Para}) := (L_{pp,i} \times B(X_{pcon}, V_{pcon}) \times Act_{pcon} \times L_{pp,j}) \cup \text{Function.}$$ (5.3)

A parameterized pattern can be constructed from sub patterns and the connectors connecting them, as the unions of their locations, variables, invariances, edges, actions, and parameters.

The instantiation of $PP$ and $PCon$ assigns actual values to the parameters in $\text{Para}$, and provides the functions in $\text{Function}$ with implementations.

Given a TA $A = (L, l_0, X, V, I, Act, E)$, a set of instantiated patterns $\text{P}$, and a set of instantiated connectors $\text{CON}$, $A$ is a pattern-based construction from $\text{P}$ and $\text{CON}$, iff:

- $L = \bigcup_{P_i \in \text{P}} L_{p,i}$,
- $\bigcup_{P_i \in \text{P}} L_{\text{init},i} = \{l_0\}$,
- $X = \bigcup_{P_i \in \text{P}} X_{p,i} \bigcup_{\text{Con}_j \in \text{CON}} X_{\text{con},j}$,
- $V = \bigcup_{P_i \in \text{P}} V_{p,i} \bigcup_{\text{Con}_j \in \text{CON}} V_{\text{con},j}$,
- $Act = \bigcup_{P_i \in \text{P}} Act_{p,i} \bigcup_{\text{Con}_j \in \text{CON}} Act_{\text{con},j}$,
- $E = \bigcup_{P_i \in \text{P}} E_{p,i} \cup \text{CON}$.

We denote it as $A = \bigcup (\text{P}, \text{CON})$, in which $\bigcup$ stands for the pattern-based construction operator.

For the convenience of later presentations, we call a pattern as a \textit{skeleton} of TA $A$, if $L_{\text{init}} \neq \emptyset$.

We list the proposed patterns in Table 5.1, and describe the details of the major patterns in the following texts. For detailed descriptions of all patterns, we refer to Paper E [26].

A \textbf{Work Unit Skeleton} models the basic structure of a work unit of a transaction and its interaction with the CC and atomicity managers. As shown in Figure 5.6, the automaton starts from the \textit{initial} location, initializes the transaction with the specified id $ti$ and priority $p$ using function $\text{initialize}(ti, p)$, and moves to the location $\text{ready}$. Upon receiving the $\text{start\_trans}[ti]$ message, it moves to the location $\text{trans\_started}$, which represents the begin of the transaction, and resets clock variable $tc$. The location $\text{trans\_committed}$ indicates the committed state of the transaction. If the value of $tc$ is greater than the specified
Table 5.1: Patterns in UPPCART

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Unit Skeleton</td>
<td>Basic structure of a work unit automaton</td>
</tr>
<tr>
<td>Operation Pattern</td>
<td>Basic structure of a begin, commit, read, or write operation within a work unit</td>
</tr>
<tr>
<td>UserAbort Pattern</td>
<td>Basic structure of a user-abort operation within a work unit</td>
</tr>
<tr>
<td>Delay Pattern</td>
<td>Basic structure of a delay between operations or transactions</td>
</tr>
<tr>
<td>Locking Pattern</td>
<td>Basic structure of a locking activity</td>
</tr>
<tr>
<td>Unlocking Pattern</td>
<td>Basic structure of an unlocking activity</td>
</tr>
<tr>
<td>RollbackImComp Pattern</td>
<td>Basic structure of a rollback or immediate compensation mechanism</td>
</tr>
<tr>
<td>DeferredComp Pattern</td>
<td>Basic structure of a deferred compensation mechanism</td>
</tr>
<tr>
<td>Transaction Sequence Skeleton</td>
<td>Basic structure of a transaction sequence</td>
</tr>
<tr>
<td>Sequence Sub-transaction Skeleton</td>
<td>Basic structure of a sub-transaction within a transaction sequence</td>
</tr>
<tr>
<td>CCManager Skeleton</td>
<td>Basic structure of a CCManager automaton</td>
</tr>
<tr>
<td>ATManager Skeleton</td>
<td>Basic structure of an ATManager automaton</td>
</tr>
<tr>
<td>IsolationObserver Skeleton</td>
<td>Basic structure of an IsolationObserver automaton</td>
</tr>
<tr>
<td>Data Skeleton</td>
<td>Basic structure of a data automaton</td>
</tr>
</tbody>
</table>
5.5 The UPPCART Framework: UPPaal for Concurrent Atomic
Real-time Transactions 49

When $tc <= PERIOD$, the automaton moves to the location $trans_started$, indicating a deadline miss. Otherwise, it waits until the specified $PERIOD$ has reached, and moves to $begin$ for the next activation. The location $trans_aborted$ represents the aborted state of the transaction. Similarly, if the value of $tr$ is greater than the specified $RECOVERY_DEADLINE$, timeliness is breached, and the work unit automaton moves to $miss_deadline$.

Between $trans_started$ and $trans_committed$ are a set of connected instantiated patterns that model the database and transaction management operations, as well as the delays between the operations. As an example, Figure 5.7 presents the Operation Pattern for modeling database read and write operations, integrated with Locking Pattern and Unlocking Pattern. In this compound pattern, we model the scheduling policy using three functions, namely, $enq_sch(ti)$, $deq_sch(ti)$ and $sch()$. After the start_operation location, the function $enq_sch(ti)$ is called, which pushes the transaction into the scheduling queue. On the edges from the location check_sched, the function $sch()$ checks whether the transaction is the next one to be executed. If yes, the automaton moves to do_operation, representing the execution of the operation; otherwise, the automaton waits at location wait, until the CPU is released by the occupying transaction or the RTDBMS, indicated via the signal in the $cpu_free$ channel.

**Figure 5.6: The Work Unit Skeleton [26]**

**DEADLINE**, the automaton moves to the location $miss_deadline$, indicating a deadline miss. Otherwise, it waits until the specified $PERIOD$ has reached, and moves to $begin$ for the next activation. The location $trans_aborted$ represents the aborted state of the transaction. Similarly, if the value of $tr$ is greater than the specified $RECOVERY_DEADLINE$, timeliness is breached, and the work unit automaton moves to $miss_deadline$.

Between $trans_started$ and $trans_committed$ are a set of connected instantiated patterns that model the database and transaction management operations, as well as the delays between the operations. As an example, Figure 5.7 presents the Operation Pattern for modeling database read and write operations, integrated with Locking Pattern and Unlocking Pattern. In this compound pattern, we model the scheduling policy using three functions, namely, $enq_sch(ti)$, $deq_sch(ti)$ and $sch()$. After the start_operation location, the function $enq_sch(ti)$ is called, which pushes the transaction into the scheduling queue. On the edges from the location check_sched, the function $sch()$ checks whether the transaction is the next one to be executed. If yes, the automaton moves to do_operation, representing the execution of the operation; otherwise, the automaton waits at location wait, until the CPU is released by the occupying transaction or the RTDBMS, indicated via the signal in the $cpu_free$ channel.
The automaton may stay at `do_operation` for at most `WCRT_op` time units, and at least `BCRT_op` time units, which represent the longest and shortest time to complete the operation. Upon the completion of the operation, a signal is sent to the IsolationObservers via channel `notify_op[ti]`. Before reaching `finish_operation`, the CPU is set to be free, and the transaction is removed from the scheduling queue by the function `deq_sch(ti)`. The interactions with the CC manager are modeled by the Locking Pattern that models the behavior of acquiring a lock, and the Unlocking Pattern that models the release of the lock.

The **IsolationObserver Skeleton** is shown in Figure 5.8. Each IsolationObserver observes a specified sequence of operations, by accepting the corresponding notification messages from the work unit automata via channel `notify_op[ti][di]` when an operation is completed. If the monitored sequence occurs, the automaton moves to the `isolation_phenomenon` location, indicating that the phenomenon has taken place.

Figure 5.9 presents the **Data Skeleton**. The clock variable `age` is reset every time a write operation is performed on the data. The value of `age` hence represents how old the data is since the last update.

We also propose patterns for modeling common abort recovery and con-
5.5 The UPPCART Framework: UPPal for Concurrent Atomic Real-time Transactions

Figure 5.8: TA skeleton for the IsolationObserver[26]

Figure 5.9: TA skeleton for Data[26]

currency control algorithms. The ATManager Skeleton models the common behavior of the atomicity manager that performs various recovery activities. The CCManager Skeleton models the common structure for various CC algorithms, as well as the interaction with the transactions and the atomicity manager. The details of these skeletons are presented in the paper [33, 26].

Automated Model Construction. To enable easier construction of UPPCART models, we have developed a tool, called U²Transformer, which implements a translational semantics from UTRAN specifications to UPPCART models. We define the translational semantics of UTRAN by mapping the UTRAN concepts with the UPPCART patterns and connectors. These mapped concepts include «RTDBMSScope», «Transaction», «TransactionSequence», «Operation», «IsolationPhenomenon», «DelayedNext», and their descendants. The specifications «TemporalCorrectnessSpecification», «AtomicitySpecification» and «IsolationSpecification» are annotations that contain information for the selection and instantiation of variant patterns.

As shown in Figure 5.10, U²Transformer accepts a system model defined in UML with the UTRAN profile, created in common UML editors including
52 Chapter 5. Thesis Contributions

Eclipse Papyrus modeling environment \(^1\) and IBM Rational Software Architect (RSA) environment \(^2\), in their respective XML format. A command-line interface is provided by the tool for the user to specify the editor in use. Then a three-step process is performed by U^2^Transformer. In the first step, the tool converts the UTRAN specification in XML into intermediate data structures in JAVA. This step is performed by the UTRANParser module. The second step is performed by the UPPCARTGenerator module, which implements the conversion between the internal UTRAN and UPPCART structures, and returns the XML-format UPPCART models for the UPPAAL tool.

Figure 5.10: Transformation process by U^2^Transformer [26]

5.5.3 Verification

With the transactions as well as the mechanisms modeled as UPPAAL TA, we are able to formally verify atomicity, isolation and temporal correctness using UPPAAL Model Checker. Table 5.2 lists the patterns to formalize the properties in UPPAAL queries. Among them, atomicity is formalized as a liveness property, that is, \(A_i\) will eventually reach locations `trans_rollback` or `trans_compensated` if the variable `abort_id` equals \(i\). Isolation and temporal correctness are formalized as invariance properties. Respectively, the isolation property specifies that the `isolation_phenomenon` locations are not reachable, the timeliness property specifies that the `miss_deadline` location of the analyzed \(T_i\) is not reachable, while temporal validity properties specify that the states where the ages of data exceed their thresholds are never reachable.

To ensure full correctness, we also check the reachability of the antecedents of the properties that contain operators “lead-to” or “imply” in Table 5.2, respectively. If we denote the antecedent as \(P\), its reachability is encoded as

---

\(^1\)https://www.eclipse.org/papyrus/
\(^2\)https://www.ibm.com/developerworks/downloads/r/architect
Table 5.2: UPPAAL query patterns for verifying transactional properties[26]

<table>
<thead>
<tr>
<th>Property Type</th>
<th>Property Description</th>
<th>UPPAAL Query Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomicity</td>
<td>$T_i$ aborted due to ERRORTYPE is eventually rolled back</td>
<td>$(\text{ATManager.abort}<em>{id} == i &amp;&amp; \text{ATManager.error}</em>{type} == \text{ERRORTYPE}) \rightarrow \text{Ai.trans}_{rolledback}$</td>
</tr>
<tr>
<td>Isolation</td>
<td>The specified isolation phenomena never occur</td>
<td>$A[] \not (O_1.isolation_{phenomenon}</td>
</tr>
<tr>
<td>Timeliness</td>
<td>$T_i$ never misses its deadline</td>
<td>$A[] \not \text{Ai.miss}_\text{deadline}$</td>
</tr>
<tr>
<td>Absolute Validity</td>
<td>When read by $T_i$, $D_j$ is never older than the absolute validity interval AVI(j)</td>
<td>$A[](\text{Ai.read}_{di_done} \implies D_j.age \leq AVI(j))$</td>
</tr>
<tr>
<td>Relative Validity</td>
<td>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)</td>
<td>$A[]((\text{Ai.read}_{dj_done}</td>
</tr>
</tbody>
</table>

“$E << P$”, and verified in UPPAAL. However, we have omitted these from Table 5.2.

The basic idea of UPPCART that models the RTDBMS as an NTA is first proposed as a general approach to verify temporal correctness for concurrent transactions in Paper C [32], and is extended as a pattern-based framework for both temporal correctness and isolation in Paper D [33]. In Paper E [26], we extend the framework for the atomicity of transactions, and develop the translation and automated construction of TA models from UTRAN specification.

5.6 The UPPCART-SMC Framework

For the purpose of analyzing large scale RTDBMS with complex mechanisms, we propose a framework called UPPCART-SMC in Paper F [34], to model the transactions and the mechanisms as a network of stochastic timed automata, and analyze atomicity, isolation and temporal correctness using statistical model checking. Similar to UPPCART, in UPPCART-SMC, we model the RTDBMS as a parallel composition of STA that represents the transaction work units and the mechanisms, as well as observers to monitor the properties.
Denoted by $N'$, the NSTA of the modeled RTDBMS is defined as follows:

$$
N' ::= W'_1 | \ldots | W'_n | A'_{CCManager} | A'_{ATManager} | O'_1 | \ldots | O'_k | D'_1 | \ldots | D'_m | S'_1 | \ldots | S'_l
$$

(5.4)

Since we are modeling the exact same RTDBMS as in Section 5.5, each STA in Equation 5.4 corresponds to a TA in Equation 5.1. That is to say, $W'_1, \ldots, W'_n$ are the STA of work units of transactions $T_1, \ldots, T_n$, respectively. $A'_{CCManager}$ is the CCManager STA. $A'_{ATManager}$ is the ATManager STA. $O'_1, \ldots, O'_k$ are the STA of IsolationObservers. $D'_1, \ldots, D'_m$ are the STA that monitor the age of data. $S'_1, \ldots, S'_l$ are automata for transaction sequences, respectively.

We also propose patterns to construct the STA in UPPCART-SMC. These patterns are similar to and based on the ones proposed for UPPCART, but with modifications to incorporate the syntax and semantics of STA. A significant change is to achieve handshake synchronization in UPPCART-SMC via broadcast channels and shared variables, since the paired synchronization channels used in UPPCART are not supported by the tool UPPAAL SMC.

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 5.3, respectively. They encode the probability of property satisfaction, over a chosen simulation time, 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 5.2 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:

$$(ATManager.abort_id == i && ATManager.error_type == ERRORTYPE) \Rightarrow \!,_{\leq RECOVERY_{DEADLINE}} Wi.trans\_rolledback.$$

This property can be checked by UPPAAL SMC with some modifications in the models, as suggested by literature [66]. We first introduce a clock variable $ta$ in $W_i$, whose value is reset when $ATManager.abort_id == i$
Table 5.3: UPPAAL SMC queries for analyzing transactional properties [34]

<table>
<thead>
<tr>
<th>Property Type</th>
<th>Property Description</th>
<th>UPPAAL Query Pattern</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-bounded Atomicity</td>
<td>$T_i$ is rolled back or compensated within $RECOVERY_DEADLINE$</td>
<td>$Pr[&lt;= n](&lt; W_b)$</td>
<td>SQ1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$Pr[&lt;= n][[] (W_b \implies W_{i,ta} &lt;= RECOVERY_DEADLINE))$</td>
<td></td>
</tr>
<tr>
<td>Isolation</td>
<td>The probability of that the specified isolation phenomena never occur</td>
<td>$Pr[&lt;= n][[] \not (O_i, isolation_phenomenon</td>
<td></td>
</tr>
<tr>
<td>Timeliness</td>
<td>The probability of $T_i$ never missing its deadline</td>
<td>$Pr[&lt;= n][[] \not W_{i,miss_deadline}$</td>
<td>SQ3</td>
</tr>
<tr>
<td>Absolute Validity</td>
<td>The probability of that, when read by $T_i$, $D_j$ is never older than the</td>
<td>$Pr[&lt;= n][[](W_i, read_di_done \implies D_j,age &lt;= AVI(j))$</td>
<td>SQ4</td>
</tr>
<tr>
<td>Relative Validity</td>
<td>The probability of that, whenever $T_i$ reads $D_j$ or $D_l$, the age differences</td>
<td>$Pr[&lt;= n][[]((W_i, read_dj_done | W_i, read_dl_done) \implies ((D_j,age - D_l,age &lt;= RVI(j,l)) &amp;&amp; (D_l,age - D_j,age &lt;= RVI(j,l))))$</td>
<td>SQ5</td>
</tr>
</tbody>
</table>


&& ATManager.error _type == ERRORTYPE holds. We also introduce 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 $Wi.trans_rollbacked$ holds. The probability of the time-bounded leads-to property is then reduced to checking the probability of: $\mathbb{E} \left[ \left( Wi.b imply Wi.ta <= RECOVERY\_DEADLINE \right) \right]$, which holds over the number of runs checked during analysis, and is specified as the UPPAAL SMC query $SQ1.2$ in Table 5.3. To ensure that this query is not trivially satisfied, we also check that $Wi.b$ is reachable, via the query $SQ1.1$.

### 5.7 Validation on Industrial Use Cases

To validate our proposed techniques, we apply them on a set of industrial use cases, and evaluate their applicability and usefulness. These use cases are presented in Paper A [29], Paper B [30] and Paper F [34], respectively.

#### Analysis of the Hardware Assisted Trace (HAT) System

In Paper A [29], we apply the proposed taxonomy to the analysis of an industrial project, the Hardware Assisted Trace (HAT) framework [67], together with its proposers from Ericsson. HAT, as shown in Figure 5.11, is a framework for debugging functional errors in an embedded system. In this framework, a debugger function runs in the same system as the debugged program, and collects both hardware and software run-time traces continuously. Two data aggregation processes are identified in the current design. At a lower level, a Program Trace Macrocell (PTM) aggregation process aggregates traces from hardware. These aggregated PTM traces, together with software instrumentation traces from the System Trace Macrocell (STM), are then aggregated by a higher level ApplicationTrace aggregation process, to create an informative trace for the debugged application.

With the DAGGTAX diagrams showing the features of the aggregation processes, the engineers could immediately identify a problem in the PTM buffer management. The problem is that the data in the buffer may be overwritten before they are aggregated. It arises due to the lack of a holistic consideration on the PTM aggregation process and the ApplicationTrace aggregation process at design time. Triggered by aperiodic external events, the PTM process could produce a large number of traces within a short period and fill up the PTM buffer. The ApplicationTrace process, on the other hand, is triggered with a
minimum inter-arrival time, and consumes the PTM traces as unsheddable raw data, meaning that each PTM trace should be aggregated by the former. When the inter-arrival time of the PTM triggering events is shorter than the minimum inter-arrival time of the ApplicationTrace process, the PTM traces in the buffer may be overwritten before they could be aggregated by the ApplicationTrace process. This problem has been observed on Ericsson’s implemented system, and awaits a solution. However, if the taxonomy was applied on the system design, this problem could have been identified before it was propagated to implementation. We have provided two solutions at design level to solve the identified problem.

This evaluation shows that the taxonomy enhances the understanding of the system by the designers. By applying analysis based on our taxonomy, design flaws can be identified and fixed prior to implementation. Design solutions can be constructed by composing reusable features, and reasoned about based on the taxonomy. As a result, the design space of the solutions could be reduced.

**Design a Cloud Monitoring System**

In Paper B [30], we evaluate DAGGTXA and SAFARE using an industrial case study provided by Ericsson that aims to design a cloud monitoring system for an enhanced auto-scaling functionality in a video streaming system, by extending the existing open-source OpenStack framework ³. OpenStack is a cloud operating system that supports auto-scaling of computation and memory resources based on aggregation of resource usages in the virtual machine level. For more accurate and efficient auto-scaling, application-level resource usages

³https://www.openstack.org/software/
need to be taken into account.

We apply DAGGTAX to the current features provided by the existing OpenStack framework for auto-scaling, based on which we design the new features of the extended system (Figure 5.12). We implement the designed layers of aggregation, propagated from raw resource measurements such as CPU consumption of individual applications, to the final . We also use SAFARE to formally analyze the consistency of the specification. For instance, when an aggregation process was specified to generate hard real-time data from a soft real-time process during the design, the consistency check of SAFARE detected and pinpointed the violation so that we could fix it prior to implementation.

Our experience shows that our taxonomy promotes a deeper understanding of the systems behavior, and raises awareness about characteristics that need to be considered as well as issues that need to be solved during the design. It helps designers to perform better analysis than otherwise, such as to identify reusable design solutions, make data management decisions, eliminate infeasible feature combinations, and calculate time-related parameters. Through formal verification, SAFARE helps to detect and eliminate contradictions at the early design stage of data aggregation processes.

Figure 5.12: Applying DAGGTAX to the design of a cloud monitoring system[30]
Design a Collision Avoidance System

We apply UTRAN, UPCART and UPCART-SMC to the design and verification of a collision avoidance system for multiple construction vehicles working autonomously in a quarry. The system should prevent vehicles from colliding into each other or local obstacles. Due to productivity reasons, the vehicles need to accomplish their missions within given deadlines. To achieve this, we design a two-layer collision avoidance system, both relying on the data management and transaction control provided by their DBMS.

The center of the global collision avoidance layer is a global DBMS that stores the map of the quarry, which is divided into smaller cells (Figure 5.13). The missions of vehicles are modeled as a series of transactions. Via concurrency control provided by the DBMS, the transactions are not allowed to access the same cell data, such that the vehicles cannot operate in the same cells and collision is avoided. Isolation is the main concern in this scenario. For productivity, vehicles may be prioritized such that some of the missions can be carried out more timely, which is addressed as temporal correctness of the transactions. Lower prioritized missions may be aborted, and compensated...
later, which is formulated as an atomicity issue.

The local collision avoidance layer in each vehicle monitors the surroundings using a camera, a sensor and a lidar, whose readings are stored in the local database. Transactions are used to update the database, and discover the occurrence of obstacles. Timely update and access of the surrounding data, as well as correct abortion and recovery of transactions upon detection of obstacles, are crucial to the safety of the vehicle.

We present the design and verification of the collision avoidance system in two papers. In Paper E [26], we consider a system with fewer vehicles and a smaller map. We apply UTRAN to describe the DBMS in both layers, during which the required behaviors are specified as atomicity, isolation and temporal correctness properties, respectively. The UTRAN specifications are then translated into UPPCART models using $U^2$-Transformer. Atomicity, isolation and temporal correctness of the designed system are model checked by UPPAAL, which proves that the current RTDBMS designs can guarantee the desired properties.

In Paper F [34], we present the analysis of the global collision avoidance layer designed for a larger number of vehicles on a bigger map. Since exhaustive model checking with UPPCART cannot reach a conclusion due to the drastic growth of states, we resort to statistical model checking with UPPCART-SMC for a probabilistic guarantee of the properties. We apply the UPPCART-SMC patterns to model the transactions and mechanisms as stochastic timed automata, which are analyzed by UPPAAL SMC. The analysis results show that the properties are satisfied with probabilities no lower than 0.9999999 and with a confidence level of 0.99, over the specified simulation time and rounds. Higher probability thresholds and confidence levels entail longer analysis time. As the system scales up with larger amounts of transactions and data, or more complex transaction management mechanisms, longer analysis time is also expected, but memory exhaustion due to state explosion is avoided.

5.8 Research Goals Revisited

Table 5.4 summarizes the relationship between the included papers, the contributions of this thesis, and the research goals. Among the contributions, the DAGGERS process proposed in Chapter 5.1 addresses our overall research goal directly, by providing a systematic process as our methodology. The other contributions are proposed in the included papers, and address the subgoals as follows:
Table 5.4: Contribution of included papers with respect to research subgoals

<table>
<thead>
<tr>
<th>Papers (Contributions)</th>
<th>Overall Goal</th>
<th>SG 1</th>
<th>SG 2</th>
<th>SG 3</th>
<th>SG 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thesis (DAGGERS)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper A (DAGGTAX, validation)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper B (SAFARE, validation)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper C (UPPCART)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper D (UPPCART)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper E (UTRAN, UPPCART, validation)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Paper F (UPPCART-SMC, validation)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- Paper A [29] proposes DAGGTAX to organize the common and variable features of data aggregation processes, such that the designers can analyze their design choices at an early development stage. SAFARE proposed in Paper B [30] develops a method and a tool to formalize DAGGTAX specifications and analyze their consistency. These contributions address **Subgoal 1**: Classify and analyze the common and variable characteristics of data aggregation, such that the consequences of design choices can be reasoned about.

- Paper E [26] proposes the UTRAN profile for the specification of transactions with atomicity, isolation and temporal correctness properties, which addresses **Subgoal 2**: Specify atomicity, isolation and temporal correctness of transactions, as well as the abort recovery, concurrency control and scheduling mechanisms, in a systematic manner.

- Papers C [32], D [33] and E [26] propose the UPPCART framework that models the transactions together with various mechanisms as UPPAAL TA, and model checks the properties using UPPAAL. Paper F [34] proposes UPPCART-SMC for probabilistic analysis of larger-scale RTDBMS that cannot be analyzed by applying UPPCART. These contributions addresses **Subgoal 3**: Facilitate flexible modeling of real-time transactions with abort recovery, concurrency control, and scheduling, as well as formal analysis of atomicity, isolation and temporal correctness.

- By applying our proposed methods to a series of industrial case studies, Papers A [29], B [30], E [26] and F [34] addresses **Subgoal 4**: Validate
the applicability and usefulness of the proposed methods via industrial use cases.

These contributions together provide a solution for designing customized real-time data management solutions based on systematic trade-offs of transactional properties, which addresses our overall research goal.
Chapter 6

Related Work

In this chapter, we discuss the related work with respect to customized transaction management in DBMS, data aggregation, as well as the formalization and analysis of transactions.

6.1 Design Methodologies for Customized Transaction Management

Designing effective and efficient transaction management to suit applications’ needs has been a hot topic in the development of DBMS. Various design methodologies have been proposed in literature for developing transaction management for DBMS. KIDS [15], an early effort to construct a customized DBMS, identifies transaction management as an aspect, and decomposes it into sub-aspects representing functionalities such as concurrency control and recovery. During the process proposed in KIDS, the database system designer has to decide the selection of run-time implementation techniques, based on the analysis of requirements and a classification of the implementation algorithms. AspectOPTIMA [17] proposes a reusable aspect-oriented framework for constructing customized transaction management. The designer selects particular aspects, such as a particular concurrency control algorithm and a recovery algorithm, which are composed to provide ACID assurance. These methodologies mainly focus on DBMS without real-time constraints. Moreover, since formal methods are not used to guide the design decisions, these methods do not enjoy the high degree of assurance provided by formal analysis.
PRISMA [14] is a software product-line-oriented process for requirement engineering of flexible transaction models, which supports identification, reasoning and composition of ACID variants and their sub-features in the requirement phase. Formal methods are applied to reason about the consistency of the selected ACID variants, but not to the analysis of the transactional behaviors under various selected transaction management mechanisms. Temporal correctness of transactions is not considered in PRISMA.

Mentis et al. [18] propose a model-driven approach for generating the implementation of selected transaction models. The implementation is modeled in state machines and verified against the ACID properties, after which the code of the transaction management can be generated. Temporal correctness is not considered in this approach.

Khachana et al. [16] propose an approach to produce a monolithic transaction processing system able to adjust the relaxation of ACID properties at runtime, according to business requirements through user interaction. This approach, however, does not perform design-time analysis on the transactional properties, and does not target real-time applications.

Several research efforts have been made to customize DBMS specially for embedded, real-time application domains. COMET [19] combines a component-based approach and aspect-oriented programing to build tailored RTDBMS. Encapsulating database functionalities as components, and crosscutting features such as transaction management as aspects, COMET generates a tailored RTDBMS by weaving the selected components and aspects together. In FAME-DBMS [20], functional requirements on a DBMS are represented as features, which are composed to construct DBMS variants. Built on top of FAME-DBMS, AUTODAMA [68] generates tailorable DBMS specially for automotive systems. In these methodologies, the selection of building modules is based on functional requirements analysis, as well as constraints on code size and performance. They mainly address resource consumption and footprint issues for embedded systems, rather than temporal correctness and the possible conflicts with ACID assurance. A more recent work in customized database management system is conducted by Wölfl [69], who proposes the FlyDB framework to generate tailored data management solutions for avionic systems from high-level models described in a domain specific language. In his work, the database product line incorporates commonalities and variations in data schema, storage, and access methods, especially designed for avionic systems. The primary goal is to achieve optimal memory footprint and response time. Although read and write operations, as well as simple referential integrity by foreign keys, are covered in the query models, FlyDB does not
consider more complex transaction semantics with concurrency control and recovery.

Compared with these aforementioned works, our DAGGERS process starts from deriving the real-time transaction models accounting for both temporal correctness and ACID properties. More importantly, we emphasize formal analysis of trade-offs between these properties, and focus on a systematic approach to analyze different decisions. By iteratively applying formal modeling and analysis of the transaction models, we are able to select the appropriate transaction management mechanisms that are assured to satisfy the desired transactional properties.

### 6.2 Taxonomies of Data Aggregation

Many researchers have promoted the understanding of data aggregation on various aspects. Among these works, considerable efforts have been made on the study of aggregate functions. Mesiar et al. [70], Marichal [71], and Rudas et al. [21] have studied the mathematical properties of aggregate functions, such as continuity and stability, and discussed these properties of common aggregate functions in detail. A procedure for the construction of an appropriate aggregate function is also proposed by Rudas et al. [21]. In order to design a software system that computes aggregation efficiently, Gray et al. [72] have classified aggregate functions into distributive, algebraic and holistic, depending on the amount of intermediate states required for partial aggregates. Later, in order to study the influence of aggregate functions on the performance of sensor data aggregation, Madden et al. [73] have extended Gray’s taxonomy, and classified aggregate functions according to their state requirements, tolerance of loss, duplicate sensitivity, and monotonicity. Fasolo et al. [37] classify aggregate functions with respect to four dimensions, which are lossy aggregation, duplicate sensitivity, resilience to losses/failures and correlation awareness. All these works above only address the characteristics of aggregate functions. Our taxonomy, although inspired by these works in the classification of aggregate functions, includes all constituents of data aggregation processes and the time constraints, and provides a feature-diagram-based representation to organize the commonalities and variabilities of data aggregation processes.

A large proportion of existing works have their focus on in-network data aggregation, which is commonly used in sensor networks. In-network aggregation is the process of processing and aggregating data at intermediate nodes when data are transmitted from sensor nodes to sinks through the network [37].
Besides a classification of aggregate functions that we have discussed in the previous paragraph, Fasolo et al. [37] classify the existing routing protocols according to the aggregation method, resilience to link failures, overhead to setup/maintain aggregation structure, scalability, resilience to node mobility, energy saving method and timing strategy. Their work, however, does not address the characteristics of the data, and does not provide a structured representation for the common and variable characteristics. The aggregation protocols are also classified by Solis et al. [74], Makhloufi et al. [75], and Rajagopalan [76], with respect to different classification criteria. In contrast to the above works focusing mainly on aggregation protocols, Alzaid et al. [77] have proposed a taxonomy of secure aggregation schemes that classifies them into different models. All these works differ from our taxonomy in that they provide taxonomies from a different perspective, such as network topology for instance. Instead, our work strives to understand the features and their implications of data aggregation processes and its constituents in design.

6.3 Formal Specification and Analysis of Transactions

Another large body of work related to this thesis is produced within the formalization and formal analysis of transactions. In the real-time community, the common technique to formally analyze transaction performance is schedulability analysis [78, 79]. However, this technique only analyzes the temporal correctness of transactions. To our knowledge, less attention has been devoted to studying to which extent logical data consistency can be achieved in the real-time paradigm. Many researchers have contributed to the verification of concurrent programs. For instance, Amighi et al. [80] applies theorem-proving based techniques to verify various properties, such as deadlock and data race freedom, of concurrent programs. Their technique, although with the potential to analyze transaction atomicity and isolation, requires high expertise to specify high-level transaction semantics and properties. In addition, temporal information and properties are not considered in their framework.

A substantial group of work on formal specification and analysis of transaction models is represented by the ACTA framework [94] and its successors, which are summarized in literature [14]. ACTA provides a first-order logic formalization to specify the transactional effects of data objects and the interaction between transactions, facilitating reasoning about transaction properties, as well as flexible synthesis of transaction models. Real-Time ACTA [95] ex-
<table>
<thead>
<tr>
<th>Related work</th>
<th>Properties</th>
<th>Formalism</th>
<th>Analysis Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mentis et al. [18]</td>
<td>ACID</td>
<td>state machines</td>
<td>model checking</td>
</tr>
<tr>
<td>Cerone et al. [81]</td>
<td>logical consistency models, no explicit ACID</td>
<td>axioms and rules</td>
<td>theorem proving</td>
</tr>
<tr>
<td>Bocchi et al. [82]</td>
<td>consistency</td>
<td>process calculi</td>
<td>model checking</td>
</tr>
<tr>
<td>Kokash et al. [83]</td>
<td>atomicity</td>
<td>Reo, constraint automata</td>
<td>model checking</td>
</tr>
<tr>
<td>Grov et al. [84] and Liu et al. [85]</td>
<td>atomicity, isolation</td>
<td>Real-Time Maude</td>
<td>model checking</td>
</tr>
<tr>
<td>Suryavanshi et al. [86]</td>
<td>various properties, no explicit ACID</td>
<td>Event-B</td>
<td>theorem proving</td>
</tr>
<tr>
<td>Kirchberg [87]</td>
<td>various properties, no explicit ACID</td>
<td>transition rules</td>
<td>model checking</td>
</tr>
<tr>
<td>Makni et al. [88]</td>
<td>various properties, no explicit ACID</td>
<td>SPIN</td>
<td>model checking</td>
</tr>
<tr>
<td>Gaaloul et al. [89]</td>
<td>atomicity</td>
<td>event calculus</td>
<td>model checking</td>
</tr>
<tr>
<td>Bourne [90]</td>
<td>atomicity</td>
<td>temporal logics</td>
<td>model checking</td>
</tr>
<tr>
<td>Lanotte et al. [91]</td>
<td>atomicity and timeliness</td>
<td>timed automata</td>
<td>model checking</td>
</tr>
<tr>
<td>Kot [92]</td>
<td>various properties, timeliness, no explicit ACID</td>
<td>timed automata</td>
<td>model checking</td>
</tr>
<tr>
<td>Liu et al. [93]</td>
<td>atomicity, isolation</td>
<td>Real-Time Maude</td>
<td>statistical model checking</td>
</tr>
</tbody>
</table>
tends ACTA with formalization of real-time constraints on transactions and data. However, the formal syntax and semantics for the specification of ACID variants provided by ACTA and Real-Time ACTA are limited, and tool support for verification is missing. SPECTRA [96], which improves the formal syntax of ACTA, and is used in KIDS for specifying transaction models, does not focus on the ACID and timeliness variants. GOLOG [97] improves ACTA by providing formal semantics for the building blocks, using situation calculus and tool support for simulation. However, organizing the building blocks with respect to ACID properties is not in their focus, and real-time properties are not supported. Used in the PRISMA process [14], SPLACID [98] improves ACTA by providing a more expressive and structured language support for ACID variants and their sub-features. Real-time properties are not considered in SPLACID.

In addition to the thread of ACTA, a variety of methods have been proposed to formalize transactions. For instance, Mentis et al. [18] model transaction models as state machines, and verify that the ACID properties can be satisfied by an implementation using model checking. Other works are intended for analyzing a selected subset of properties in particular scenarios. A list of non-ACTA related works are presented in Table 6.1. Compared with these works, our work is different in several aspects. First, our process strives to analyze both ACID and temporal correctness, which are not entirely covered by the aforementioned works. Second, we contribute to a general, reusable and flexible modeling approach for modeling ACID and temporal correctness properties, as well as the supporting transaction management mechanisms. In particular, our timed-automata framework provides flexible modeling capability for a range of concurrency control mechanisms by using skeletons and patterns.
Chapter 7

Conclusions and Future Work

In this thesis, we have proposed a systematic process called DAGGERS for developing customized database management systems with data aggregation support for real-time applications. Our contribution augments the existing DBSM design methodology with systematic decisions on the customization of transaction management mechanisms, based on the trade-offs between ACID and temporal correctness of the application-specific transactions. The DAGGERS process guides the selection of appropriate transaction management mechanisms for the RTDBMS, via systematic specification and formal analysis of the traded-off properties, which can offer a high-degree of assurance.

A series of techniques have been proposed to facilitate the proposed DAGGERS process. We have proposed the DAGGTAX taxonomy to specify the features of data aggregation processes, which are later modeled as transactions. Three design rules and a set of design heuristics have been proposed based on the taxonomy to provide guidance for the design of data aggregation processes. The consistency of a DAGGTAX-based specification can be checked formally and automatically by our SAFARE tool, using SAT solving techniques.

We have proposed a UML-profile called UTRAN, which supports high-level, explicit specification of transactions, transactional properties (atomicity, isolation and temporal correctness), as well as the underlying transactional management mechanisms (abort recovery, concurrency control, scheduling). We have proposed two formal frameworks, UPPCART and UPPCART-SMC, for the formal analysis of the transactions. Our UPPCART framework enables
pattern-based formal modeling of atomic, concurrent, real-time transactions, which allows the desired atomicity, isolation and temporal correctness properties to be model checked rigorously. Our UPPCART-SMC framework also offers pattern-based model construction, and allows large and complex DBMS to be analyzed by statistical model checking. The transformation from UTRAN to UPPCART and UPPCART-SMC is automated by our tool U²-Transformer.

The validation on a series of real-world industrial use cases have demonstrated that our proposed techniques are useful to the design of customized RTDBMS. Our taxonomy enhances the understanding of the designed system, and raises awareness about the characteristics that need to be considered during the design. By applying the taxonomy, design flaws could be spotted and fixed prior to implementation. The underlying feature model allows data aggregation processes to be constructed by composing reusable features, whose feasibility can be reasoned about while time-related parameters can be derived. Our timed-automata-based frameworks allows designers to model transactions with various transaction management mechanisms, and formally analyze the satisfiability of atomicity, isolation and temporal correctness. Among them, UPPCART provides formal proof for the designed system model with the selected transaction models and mechanisms. UPPCART-SMC provides probabilistic guarantee for the system design, and is suitable for larger scale systems. Based on the analysis, the designer can choose a desired trade-off and select the appropriate mechanisms that have been assured to achieve the selected trade off.

7.1 Limitations

Our DAGGERS process and the proposed methods assume finite numbers of data and transactions. The data access patterns of these transactions must be able to be identified and described during the design phase, which is the case in many real-time systems. Database systems with undecidable data or transactions are out of scope of our design methodology. A typical example of such systems is the cloud databases, where the amount of data can grow without bound, and transactions may be constructed dynamically at run time. In this case, one can neither specify all data in the system using DAGGTX, nor model the transactions during system design.

The taxonomy provides the common and variable features of data aggregation processes, with a focus on the high-level real-time properties, and based on our surveyed applications. Since the survey is by no means exhaustive, some
characteristics may not be included in the taxonomy if the applications demonstrating these characteristics are not covered in the survey. For instance, we have realized that in many telecommunication systems, probabilistic features may have impact on the real-time performance, and therefore are of interest to the system designers. To incorporate new application-specific features into the taxonomy, one may need to construct a new feature model. To achieve this, DAGG TAX can serve as a basis for the evolution of the new feature model, since it contains the basic constituents of data aggregation processes.

As for the formal models of the RTDBMS, we have only proposed patterns and their automated transformation for common pessimistic concurrency control algorithms. Other algorithms in the optimistic or version-based concurrency control categories are not considered in this work. We also do not consider other recovery algorithms than rollback and compensation. The patterns and transformation of other concurrency control and abort recovery mechanisms, however, can be constructed in the future using our principles.

Another limitation lies in the need of manual adjustments in the generated TA models. Although we have proposed automated construction of TA for UPPCART and UPPCART-SMC, which reduces modeling efforts to a great extent, manual adjustments of the models may still be needed, especially for the purpose of optimization with respect to verification time. This optimization can be related to the particular transaction workload and the assumed DBMS.

As shown in Paper E [26], if a transaction has only one read or write operation, whose execution time is sufficiently short, we may consider to abstract the entire transaction as one atomic unit, and remove the begin and commit operation patterns from its model. This reduces the number of states for the verification. Manual adjustment is also required when we iterate through different transaction management mechanisms. The designer needs to manually adjust the UTRAN model in order to select a different candidate mechanism, which can be transformed and analyzed automatically later.

### 7.2 Future Work

The work of this thesis opens several future research directions. One possible future work involves the methods for trading off other transaction properties and mechanisms, in addition to the ones already considered in our current work. In this thesis we have proposed the method of modeling transactions with abort recovery, concurrency control, and scheduling, with the aim of analyzing atomicity, isolation and temporal correctness. In the future work we can
develop methods to address the analysis of consistency and durability, together with the corresponding DBMS mechanisms.

An interesting direction is to provide methods for system generation in DAGGERS, such as generating data management code from the proved models, or generating configuration files for configurable DBMS. Another extension of DAGGERS is to introduce model-based testing, which creates test cases from the UML specifications and the formal models [99], in order to test the generated RTDBMS against the temporal and logical correctness requirements.

One future direction to advance our work is the integration of other formal methods into the DAGGERS process. In order to provide rigorous analysis for advanced constraints, techniques based on Satisfiability Modulo Theories (SMT) [54] can be integrated into SAFARE, which in the current version can only analyze boolean propositional formulas with SAT solving. Program verification tools and techniques [100, 101] can be integrated into the process for the verification of user-defined functions that implement the lock resolutions in the concurrency control algorithms, especially for their functional correctness. In this thesis, we assume that this code is correctly provided. In the future work, this code can also be verified formally to provide a more rigorous guarantee for the system design.
Bibliography


