Mälardalen University Press Licentiate Theses
No. 258

SYSTEMATIC DESIGN OF DATA MANAGEMENT FOR REAL-TIME DATA-INTENSIVE APPLICATIONS

Simin Cai

2017

School of Innovation, Design and Engineering

ISBN 978-91-7485-334-6
ISSN 1651-9256
Printed by E-Print AB, Stockholm, Sweden

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 the consistency of data, such computations must be both logically correct (producing correct and consistent results) and temporally correct (completing before specified deadlines). One solution to ensure logical and temporal correctness is to model these computations 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 are achieved by the customized RTDBMS with appropriate run-time mechanisms. However, developing such a data management solution with provided guarantees is not easy, partly due to inadequate support for systematic 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, and thus they may be propagated to the implementation. Secondly, trade-off analysis of conflicting properties, such as conflicts between transaction isolation and temporal correctness, is mainly performed ad-hoc, which increases the risk of unpredictable behavior.

In this thesis, we propose a systematic approach to develop transaction-based data management with data aggregation support for real-time systems. Our approach includes the following contributions: (i) a taxonomy of data aggregation, (ii) a process for customizing transaction models and RTDBMS, and (iii) a pattern-based method of modeling transactions in the timed automata framework, which we show how to verify with respect to transaction isolation and temporal correctness. Our proposed taxonomy of data aggregation processes helps in identifying their common and variable characteristics, based on which their implications can be reasoned about. Our proposed process

allows designers to derive transaction models with desired properties for the data-related computations from system requirements, and decide the appropriate run-time mechanisms for the customized RTDBMS to achieve the desired properties. To perform systematic trade-off analysis between transaction isolation and temporal correctness specifically, we propose a method to create formal models of transactions with concurrency control, based on which the isolation and temporal correctness properties can be verified by model checking, using the UPPAAL tool. By applying the proposed approach to the development of an industrial demonstrator, we validate the applicability of our approach.

To my parents

Acknowledgments

I would like to thank, first and foremost, my supervisors Associate Professor Cristina Seceleanu, Dr. Dag Nyström and Associate Professor Barbara Gallina. Thank you for your patience, guidance and encouragement to my research, as well as your optimism and perseverance that have inspired me during all these years. I also wish to express my gratitude to Alf Larsson from Ericsson, Bengt Gunne from Mimer, Detlef Scholle from Alten, as well as your colleagues, for the precious support and feedback to the DAGGERS project.

I also would like to thank my grading committee members Professor Sten F Andler and Professor Mikael Sjödin, especially my faculty examiner Associate Professor Luís Almeida. Thank you for your time and effort to review my thesis. It is an honor for me to have you in my grading committee.

Big hugs to my friends and colleagues at MDH (which will be a long list :-)), not only for the discussions and cooperations, but also for making these years a pleasant journey in my life. Many thanks to the (guest) professors and lecturers at MDH, who have provided me knowledge and inspiration to continue my research.

Many hugs to my friends, Yemao, Xueming, Nico, Wei, Tengjiao, Anders, Yanzhu, and more. Thank you for your company and friendship through thick and thin. Special thanks to Fredrik, for bringing me a more beautiful world since we met.

Last but not least, it is never enough to express my gratitude to my parents for their selfless love. Thank you for letting me choose my way, and supporting me with everything.

Simin Cai
Västerås, May, 2017

List of Publications

Papers Included in the Licentiate Thesis[1]

Paper A *DAGGTAX: A Taxonomy of Data Aggregation Processes.* Simin Cai, Barbara Gallina, Dag Nyström, and Cristina Seceleanu. Technical Report. Mälardalen Real-Time Research Center, Mälardalen University, Sweden, May 2017. A shorter version has been submitted to the 7th International Conference on Model & Data Engineering (MEDI), Springer.

Paper B *A Formal Approach for Flexible Modeling and Analysis of Transaction Timeliness and Isolation.* Simin Cai, Barbara Gallina, Dag Nyström, and Cristina Seceleanu. In Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS), Brest, France, 19-21st October 2016. ACM.

Paper C *Towards the Verification of Temporal Data Consistency in Real-Time Data Management.* Simin Cai, Barbara Gallina, Dag Nyström, and Cristina Seceleanu. In Proceedings of the 2nd International Workshop on modeling, analysis and control of complex Cyber-Physical Systems (CPSDATA), Vienna, Austria, April 2016. IEEE.

Paper D *Design of Cloud Monitoring Systems via DAGGTAX: a Case Study.* Simin Cai, Barbara Gallina, Dag Nyström, Cristina Seceleanu, and Alf Larsson. In Proceedings of the 8th International Conference on Ambient Systems, Networks and Technologies (ANT), Madeira, Portugal, May 2017. Elsevier.

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

Contents

I Thesis **1**

1 Introduction **3**
1.1 Thesis Overview 6

2 Preliminaries **11**
2.1 Data Aggregation 11
2.2 The Concept of Transaction 12
2.2.1 Relaxed ACID Properties 14
2.2.2 Pessimistic Concurrency Control 16
2.2.3 Real-time Transactions and Temporal Correctness . . 17
2.3 Model Checking Using UPPAAL 18
2.3.1 UPPAAL Timed Automata 18

3 Research Summary **21**
3.1 Problem Description 21
3.2 Research Goals 22
3.2.1 Research Subgoals 22
3.3 Research Method 23
3.4 Thesis Contributions 24
3.4.1 The DAGGERS Process 24
3.4.2 DAGGTAX: A Taxonomy of Data Aggregation Processes 26
3.4.3 A Timed-Automata-based Approach for Flexible Modeling and Verification of Isolation and Temporal Correctness 28
3.4.4 Validation on Industrial Use Cases 34
3.5 Research Goals Revisited 36

4 Related Work **39**
4.1 Taxonomies of Data Aggregation 39
4.2 Customized Transaction Management 40
4.2.1 Design methodologies for developing customized transaction management 40
4.2.2 Formalization and analysis of transaction models . . . 42

5 Conclusions and Future Work **45**
5.1 Future Work 46

Bibliography **49**

II Included Papers **57**

6 Paper A:
DAGGTAX: A Taxonomy of Data Aggregation Processes **59**
6.1 Introduction 61
6.2 Related Work 63
6.3 Preliminaries 64
6.3.1 Timeliness and Temporal Data Consistency 64
6.3.2 Feature Model and Feature Diagram 65
6.4 A Survey of Data Aggregation Processes 66
6.4.1 General-purpose Infrastructures 67
6.4.2 Ad Hoc Applications 71
6.4.3 Survey Results 76
6.5 Our Proposed Taxonomy 77
6.5.1 Raw Data 79
6.5.2 Aggregate Function 81
6.5.3 Aggregated Data 82
6.5.4 Triggering Pattern 83
6.5.5 Real-time (P) 84
6.6 Design Rules and Heuristics 84
6.6.1 Design Rules 85
6.6.2 Design Heuristics 86
6.7 Evaluation: an Industrial Case Study 89
6.7.1 Problem identified in the HAT design 91
6.7.2 Solutions 92
6.7.3 Comparison with other taxonomies 94

6.7.4 Summary 94
6.8 Discussion 95
6.9 Conclusion 96
Bibliography 97

7 Paper B: A Formal Approach for Flexible Modeling and Analysis of Transaction Timeliness and Isolation 103
7.1 Introduction 105
7.2 Preliminaries 106
7.2.1 The Concept of Transaction 106
7.2.2 Isolation 108
7.2.3 Pessimistic Concurrency Control (PCC) 109
7.2.4 Timed Automata and UPPAAL 110
7.3 Our Approach 111
7.3.1 Work Unit Skeleton and Operation Patterns 113
7.3.2 Concurrency Control Skeleton and Patterns 115
7.3.3 IsolationObserver Skeleton 117
7.3.4 Reference Algorithm: Rigorous 2PL 118
7.4 Adjustments for Various PCC 121
7.4.1 Concurrency Control for Relaxed Isolation 121
7.4.2 Real-time Concurrency Control 122
7.5 Verification 127
7.6 Related Work 131
7.7 Conclusion 132
Bibliography 135

8 Paper C: Towards the Verification of Temporal Data Consistency in Real-Time Data Management 139
8.1 Introduction 141
8.2 Background 142
8.2.1 Temporal Data Consistency 142
8.2.2 Timed Automata and UPPAAL 143
8.3 Assumed System 145
8.4 Modeling Transaction Work Units and Data 146
8.4.1 Modeling Transaction Work Units 147
8.4.2 Modeling the Age of Data 148
8.4.3 Modeling the Lock Manager 149

8.5 Verification of Temporal Data Consistency and Timeliness . . 152
8.5.1 Formalizing the Requirements 152
8.5.2 Verification Results 154
8.6 Related Work 154
8.7 Conclusion 156
Bibliography 159

9 Paper D:
Design of Cloud Monitoring Systems via DAGGTAX: a Case Study 161
9.1 Introduction 163
9.2 Background 165
9.3 Case Study and Results 168
9.3.1 Case Study Description 168
9.3.2 Application of DAGGTAX 169
9.3.3 System Implementation 171
9.4 Benefits of DAGGTAX 173
9.5 Related Work 175
9.6 Conclusion and Future Work 176
Bibliography 177

I

Thesis

Chapter 1

Introduction

Traditionally, real-time systems used to be closed systems with static well-defined functionalities managing small amounts of data. In recent years, however, real-time systems are designed to provide more advanced functionalities, with higher degrees of openness to other systems, and consequently become more data intensive. For instance, the amount of data managed by software is increasing in modern automotive systems. In some recent models, over 2500 signals are generated and processed in real-time [1]. In factory automation systems, hundreds of sensors are deployed to monitor the states of the working environment and the system, based on which time-constrained actions are taken to complete production work [2]. Managing large amounts of data has become an emerging challenge that the designers of real-time systems are facing.

Not only the amounts of data are growing, but also the data-related computations are becoming more complex in data-intensive real-time systems, since the latter needs to meet both temporal and logical constraints [3, 4]. On one hand, just as in traditional real-time systems, these computations need to satisfy the timeliness requirement, that is, to meet their deadlines. On the other hand, data-intensive real-time systems often bear a higher concern with respect to logical data consistency compared with the traditional ones. Concurrent access of data is common in data-intensive systems, which may introduce unwanted interference that harms logical data consistency. In addition, application semantics may entail various requirements, such as failure recovery and persistence of computation results, which also increases the complexity of the computations.

One common type of complex data-centric computation is data aggrega-

tion, which is defined as the process of producing a synthesized form of data from multiple data items using an aggregate function [5]. Compared to the raw data before aggregation, the aggregated data is usually smaller in size, often meaning reduced data storage and transmission costs, while the key information is still preserved. Therefore, data aggregation has been extensively applied in a variety of real-time applications, such as automotive [6] and avionic systems [7], in which data are aggregated from various sensors and electronic units. 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. In a factory automation system, for instance, a multi-level aggregation design can be adopted for production monitoring [2]. Condition data are collected from field devices such as different sensors, where the first-level data aggregation is performed. The aggregated results are transmitted to, and further aggregated in, the production cell controllers, the factory monitoring system, and even in the cloud. In such a system, each level of aggregation may have its unique characteristics, not only in the functional aspects implemented by specific aggregate functions, but also in non-functional properties [8] including logical and temporal correctness.

A promising solution to manage the increasing amounts of data and the complex computations is to use Real-Time DataBase Management Systems (RTDBMS) for structured data management. For instance, Almeida et al. [9] adopt an RTDBMS to manage real-time sensor data and coordinate the actions of autonomous agents. In an RTDBMS, data-related computations are modeled as *transactions*, which are collections of logically related operations on data, which maintain both *logical data consistency* and *temporal correctness* [3]. Among them, logical data consistency is maintained by ensuring the so-called *ACID* properties, which refer to, respectively: *Atomicity* (a transaction either runs completely or rollbacks all changes), *Consistency* (a transaction executing by itself does not violate logical constraints), *Isolation* (uncommitted changes of one transaction shall not be seen by concurrent transactions), and *Durability* (committed changes are made permanent) [10, 11]. Temporal correctness includes two aspects: *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 [3, 12]. Ideally, both logical data consistency and temporal correctness should be guaranteed by the RTDBMS. However, conflicts could arise when temporal correctness is breached due to the unpredictability introduced by the transaction management mechanisms for ACID

such as concurrency control. In such situations, ACID assurance is often relaxed in favor of temporal correctness [13]. Depending on the specific application semantics, the relaxation of ACID could vary, in the spectrum of one or more properties, that is, A, C, I, and D [14].

In order to design a transaction-based data management solution for a data-intensive real-time system, the system designer needs to model the data-related computations as transactions, and design an RTDBMS to manage these transactions so that the desired correctness requirements are met. To our knowledge, existing DBMS design methodologies [15, 16, 17, 18, 14, 19, 20] do not provide support for the systematic analysis of trade-offs between logical data consistency and temporal correctness. Following these methodologies, designers may either design data management solutions without being aware of their impact on logical data consistency and temporal correctness, or choose an inappropriate relaxation of ACID properties and the transaction management mechanisms without sound analysis. Consequently, the risks of failing to identify conflicting properties in the design increase, and hence such conflicts could propagate to the implementation phase, leading to system-level failures during the execution.

In this thesis, we investigate how to design data aggregation and management systematically for data-intensive real-time systems. We propose an engineering process called DAGGERS (Data AGGregation for Embedded Real-time Systems) [21] to systematically develop RTDBMS customized for systems that need to trade-off between logical data consistency and temporal correctness. The DAGGERS process consists of the following steps: (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; (iii) Generating the RTDBMS using the selected transaction model and the mechanisms.

We focus on techniques to facilitate step (i) and (ii) in this thesis. Concretely, for step (i) we propose a taxonomy for data aggregation and its properties. We do this since this type of data-related computation is essential in many applications, yet a structured knowledge base for data aggregation processes is missing, which hinders applying systematic analysis in the design. For step (ii), we propose a timed-automata-based approach to model data-related computations, and analyze the trade-offs between logical data consistency and temporal correctness using model-checking techniques. We especially focus on the trade-offs between isolation and temporal correctness, and the selection

of the appropriate concurrency control algorithm. However, the approach can be extended for the analysis of other transactional properties.

To ensure applicability, we validate the proposed solutions on two industrial projects, with the engineers from industry in the loop. The validation shows that our proposed taxonomy can help to ease the effort in designing data aggregation for real-time data-intensive systems, and prevent software design flaws prior to implementation. The validation of the entire approach will be performed in our future work.

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), a brief description of our research goals, methods and contributions (Chapter 3), a discussion on the related work (Chapter 4), and conclusions and future work (Chapter 5).

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

Paper A *DAGGTAX: A Taxonomy of Data Aggregation Processes.* Simin Cai, Barbara Gallina, Dag Nyström, and Cristina Seceleanu. Technical Report. Mälardalen Real-Time Research Center, Mälardalen University, Sweden, May 2017. A shorter version has been submitted to the 7th International Conference on Model & Data Engineering (MEDI).

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. Due to the heterogeneity and real-time constraints in such systems, designing appropriate data aggregation processes often demands considerable efforts. A study on the characteristics of data aggregation processes will provide a comprehensive view for the designers, and facilitate potential tool support to ease the design process. In this paper, we propose a taxonomy called DAGGTAX, which is a feature diagram that models the common and variable characteristics of data aggregation processes, especially focusing on the real-time aspect. The taxonomy can serve as the foundation of a design tool that enables designers to build an aggregation process by selecting and composing desired features, and to reason about the feasibility of the design. We also provide a set of design heuristics that could help designers to decide the appropriate mechanisms for achieving the selected features. Our

industrial case study demonstrates that DAGGTAX not only strengthens the understanding, but also facilitates the model-driven design of data aggregation processes.

Paper contribution: I was the main driver of the paper. I performed the survey on data aggregation processes and proposed the taxonomy. I also conducted the industrial case study and wrote the paper. The other authors contributed with important ideas and comments.

Paper B *A Formal Approach for Flexible Modeling and Analysis of Transaction Timeliness and Isolation.* Simin Cai, Barbara Gallina, Dag Nyström, and Cristina Seceleanu. In Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS), Brest, France, 19-21st October 2016. ACM.

Abstract: Traditional Concurrency Control (CC) mechanisms ensure absence of undesired interference in transaction-based systems and enforce isolation. However, CC may introduce unpredictable delays that could lead to breached timeliness, which is unwanted for real-time transactions. To avoid deadline misses, some CC algorithms relax isolation in favor of timeliness, whereas others 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 modularization for the basic transactional constituents, which enables flexible modeling and composition of various candidate CC algorithms, and thus reduces the effort of selecting the appropriate CC algorithm.

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

Paper C *Towards the Verification of Temporal Data Consistency in Real-Time Data Management.* Simin Cai, Barbara Gallina, Dag Nyström, and Cristina Seceleanu. In Proceedings of the 2nd International Workshop on modeling, analysis and control of complex Cyber-Physical Systems (CPSDATA),

Vienna, Austria, April 2016. IEEE.

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 (RTDBMS) must ensure both transaction timeliness and temporal data consistency. RTDBMS 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.

Paper contribution: I was the main driver of the paper. I developed the formal models and performed the verification, and wrote the paper. The other authors contributed with important ideas and comments.

Paper D *Design of Cloud Monitoring Systems via DAGGTAX: a Case Study.* Simin Cai, Barbara Gallina, Dag Nyström, Cristina Seceleanu, and Alf Larsson. In Proceedings of the 8th International Conference on Ambient Systems, Networks and Technologies (ANT), Madeira, Portugal, May 2017. Elsevier.

Abstract: Efficient auto-scaling of cloud resources relies on the monitoring of the cloud, which involves multiple aggregation processes and large amounts of data with various and interdependent requirements. A systematic way of describing the data together with the possible aggregations is beneficial for designers to reason about the properties of these aspects as well as their implications on the design, thus improving quality and lowering development costs. In this paper, we propose to apply DAGGTAX, a feature-oriented taxonomy for organizing common and variable data and aggregation process properties, to the design of cloud monitoring systems. We demonstrate the effectiveness of DAGGTAX via a case study provided by industry, which aims to design a cloud monitoring system that serves auto-scaling for a video streaming system. We design the cloud monitoring system by selecting and composing DAGGTAX

features, and reason about the feasibility of the selected features. The case study shows that the application of DAGGTAX can help designers to identify reusable features, analyze trade-offs between selected features, and derive crucial system parameters.

Paper contribution: I was the main driver of the paper. I applied the taxonomy, designed the system, implemented a prototype, and wrote the paper. Alf Larsson provided the industrial use case, and useful comments. The other authors contributed with important ideas and comments.

Chapter 2

Preliminaries

In this chapter we present the needed preliminaries of this thesis. We first present the background knowledge about data aggregation, a common type of data-related computations considered in this thesis. We then recall the concept of transaction, including the relaxation of ACID in RTDBMS. After that, we briefly present the basics of model checking with UPPAAL, which is the formal analysis technique used in this thesis.

2.1 Data Aggregation

Data aggregation is the process of producing a synthesized form of data from multiple data items using an aggregate function [5]. It is applied extensively in information systems [5, 22, 23]. For instance, in database systems, data tuples are aggregated to compute statistical values; in resource-constrained systems, large amounts of data are aggregated to save storage or transmission resources; in systems concerning privacy and security, aggregation of details 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 [24]. VigilNet exploits four levels of aggregation to perform real-time surveillance [25], as shown in Figure 2.1. The first-level aggregation takes place in the sensor layer,

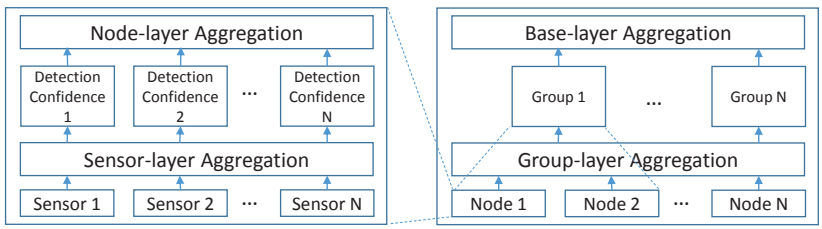

Figure 2.1: Data aggregation architecture of VigilNet [25]

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 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 6.

2.2 The Concept of Transaction

A *transaction* is a partially-ordered set of logically-related operations on the database, which as a whole guarantee the logical data consistency [10], that is, satisfying a set of integrity constraints imposed on the database [26]. 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 [10, 11]. Atomicity refers to the "all-or-nothing" semantics, meaning that if a transaction fails before completion, all its changes should be rolled back. 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. A database management system enforces these properties by applying various mechanisms to the transaction management. For instance, various logging and recovery mechanisms are among the choices for maintaining atomicity and durability, while in modern DBMS, concurrency control techniques are applied to achieve isolation [27].

Program 2.1: Transaction T that transfers 100 from A to B

```
Begin
        read A
        subtract(A, 100)
        write A
        if A<0, Abort
        read B
        add(B, 100)
        write B
Commit
```

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 aborted during the execution, 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).

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 [28, 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 [19]. Second, full ACID assurance may not be possible under the particular system constraints. As stated in the CAP theorem [29], in distributed database systems with network Partitions (P), trade-offs always occur between logical data Consistency (C) and Availability (A). Further, the PACELC theorem [30] 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.

As an example, 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 research 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 [31], 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 [32], 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. For more information about transaction models and relaxation variants of ACID, we refer the readers to literature [14].

In this thesis, we are particularly interested in the *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 [33], and later extended and generalized by Berenson et al. [34] and Adya et al. [35]. An isolation level is defined as the property of avoiding a particular subset of phenomena (or anomalies), that is, the interferences caused by concurrent execution [35, 33]. Assuming that $T1$ and $T2$ are two transactions as defined previously, we describe the phenomena introduced by the SQL-92 standard as follows:

- **Dirty Read.** Transaction $T2$ reads a data item that was modified by transaction $T1$ before $T1$ commits. If $T1$ is rolled back, the data read by $T2$ is not valid.

- **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 $T1$ become 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

Table 2.1: Isolation levels in the ANSI/ISO SQL-92 standard [33]

Isolation level	Dirty read	Non-repeatable Read	Phantom
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	Not Possible	Possible	Possible
REPEATABLE READS	Not Possible	Not Possible	Possible
SERIALIZABILITY	Not Possible	Not Possible	Not Possible

listed in Table 2.1, the SERIALIZABLE level precludes all types of phenomena, whereas other levels can be defined to preclude a selected set of phenomena.

2.2.2 Pessimistic Concurrency Control

In order to achieve isolation, a DBMS applies *concurrency control* that regulates the execution of concurrent transactions and prevents unwanted interferences. Among various types of concurrency control applied in DBMS, in this thesis we focus on one of the most common type called Pessimistic Concurrency Control (PCC), which employs locking techniques to prevent interferences [27]. 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 the lock, wait, or be aborted, when lock conflicts occur [27].

A wide range of PCC algorithms have been proposed in literature [27]. 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. [36] and Berenson et al. [34], one can achieve the different SQL-92 isolation levels by adjusting the lock types and locking durations (Table 2.2). In this table, a lock on a data item refers to the fact that a 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.

Table 2.2: SQL-92 isolation levels achieved by adjusting locks [36, 34]

Isolation level	Read locks on data	Write locks on data
READ UNCOMMITTED	no locks	long write locks
READ COMMITTED	short read locks on data item	long write locks on data item
REPEATABLE READS	long read locks on data item, short read locks on phantom	long write locks on data item
SERIALIZABILITY	long read locks on both data item and phantom	long write locks on data item

2.2.3 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 [3]. 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* [12]. 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 other real-time data (base data) is relatively valid, if the base data are sampled within a specified relative validity interval.

In RTDBMS with hard 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 [13]. For instance, it is common to relax durability, since disk I/O for storing/accessing persistent data is often considered too unpredictable for RTDBMS. 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. Therefore, RTBDMS may choose a concurrency control algorithm that achieves relaxed isolation, and improves timeliness [37, 38, 39].

In this thesis, we mainly focus on the formal verification of isolation and temporal correctness of real-time transactions, to ensure that a chosen concurrency control algorithm achieves the desired relaxation level while preserving temporal correctness. The proposed framework is presented in Chapter 7 and 8.

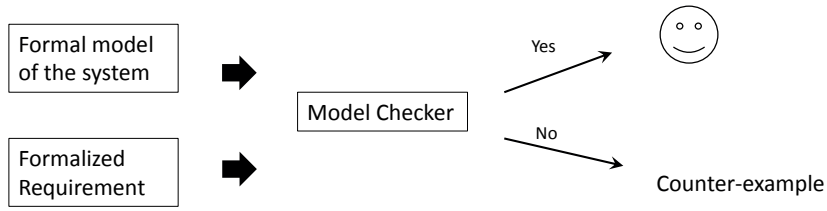

Figure 2.2: Model-checking technique

2.3 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 exhaustively and automatically exploring all possible states of the model [40]. An overview of the model-checking technique is shown in Figure 2.2. The formal model of the system is described in a language such as UPPAAL Timed Automata [41] presented in Chapter 2.3.1. The properties to be verified are formalized in some logic, in our case as temporal logic (Timed Computation Tree Logic [42]) formulas. The model checker implementing a model-checking algorithm can then automatically verify whether the properties are satisfied by the system.

The properties verified by a model checker are of two main types: (i) *safety* properties, of the form "something (bad) will never happen", and (ii) *liveness* properties, of the form "something (good) will eventually happen". In this thesis, we focus on verifying only safety properties, as exemplified in Chapter 2.3.1. The result of the verification given by the model checker is a "yes/no" answer, indicating that the verified property is satisfied/violated, respectively. For safety properties, if a "no" answer is given, a model execution trace could be returned that acts as a counterexample to the safety property, as shown in Fig 2.2.

2.3.1 UPPAAL Timed Automata

In this thesis, we use the Timed Automata (TA) formal framework [43] to model real-time transactions, and the UPPAAL model checker [41] 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, which uses an extended version of TA for modeling, called UPPAAL TA in this thesis.

Timed automata [43] are finite-state automata extended with real-valued clock variables. As mentioned previously, UPPAAL TA [41] extends TA with discrete variables as well as other modeling features, like urgent and committed locations, synchronization channels, etc. A real-time system can be modeled as a network of TA composed via the parallel composition operator ("||"), which allows an individual automaton to carry out internal actions, while pairs of automata can perform handshake synchronization. The locations of all automata, together with the clock valuations, define the state of a TA.

We illustrate the basics of UPPAAL TA via a simple example. For more details, we refer the readers to the literature [41]. Figure 2.3 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: (i) delay as long as the invariant, which is a conjunction of boolean conditions expressed as clock constraints associated to the location, is satisfied; (ii) take a transition along an edge from this location, as long as the specified guard, which is a conjunction of constraints on discrete variables or clock variables, is satisfied. In Figure 2.3a, A1 may stay at L2 until the value of *cl* reaches 3, or move to L2 when the value of *c1* is greater than 1. While moving from L2 to L3, A1 synchronizes with automaton A2 via handshake synchronization, by using a synchronization channel *ch*. An exclamation mark "!" following the channel name denotes the sender, and a question mark "?" denotes the receiver. An assignment resets the clock or sets a discrete variable when an edge is traversed. Guards and assignments can be user-defined functions. In our example, when A1 moves from L2 to L3, the value of *a* is incremented by the function *inc(a)*.

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.

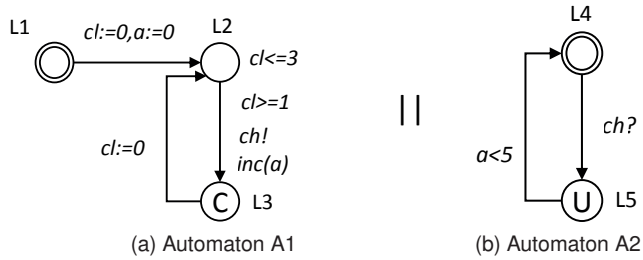

Figure 2.3: Example of a network of UPPAAL TA: A1 || A2

The properties to be verified by model checking the resulting network of timed automata are specified in a decidable subset of (Timed) Computation Tree Logic ((T)CTL) [42], and checked by the UPPAAL model checker. UPPAAL supports verification of liveness and safety properties [41]. As mentioned, in this thesis, we focus on verifying only safety properties. For instance, one can specify the safety property "A1 never reaches location L2" as "$A[\,]\ not\ A1.L2$", in which "A" is a path quantifier and reads "for all paths", whereas "$[\,]$" is the "always" path-specific temporal operator. If a safety property is not satisfied, a counterexample will be provided by UPPAAL. We refer the readers to literature [41] for more information about UPPAAL.

Chapter 3

Research Summary

In this chapter, we present a summary of our research. We formulate the research problem and research goals, describe the research method applied in our research, and present the contributions of the thesis.

3.1 Problem Description

As the amount of data and the complexity of computations are growing, RTDBMS with data aggregation support can be promising for managing the logical data consistency and temporal correctness in real-time data-intensive systems. Designing an RTDBMS is not a trivial task, as full ACID assurance for logical data consistency may need to be relaxed in order to satisfy temporal correctness. Due to a lack of support for systematic analysis, the relaxations are often decided by designers in an ad-hoc manner, which could lead to inappropriate designs that fail to satisfy the desired properties.

To overcome the drawback of ad-hoc design, systematic analysis support is needed for the design of RTDBMS with data aggregation. To achieve this, several issues need to be addressed. First, a methodology is lacking that guides the designer to systematically decide an appropriate ACID relaxation from a rich spectrum of possible choices. Second, the characteristics of the data aggregation computations, as well as their implications with respect to logical data consistency and temporal correctness, are essential to systematic design, but they are not well-understood. Last but not least, existing techniques cannot provide assurance that the selected run-time mechanisms for the RTDBMS can

guarantee the decided trade-offs.

3.2 Research Goals

Given the aforementioned problems, we present our overall research goal and the concrete subgoals in this section. Our overall research goal is formulated as follows:

Overall Research Goal. *Enable the systematic design of transaction-based data management with data aggregation support for real-time systems, so that the desired ACID properties and temporal correctness are guaranteed.*

3.2.1 Research Subgoals

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. *Identify and classify the common and variable characteristics of data aggregation such that they can be systematically reasoned about.*

When designing RTDBMS, the challenge is how to derive a transaction model with the appropriate transactional properties, and decide the appropriate mechanisms from a set of candidates, based on systematic analysis. In particular, in this thesis we focus on the selection of concurrency control algorithms based on analysis of trade offs between isolation and temporal correctness. Due to the large number of candidate concurrency control algorithms, we need to find a way to provide flexibility in the modeling and the analysis of the algorithms together with the transactions. Based on this, we formulate our second research subgoal as follows:

Subgoal 2. *Design a method that allows for flexible modeling of real-time transactions and concurrency control, and verification of isolation and temporal correctness.*

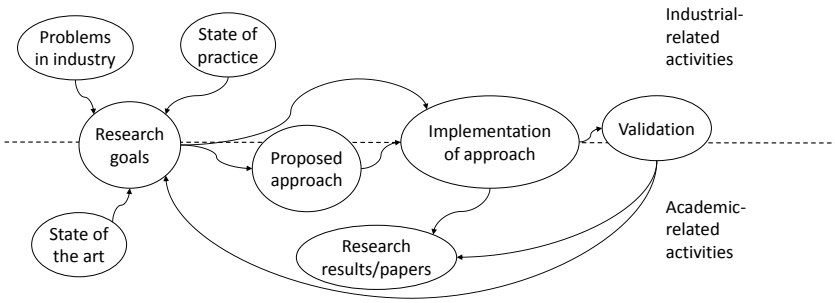

Figure 3.1: The research process of the thesis

The next issue to research into after meeting subgoal 2 is the applicability and usefulness of the proposed approach. Therefore, our third subgoal is presented as follows:

Subgoal 3. *Validate the applicability and usefulness of the proposed approach on an industrial use case.*

3.3 Research Method

In this section we introduce the methods that we use 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 3.1. This research is initiated by industrial problems that have not been solved by industrial solutions nor thoroughly studied by academic researchers. Based on the industrial problems, the state of practice and the state of the art, we formulate the research goals. To address these goals, we propose a systematic approach, and implement techniques to facilitate the approach, which could be applied to industrial applications. Finally, we validate the approach by applying it to the development of an industrial application. 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 "crit-

ical analysis of literature" method [44] to study the state of the art and state of practice of the researched area. We gather literature in areas including data aggregation, real-time data management, transaction modeling, etc., and critically analyze the challenges, approaches and solutions related to our research goals. During the implementation of our approach, we apply the "proof of concepts" method [44] to show the correctness and applicability of our proposed approach. When validating the research with industry, we apply the "proof by demonstration" method [44], by developing a demonstrator in an industrial setting using our proposed approach. The developed demonstrator, as well as the development process, are eventually evaluated with respect to our research goals by both the researchers and the industrial partners.

3.4 Thesis Contributions

In this section, we present the technical contributions of this thesis, which address the aforementioned research goals.

3.4.1 The DAGGERS Process

As a first step towards reaching our overall research goal 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 is a methodology intended to tackle the overall research goal.

An overview of the DAGGERS process is presented in Fig 3.2, including three main steps as follows.

Step I: 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

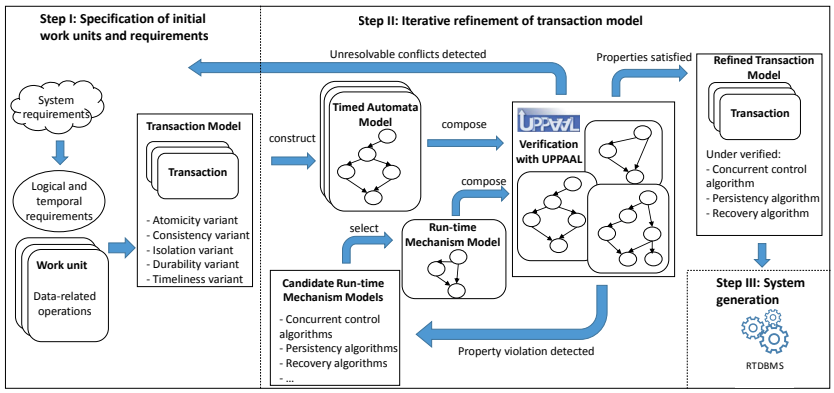

Figure 3.2: The DAGGERS process

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 model the work units as a set of timed automata [43], on which the transactional properties are specified formally, and can be checked by the UPPAAL model checker. We also assume that a repository of timed-automata models of commonly used run-time mechanisms has been prepared, which can be reused and composed with the timed automata of the work units. The models are checked iteratively. If model-checking shows that unsolvable conflicts occur with a particular candidate run-time mechanism, this mechanism is replaced by another candidate, and the models are verified again. This iterative process continues until all properties are satisfied by some selected mechanisms.

The refinement is the iterative "select-check" process as follows. First, we select one candidate mechanism from the repository and form a network of timed automata with the work unit models. Second, we model check the automata network 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 ensure 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 this step is the refined transaction models that are proved to achieve the appropriate ACID and temporal correctness variants under the selected run-time mechanisms.

Step III: System generation. With the verified transaction models, the designer can implement the transactions in SQL or other programming languages. In addition, a customized RTDBMS can be generated by composing or configuring the verified run-time mechanisms. In this thesis, we only focus on the RTDBMS design, while leaving the system generation as future work.

3.4.2 DAGGTAX: A Taxonomy of Data Aggregation Processes

In order to gain the knowledge for the systematic analysis of data aggregation, we have extensively surveyed data aggregation processes as proposed in theory and used in practice, and investigated their common and variable characteristics. Based on the survey results, we propose a taxonomy of data aggregation processes, called DAGGTAX (Data AGGregation TAXonomy).

The proposed taxonomy is presented as a feature diagram [45], in which each characteristic is modeled as a feature. It covers the common and variable characteristics of the main constituents of an aggregation process, which are the *raw data*, the *aggregate function* and the *aggregated data*. It also covers the features of the *triggering patterns* of the process, as well as the *real-time properties*.

Figure 3.3 presents the overview 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 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

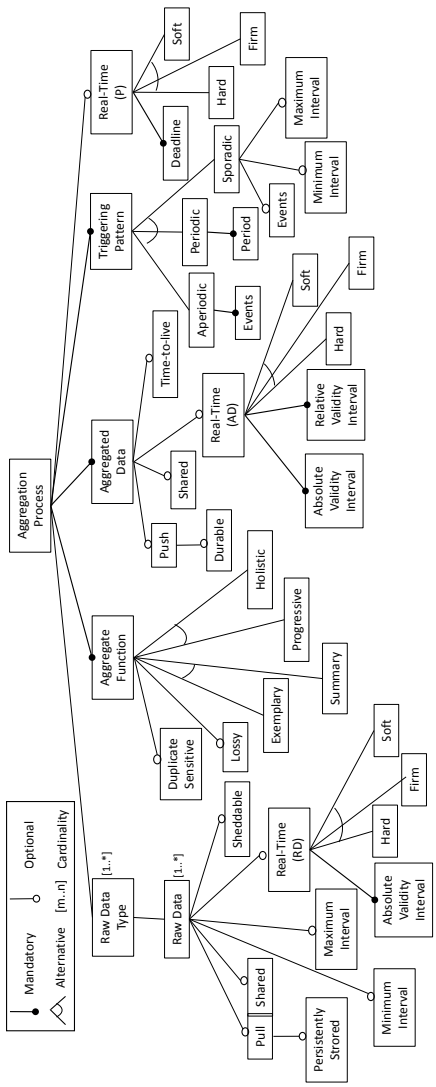

Figure 3.3: Our proposed taxonomy of data aggregation processes

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 [m..n] ($n \geq m \geq 0$) annotated with 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 3.3, a data aggregation process may have at least one "raw data type".

Our taxonomy provides a comprehensive view of data aggregation processes for the designers. Using the taxonomy, a data aggregation process can be constructed via the selection of desired features and their combination. Based on the taxonomy, we have introduced three design rules that eliminate some of the infeasible combinations of features during the design. For instance, a data aggregation process designed with both "soft" "real-time (P)" and "hard" "real-time (AD)" features are considered infeasible, as a soft real-time process may miss its deadline, and hence cannot guarantee hard real-time aggregated data produced by the former. We have also proposed a set of design heuristics to help the designer to 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.

This contribution is proposed in Paper A. It addresses subgoal 1 by providing a structured way of representing knowledge encompassing the common and variable characteristics of data aggregation processes. The feature-oriented representation of the taxonomy allows for potential systematic analysis via tool support. DAGGTAX raises the awareness of the dependencies between the data and the aggregation process, as well as their implications to system properties, which helps designers to reason about the designs systematically, and eliminate infeasible designs prior to implementation.

3.4.3 A Timed-Automata-based Approach for Flexible Modeling and Verification of Isolation and Temporal Correctness

As another contribution to address our research goals, we propose a timed-automata-based approach for modeling real-time concurrent transaction systems, and model checking isolation and temporal correctness under various concurrency control algorithms.

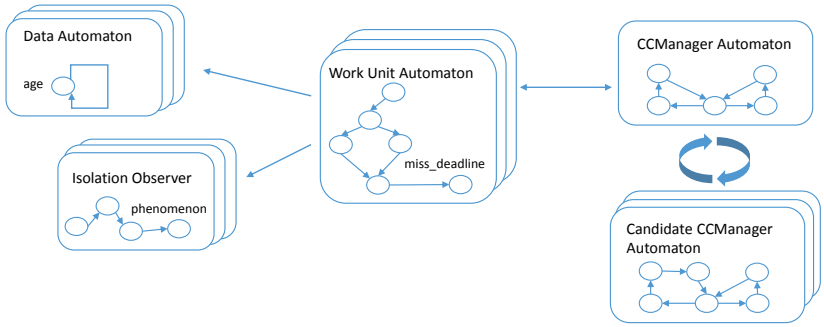

Figure 3.4: Our modeling framework

Modeling. Figure 3.4 shows the framework for modeling a real-time concurrent transaction system. We model the entire system as a timed-automata network, which consists of four types of automata: a set of work unit automata, a set of IsolationObserver automata, a set of Data automata, and a CCManager (Concurrency Control Manager) automaton.

We propose a set of automata skeletons and parameterized patterns as basic modeling blocks to reduce the modeling effort. Such skeletons model the basic structures of transactional constituents and common concurrency control algorithms, while the parameterized patterns model finer-grained recurring database operations, such as reads and writes. In the following text we explain our modeling framework in detail.

A work unit automaton models the work unit of a transaction as well as the interactions with the concurrency control manager. For each work unit automaton, we define a clock variable to trace the time spent by the transaction, and a location *miss_deadline* to represent the status of timeliness being breached. This location is reached only if the clock value exceeds a predefined deadline. Figure 3.5 presents the skeleton for a basic work unit with the locations *begin* and *end*, representing the boundary of the work unit, and a set of operation patterns modeling the data-related operations. This basic work unit skeleton can be extended with *commit_trans* and *abort* to represent successful and failed termination under the full atomicity and durability assumption, as shown in Figure 3.6. In this figure, a clock variable *tc* is defined to trace the elapsed time, and the *miss_deadline* location represents the violation of timeliness.

The work unit skeletons are enriched with instantiated parameterized patterns for data-related operations. An example of the parameterized pattern for

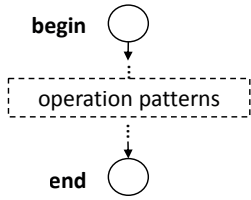

Figure 3.5: Timed automaton skeleton for a work unit

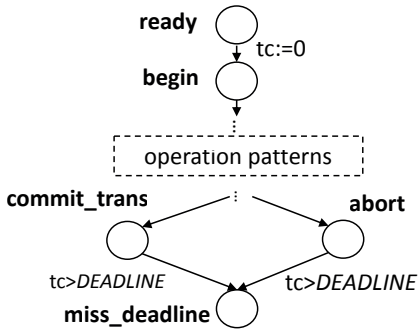

Figure 3.6: Work unit skeleton with atomicity and durability

read/write operations is presented in Figure 3.7. In this example, *tp* is a clock variable modeling the time, while *cs* is a discrete variable modeling the CPU resource. If the CPU is taken (indicated by the guard *cs==1*), the work unit automaton moves to the *wait* location. Otherwise, the automaton moves to the *operation* location. The invariant in this location ($tp <= WCRT$) constrains the automaton to stay at this location for at most *WCRT* time units, which is the worst-case response time of this operation. The guard $tp >= BCRT$ constrains the automaton to stay at *operation* for at least *BCRT* time units, which is the best-case response time of this operation. As long as both constraints are satisfied, the automaton can move to *operation_done*, and sets the CPU free.

An IsolationObserver automaton is created to monitor a concurrency phenomenon that should be precluded by a particular isolation level. If a monitored phenomenon occurs, the IsolationObserver will reach the location representing the phenomenon, indicating that isolation is breached. The automaton skeleton for an IsolationObserver is described in Figure 3.8. Since a phenomenon is defined as a particular sequence of operations, we let the IsolationObserver re-

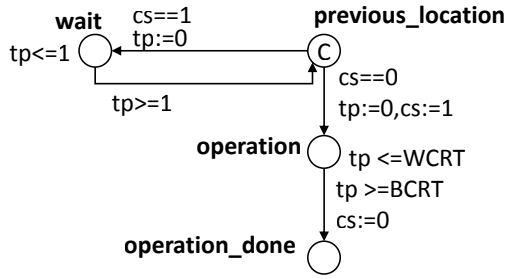

Figure 3.7: Read/write operation pattern

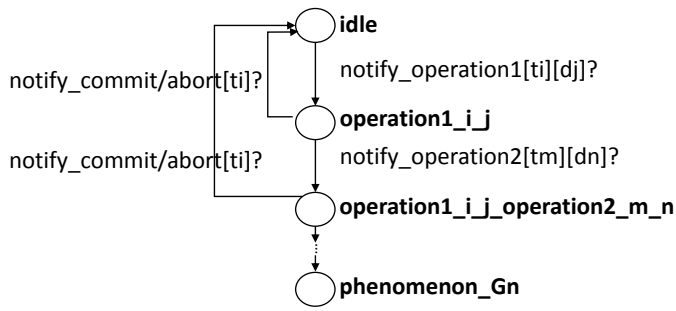

Figure 3.8: Automaton skeleton for an IsolationObserver

ceive the synchronization signals from the work units when such operations are performed. For instance, when work unit T_i successfully completes an operation on data D_j, it broadcasts the signal via channel *notify_operation1[ti][dj]*. When the IsolationObserver receives this signal, it moves from the *idle* location to the *operation1_i_j* location. If the sequence of operations that defines phenomenon Gn does occur, the IsolationObserver will eventually reach the *phenomenon_Gn* location.

A Data automaton models a data instance accessed by transactions. We define a clock variable "age" to trace the age of the data instance, which is reset when the data is updated. The skeleton for a data instance is shown in Figure 3.9.

The CCManager automaton models the concurrency control manager that applies a selected concurrency control algorithm. In Figure 3.10, we present the automaton skeleton for a PCC manager as an example. When the automa-

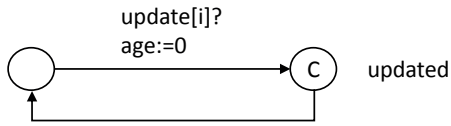

Figure 3.9: Automaton skeleton for a data instance

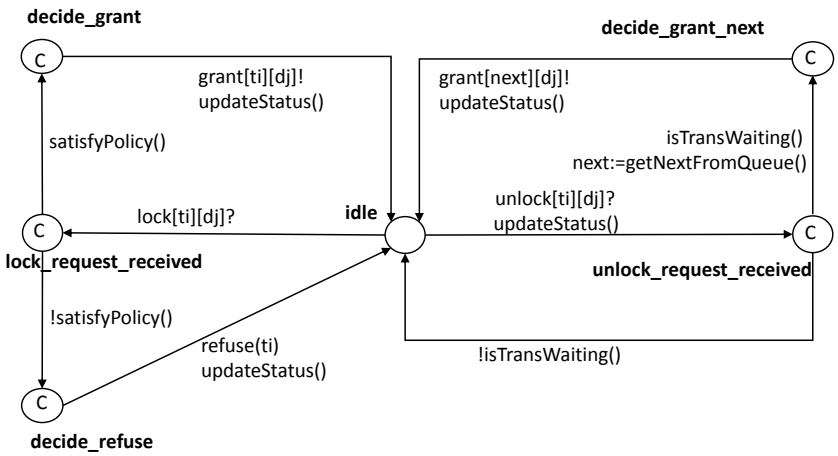

Figure 3.10: Automaton skeleton for a PCC manager

ton receives a locking request via the channel *lock[ti][dj]?*, it takes the transition from the initial location *idle* to *lock_request_received*. A user-defined function called *satisfyPolicy()*, which implements the lock request resolution of the modeled concurrency control algorithm, is defined as a guard on the edges from *lock_request_received*. Taking Two-Phase Locking (2PL [27]) as an example, *satisfyPolicy()* evaluates to false if the data required by the transaction has already been locked by another transaction. If *satisfyPolicy()* returns true, the automaton moves to *decide_grant*. It then immediately sends the signal *grant[ti][dj]!* to transaction T_i, and updates the status of the transactions and the locks, using a user-defined function *updateStatus()*. If *satisfyPolicy()* returns false, the CCManager moves to *decide_deny*, and takes actions as implemented in function *deny()*, before it moves back to *idle*. Since the CCManager has the highest priority, and the time on lock resolution is negligible, all locations in this model are committed locations. When receiving an unlocking

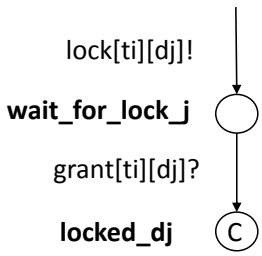

Figure 3.11: Locking pattern

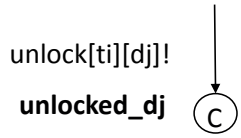

Figure 3.12: Unlocking pattern

request via *unlock[ti][dj]?*, CCManager updates the status, and moves to *unlock_request_received*. On the transitions from this location, the guards check if any transaction is waiting for locking the data, by a user-defined function *isTransWaiting()*. If this function returns true, the automaton sends a signal via *grant[next][dj]!*, to the next transaction obtained by the *getNextFromQueue()* function, and updates the status accordingly.

We also propose patterns for locking and unlocking operations, as presented in Figure 3.11 and Figure 3.12 respectively. After the transaction sends a message via *lock[ti][dj]!*, it waits at location *wait_for_lock_j*, until it receives the message *grant[ti][dj]?*. The patterns can be inserted into the work unit automata at particular positions depending on the selected PCC algorithm.

Using these skeletons and patterns, one can easily compose a network of automata to model a real-time concurrent transaction-based system. Our framework allows different PCC algorithms to be modeled and composed with the rest of the system flexibly. To model the different types of locks and durations, the designer only needs to adjust the types and locations of the instantiated locking/unlocking patterns. The resolution policies for a different PCC algorithm can also be implemented easily since it is well-encapsulated in user-defined functions.

Formal Verification. The properties that we try to verify can be checked against the timed-automata network model constructed using our framework. We specify the properties to be verified as (Timed) Computation Tree Logic ((T)CTL) formulas, as presented previously in Chapter 2.3.1, and use the UPPAAL tool to model check the formalized properties.

To verify timeliness of a transaction, we verify that the *miss_deadline* location is not reachable. For instance, the timeliness of T_i can be specified as:

$$A[\,]\, not\, Ti.miss\_deadline.$$

The absolute validity of data D_i, which requires that the age of D_i is always smaller than or equal to its specified absolute validity interval *AVI(i)*, can be specified as:

$$A[\,]\, Di.age <= AVI(i).$$

The relative validity, referring to the property that the age differences of D_i and D_j should be smaller than or equal to the specified relative validity interval *RVI(i,j)*, can be specified as the following formula:

$$A[\,]\, ((Di.updated\, imply\, Dj.age <= RVI(i,j))\\ and\, (Dj.updated\, imply\, Di.age <= RVI(i,j))).$$

Similarly, verifying a specified isolation level equals to proving that all locations representing the phenomena to be precluded are not reachable. Verifying temporal data consistency equals to checking the age of data against the property. For instance, to verify that T_i and T_j achieve SERIALIZABLE isolation, one must prove that none of the precluded phenomena Gn could occur. This is equivalent to proving that the *phenomenon_Gn* location of IsolationObserver O_n is not reachable, which can be specified as:

$$A[\,]\, not\, On.phenomenon\_Gn.$$

This contribution addresses subgoal 2, and is proposed in Paper B and C. In Paper B we propose the approach for model checking isolation and timeliness. Paper C extends the approach with observer automata for the age of data and temporal data consistency. Our framework enables flexible modeling of real-time transaction-based systems with various concurrency control algorithms, and facilitates formal verification of isolation and temporal correctness, as targeted by subgoal 2.

3.4.4 Validation on Industrial Use Cases

In order to validate the applicability and usefulness of our proposed approach, we apply them to various industrial, or industrially-relevant, use cases. This contribution addresses subgoal 3.

In Paper A we apply the proposed taxonomy to the analysis of an industrial project, the Hardware Assisted Trace (HAT) framework [46], together with its proposers from Ericsson. HAT, as shown in Figure 3.13, is a framework for

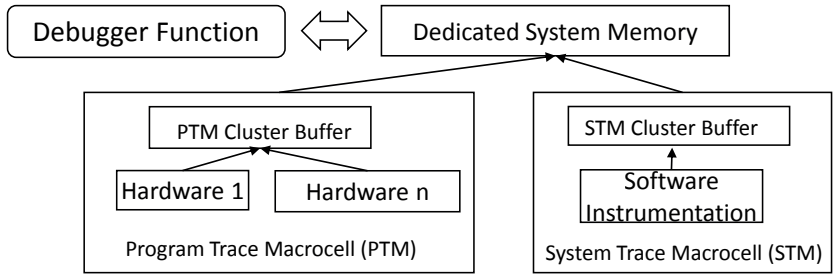

Figure 3.13: General architecture of the Hardware Assisted Trace system

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.

In Paper D we apply the taxonomy to design the monitoring subsystem for achieving auto-scaling functionality of a cloud system, with Ericsson engineers in the loop. We apply DAGGTAX to both the current OpenStack framework for auto-scaling, and a new design that extends the current framework. 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.

This contribution addresses subgoal 3. It demonstrates the applicability of DAGGTAX, and shows that DAGGTAX can help to ease the design effort and reduce software flaws prior to implementation.

3.5 Research Goals Revisited

In this section we present the contributions of this thesis by presenting the relationship between the included papers and the research subgoals. The DAGGERS process proposed in Chapter 3.4.1 addresses our overall research goal directly, by providing a systematic process as our methodology. Each research subgoal is targeted by one or two papers, as illustrated in Table 3.1. The addressed subgoals, together with the DAGGERS process, provide a solution for designing real-time data management solutions based on systematic analysis of transactional properties, and address our overall research goal.

Table 3.1: Contribution of included papers with respect to research subgoals

	Overall goal	Subgoal 1	Subgoal 2	Subgoal 3
DAGGERS	X			
Paper A		X		X
Paper B			X	
Paper C			X	
Paper D			X	

Chapter 4

Related Work

In this chapter, we discuss the related work with respect to data aggregation, as well as the development of customized transaction management for (RT)DBMS.

4.1 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. [47], Marichal [48], and Rudas et al. [5] 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. [5]. In order to design a software system that computes aggregation efficiently, Gray et al. [49] 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. [50] have extended Gray's taxonomy, and classified aggregate functions according to their state requirements, tolerance of loss, duplicate sensitivity, and monotonicity. Fasolo et al. [23] 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 [23]. Besides a classification of aggregate functions that we have discussed in the previous paragraph, Fasolo et al. [23] 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. [51], Makhloufi et al. [52], and Rajagopalan [53], with respect to different classification criteria. In contrast to the above works focusing mainly on aggregation protocols, Alzaid et al. [54] 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.

4.2 Customized Transaction Management

Designing effective and efficient transaction management to suit applications' need has been a hot topic in the development of DBMS. In this section we discuss the related work regarding the design methodologies, and the formalization and analysis of transaction models.

4.2.1 Design methodologies for developing customized transaction management

Various design methodologies have been proposed in literature for developing transaction management for DBMS. KIDS [15], an early effort to construct a 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 [18] 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, formal methods are not used to guide the design decisions.

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. [20] 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. [19] 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.

COMET [16] 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 [17], functional requirements on a DBMS are represented as features, which are composed to construct DBMS variants. Built on top of FAME-DBMS, AUTODAMA [55] 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.

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 the analysis of trade-offs between these properties, and focus on a systematic approach to analyze different decisions. By iteratively applying formal modeling and verification of the transaction models, we are able to select the appropriate transaction management mechanisms that are proved to guarantee the desired transactional properties.

4.2.2 Formalization and analysis of transaction models

In the real-time community, the common technique to formally analyze transaction performance is schedulability analysis [56]. 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.

Formal verification of transaction models plays an important role in the design of customized transaction management. Some substantial work has already been carried out to specify transaction models and reason about their properties. One group of work is represented by the ACTA framework [69] and its successors. 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 [70] extends 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 [71], 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 [72] 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, SPLACID [73] improves ACTA by providing a more expressive and structured language support for ACID variants and their sub-features, however real-time properties are not considered.

In addition to the thread of ACTA, a variety of methods have been proposed to formalize transactions. For instance, Mentis et al. [20] 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

Table 4.1: Summary of non-ACTA related work

Related work	**Properties**	**Formalism**	**Analysis Technique**
Mentis et al. [20]	ACID	state machines	model checking
Cerone et al. [57]	logical consistency models, no explicit ACID	axioms and rules	theorem proving
Bocchi et al. [58]	consistency	process calculi	model checking
Kokash et al. [59]	atomicity	Reo, constraint automata	model checking
Grov et al. [60] and Liu et al. [61]	atomicity, isolation	Real-Time Maude	model checking
Suryavanshi et al. [62]	various properties, no explicit ACID	Event-B	theorem proving
Kirchberg [63]	various properties, no explicit ACID	transition rules	model checking
Makni et al. [64]	various properties, no explicit ACID	SPIN	model checking
Gaaloul et al. [65]	atomicity	event calculus	model checking
Bourne [66]	atomicity	temporal logics	model checking
Lanotte et al. [67]	atomicity and timeliness	timed automata	model checking
Kot [68]	various properties, timeliness, no explicit ACID	timed automata	model checking

for analyzing a selected subset of properties in particular scenarios. A list of non-ACTA related works are presented in Table 4.1. 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 rage of concurrency control mechanisms by using skeletons and patterns.

Chapter 5

Conclusions and Future Work

In this thesis we propose a systematic approach for designing data management with data aggregation support for real-time applications, in which data aggregation, as well as other data accessing and manipulating computations, are modeled as transactions managed by an RTDBMS. Our approach is the DAGGERS process that systematically derives customized transaction models incorporating the desired trade offs between ACID and temporal correctness properties, and decides the appropriate transaction management mechanisms to generate the RTDBMS. The trade-off decisions are made via formal verification of the desired properties. Concretely, we have proposed the following contributions to facilitate the process:

- A taxonomy of data aggregation processes, depicted as a feature diagram, for analyzing the characteristics of data aggregation and their implications. Three design rules and a set of design heuristics are proposed based on the taxonomy to provide guidance for the design of data aggregation processes.

- A pattern-based formal approach, within the timed automata framework, which facilitates flexible modeling of transactions with various concurrency control algorithms, and model checking isolation and temporal correctness of the modeled transactions.

By the DAGGERS process the system designer can systematically analyze the data-related computations in the real-time application, and trade off the conflicting properties. An RTDBMS can then be generated with run-time mechanisms that are verified to achieve the desired properties.

The validation on two real-world industrial case studies shows that 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 approach allows designers to model transactions with various concurrency control algorithms, and formally analyze the satisfiability of isolation and temporal correctness. Based on the analysis the designer can choose a desired trade-off and select the appropriate concurrency control algorithm that has been proved to achieve the selected trade off.

5.1 Future Work

The work of this thesis opens several future research directions. One possible future work involves the integration of DAGGTAX with state-of-art architectural and process modeling frameworks, such that the feature selection of data aggregation processes can be reasoned about together with the choices of deployment and business process models.

Another future work may involve the trading-off methods for other transaction properties. In this thesis we have proposed the method of modeling transactions with concurrency control, with the aim of analyzing isolation and temporal correctness. In the future work we need to develop methods to address the analysis of atomicity, consistency and durability, together with the corresponding DBMS mechanisms.

Integration of other formal techniques may greatly enhance the applicability and scalability of DAGGERS. Since real-time data-intensive applications are often heterogeneous in the amount of data, complexity of computation, and the desired properties, different analysis techniques may need to be applied depending on the particular system or module. For instance, for a large system, Statistical Model Checking [74] could be a better choice than exhaustive model checking in terms of scalability. For quantitative analysis of data production and consumption in aggregation, dataflow models [75] could be useful. Our

future work will investigate which formal methods could be exploited to analyze different situations, and how they can be integrated into the big picture of system design.

Last but not least, a larger scale evaluation of the entire DAGGERS process on the design of an industrial system, with respect to its scalability and efficiency, is also an interesting future work.

Bibliography

[1] R. Hegde, G. Mishra, and K. Gurumurthy. *An insight into the hardware and software complexity of ecus in vehicles*, pages 99–106. Springer, 2011.

[2] A. N. Lee and J. L. M. Lastra. Data aggregation at field device level for industrial ambient monitoring using web services. In *Proceedings of the 9th IEEE International Conference on Industrial Informatics*, pages 491–496, July 2011.

[3] K. Ramamritham. Real-time databases. *Distributed and Parallel Databases*, 1(2):199–226, 1993.

[4] S. F. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Eftring. Deeds towards a distributed and active real-time database system. *ACM Sigmod Record*, 25(1):38–51, March 1996.

[5] I. J. Rudas, E. Pap, and J. Fodor. Information aggregation in intelligent systems: An application oriented approach. *Knowledge-Based Systems*, 38:3–13, 2013.

[6] G.R. Goud, N. Sharma, K. Ramamritham, and S. Malewar. Efficient real-time support for automotive applications: A case study. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 335–341, 2006.

[7] K. Bür, P. Omiyi, and Y. Yang. Wireless sensor and actuator networks: Enabling the nervous system of the active aircraft. *IEEE Communications Magazine*, 48(7):118–125, 2010.

[8] D. Mairiza, D. Zowghi, and N. Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 311–317, 2010.

[9] L. Almeida, F. Santos, T. Facchinetti, P. Pedreiras, V. Silva, and L. S. Lopes. Coordinating distributed autonomous agents with a real-time database: The cambada project. In *Proceedings of the 19th International Symposium of Computer and Information Sciences*, 2004.

[10] J. Gray and A. Reuter. *Transaction processing: concepts and techniques*. Morgan Kaufmann Publishers Inc., 1992.

[11] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, 1983.

[12] X. Song and J.W.S. Liu. How well can data temporal consistency be maintained? In *Proceedings of the 1992 IEEE Symposium on Computer-Aided Control System Design (CACSD)*, pages 275–284, 1992.

[13] J. A. Stankovic, S. H. Son, and J. Hansson. Misconceptions about real-time databases. *Computer*, 32(6):29–36, 1999.

[14] B. Gallina. *PRISMA: a software product line-oriented process for the requirements engineering of flexible transaction models*. PhD thesis, University of Luxembourg, Luxembourg, 2010.

[15] A. Geppert, S. Scherrer, and K.R. Dittrich. Kids: Construction of database management systems based on reuse. *University of Zurich, Switzerland*, 1997.

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

[17] M. Rosenmüller, N. Siegmund, H. Schirmeier, J. Sincero, S. Apel, T. Leich, O. Spinczyk, and G. Saake. Fame-dbms: tailor-made data management solutions for embedded systems. In *Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management*, pages 1–6. ACM, 2008.

[18] J. Kienzle, E. Duala-Ekoko, and S. Gélineau. Aspectoptima: A case study on aspect dependencies and interactions. In *Transactions on Aspect-Oriented Software Development V*, pages 187–234. Springer, 2009.

[19] R. T. Khachana, A. James, and R. Iqbal. Relaxation of acid properties in autra, the adaptive user-defined transaction relaxing approach. *Future Generation Computer Systems*, 27(1):58–66, 2011.

[20] A. Mentis and P. Katsaros. Model checking and code generation for transaction processing software. *Concurrency and Computation: Practice and Experience*, 24(7):711–722, 2012.

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

[22] S. F. Andler, L. Niklasson, B. Olsson, A. Persson, L. J. de Vin, B. Wangler, T. Ziemke, and T. Planstedt. Information fusion from databases, sensors and simulations: A collaborative research program. In *Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop*, pages 234–244, April 2005.

[23] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-network aggregation techniques for wireless sensor networks: a survey. *IEEE Wireless Communications*, 14(2):70–87, 2007.

[24] J. L. Azevedo, B. Cunha, and L. Almeida. Hierarchical distributed architectures for autonomous mobile robots: A case study. In *Proceedings of the 2007 IEEE Conference on Emerging Technologies and Factory Automation*, pages 973–980, Sept 2007.

[25] T. He, L. Gu, L. Luo, T. Yan, J.A. Stankovic, and S.H. Son. An overview of data aggregation architecture for real-time tracking with sensor networks. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, pages 8 pp.–, 2006.

[26] K. Ramamritham and P. K. Chrysanthis. A taxonomy of correctness criteria in database applications. *The International Journal on Very Large Data Bases*, 5(1):085–097, January 1996.

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

[28] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems*, 8(2):186–213, June 1983.

[29] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, June 2002.

[30] D. J. Abadi. Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. *Computer*, 45(2):37–42, 2012.

[31] E. B. Moss. *NESTED TRANSACTIONS: AN APPROACH TO RELIABLE DISTRIBUTED COMPUTING*. PhD thesis, Massachusetts Institute of Technology, USA, 1981.

[32] H. Garcia-Molina and K. Salem. Sagas. *ACM SIGMOD Record*, 16(3):249–259, December 1987.

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

[34] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ansi sql isolation levels. *ACM SIGMOD Record*, 24(2):1–10, May 1995.

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

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

[37] P. S. Yu, K. L. Wu, K. J. Lin, and S. H. Son. On real-time databases: concurrency control and scheduling. *Proceedings of the IEEE*, 82(1):140–157, Jan 1994.

[38] T. W. Kuo and A. K. Mok. Application semantics and concurrency control of real-time data-intensive applications. In *Proceedings of the 13th Real-Time Systems Symposium*, pages 35–45, Dec 1992.

[39] L. B. C. DiPippo and V. F. Wolfe. Object-based semantic real-time concurrency control. In *Proceedings of the 14th Real-Time Systems Symposium*, pages 87–96, 1993.

[40] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[41] K. G. Larsen, P. Pettersson, and Y. Wang. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, 1997.

[42] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2 – 34, 1993.

[43] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[44] H. J. Holz, A. Applin, B. Haberman, D. Joyce, H. Purchase, and C. Reed. Research methods in computing: What are they, and how should we teach them? *ACM SIGCSE Bulletin*, 38(4):96–114, June 2006.

[45] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, USA, 1990.

[46] C. Vitucci and A. Larsson. Hat, hardware assisted trace: Performance oriented trace & debug system. In *Proceedings of 26th International Conference on Software & Systems Engineering and their Applications*, 2015.

[47] R. Mesiar, A. Kolesárová, T. Calvo, and M. Komorníková. A review of aggregation functions. In *Fuzzy Sets and Their Extensions: Representation, Aggregation and Models*, volume 220, pages 121–144. Springer Berlin Heidelberg, 2008.

[48] J. Marichal. *Aggregation Functions for Decision Making*, pages 673–721. ISTE, 2010.

[49] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[50] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.

[51] I. Solis and K. Obraczka. In-network aggregation trade-offs for data collection in wireless sensor networks. *International Journal of Sensor Networks*, 1(3-4):200–212, 2006.

[52] R. Makhloufi, D. Guillaume, B. Grégory, and D. Gaïti. A survey and performance evaluation of decentralized aggregation schemes for autonomic management. *International Journal of Network Management*, 24(6):469–498, 2014.

[53] R. Rajagopalan and P.K. Varshney. Data-aggregation techniques in sensor networks: A survey. *IEEE Communications Surveys Tutorials*, 8(4):48–63, 2006.

[54] H. Alzaid, F. Ernest, J. Manuel, G. Nieto, and D. Park. A taxonomy of secure data aggregation in wireless sensor networks. *International Journal of Communication Networks and Distributed Systems*, 8(1-2):101–148, 2012.

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

[56] S. Han, K. Lam, J. Wang, K. Ramamritham, and A.K. Mok. On co-scheduling of update and control transactions in real-time sensing and control systems: Algorithms, analysis, and performance. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2325–2342, 2013.

[57] A. Cerone, G. Bernardi, and A. Gotsman. A Framework for Transactional Consistency Models with Atomic Visibility. In *Proceedings of the 26th International Conference on Concurrency Theory*, volume 42, pages 58–71. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[58] L. Bocchi and H. Melgratti. On the behaviour of general purpose applications on cloud storages. *Service Oriented Computing and Applications*, 9(3):213–227, 2015.

[59] N. Kokash and F. Arbab. Formal design and verification of long-running transactions with extensible coordination tools. *IEEE Transactions on Services Computing*, 6(2):186–200, April 2013.

[60] J. Grov and P. C. Ölveczky. *Increasing Consistency in Multi-site Data Stores: Megastore-CGC and Its Formal Analysis*, pages 159–174. Springer International Publishing, 2014.

[61] S. Liu, P. C. Ölveczky, M. R. Rahman, J. Ganhotra, I. Gupta, and J. Meseguer. Formal modeling and analysis of ramp transaction systems. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 1700–1707. ACM, 2016.

[62] R. Suryavanshi and D. Yadav. Modeling of multiversion concurrency control system using event-b. In *Proceedings of the 2012 Federated Conference on Computer Science and Information Systems*, pages 1397–1401, Sept 2012.

[63] M. Kirchberg. Using abstract state machines to model aries-based transaction processing. *Journal of Universal Computer Science*, pages 157–194, 2009.

[64] A. Makni, R. Bouaziz, and F. Gargouri. Formal verification of an optimistic concurrency control algorithm using spin. In *Proceedings of the 13th International Symposium on Temporal Representation and Reasoning*, pages 160–167, June 2006.

[65] W. Gaaloul, M. Rouached, C. Godart, and M. Hauswirth. *Verifying Composite Service Transactional Behavior Using Event Calculus*, pages 353–370. Springer Berlin Heidelberg, 2007.

[66] S. Bourne. *Formal Verification of Transactional and Configurable Service-Oriented Processes*. PhD thesis, UNIVERSITY OF ADELAIDE, Australia, 2016.

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

[68] M. Kot. Modeling selected real-time database concurrency control protocols in uppaal. *Innovations in Systems and Software Engineering*, 5(2):129–138, 2009.

[69] P.K. Chrysanthis and K. Ramamritham. Synthesis of extended transaction models using acta. *ACM Transactions on Database Systems*, 19(3):450–491, 1994.

[70] M. Xiong and K. Ramamritham. Specification and analysis of transactions in real-time active databases. In *Real-Time Database and Information Systems: Research Advances*, volume 420, pages 327–351. Springer, 1997.

[71] A. Geppert. *Methodical construction of database management systems*. PhD thesis, University of Zuerich, Switzerland, 1994.

[72] I. Kiringa. Simulation of advanced transaction models using golog. In *Database Programming Languages*, volume 2397, pages 318–341. Springer Berlin Heidelberg, 2002.

[73] B. Gallina and N. Guelfi. Splacid: An spl-oriented, acta-based, language for reusing (varying) acid properties. In *Proceedings of the 32nd Annual IEEE Software Engineering Workshop*, pages 115–124, 2008.

[74] A. David, K. Larsen, A. Legay, M. Mikučionis, D. Poulsen, J. van Vliet, and Z. Wang. Statistical model checking for networks of priced timed automata. In *Formal Modeling and Analysis of Timed Systems*, pages 80–96. Springer, 2011.

[75] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, May 1995.