Mälardalen University Press Dissertations
No. 125

# DATA MANAGEMENT IN COMPONENT-BASED EMBEDDED REAL-TIME SYSTEMS

Andreas Hjertström

2012


School of Innovation, Design and Engineering

ISBN 978-91-7485-064-2
ISSN 1651-4238
Printed by Mälardalen University, Västerås, Sweden

Mälardalen University Press Dissertations
No. 125

# DATA MANAGEMENT IN COMPONENT-BASED EMBEDDED REAL-TIME SYSTEMS

Andreas Hjertström

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid
Akademin för innovation, design och teknik kommer att offentligen försvaras
fredagen den 1 juni 2012, 13.00 i Gamma, Mälardalen University, Västerås.

Fakultetsopponent: Prof. Michel R.V. Chaudron, Universiteit
Leiden, Leiden Institute of Advanced Computer Science


Akademin för innovation, design och teknik

Abstract

This thesis presents new data management techniques for run-time data in component-based embedded real-time systems. These techniques enable data to be modeled, analyzed and structured to improve data management during system development, maintenance, and execution. The foundation of our work is a case-study that identifies a number of problems with current state-of-practice in data management for industrial embedded real-time systems.

We introduce two novel concepts: the data entity and the database proxy. The data entity is a design-time concept that allows designers to manage data objects throughout different design and maintenance activities. It includes data-type specification, documentation, specification of timing and quality properties, tracing of dependencies between data objects, and enables analysis and automated validation.

The database proxy is a run-time concept designed to allow the use of state-of-the-art database technologies in contemporary software-component technologies for embedded systems. Database proxies decouple components from an underlying database residing in the component framework. This allows components to remain encapsulated and reusable, while providing temporally predictable access to data maintained in a database, thus enabling the use of database technologies, which has previously excluded, in these systems.

To validate our proposed techniques, we present a tool implementation of the data entity as well as implementations of the database proxy approach, using commercial tools, the AUTOSAR standardized automotive software architecture, and automotive hardware. Our results show that the presented techniques can contribute to the development of future component-based embedded real-time systems, by providing structured and efficient data management.

ISBN 978-91-7485-064-2
ISSN 1651-4238

# Abstract

This thesis presents new data management techniques for run-time data in component-based embedded real-time systems. These techniques enable data to be modeled, analyzed and structured to improve data management during system development, maintenance, and execution. The foundation of our work is a case-study that identifies a number of problems with current state-of-practice in data management for industrial embedded real-time systems.

We introduce two novel concepts: the data entity and the database proxy. The data entity is a design-time concept that allows designers to manage data objects throughout different design and maintenance activities. It includes data-type specification, documentation, specification of timing and quality properties, tracing of dependencies between data objects, and enables analysis and automated validation.

The database proxy is a run-time concept designed to allow the use of state-of-the-art database technologies in contemporary software-component technologies for embedded systems. Database proxies decouple components from an underlying database residing in the component framework. This allows components to remain encapsulated and reusable, while providing temporally predictable access to data maintained in a database, thus enabling the use of database technologies, which has previously excluded, in these systems.

To validate our proposed techniques, we present a tool implementation of the data entity as well as implementations of the database proxy approach, using commercial tools, the AUTOSAR standardized automotive software architecture, and automotive hardware. Our results show that the presented techniques can contribute to the development of future component-based embedded real-time systems, by providing structured and efficient data management.

# Swedish Summary - Svensk Sammanfattning

Inbyggda realtidssystem blir allt vanligare i de produkter och tjänster vi använder. Utvecklingstakten går allt fortare och programvaran blir allt mer komplex. Inbyggda system finns idag i t.ex. mobiltelefoner, bilar, flygplan och robotar, där programvaran kan utgöras av flera miljoner rader kod och tusentals dataelement som är distribuerade över ett stort antal datorer ihopkopplade i nätverk. Utveckling och underhåll av dessa komplexa system medför en allt högre kostnad. För att utveckla elektroniksystemet är kostnaden, i en modern, avancerad bil idag, omkring 40% av den totala utvecklingskostnaden. Inom fordonsindustrin drivs denna utveckling av framför allt hårdare miljökrav, nya funktioner samt krav på bättre aktiv och passiv säkerhet.

För att hantera utvecklingen av dessa system försöker man göra informationen om systemet mer överblickbar genom att gruppera funktioner i olika komponenter som kan kommunicera genom ett förutbestämt gränssnitt. Denna teknik kallas för komponentbaserad utveckling. Komponentbaserade tekniker som används idag fokuserar främst på att hantera funktioner, och saknar bra metoder för att hantera den stora mängd data som utväxlas mellan komponenterna. Nya metoder för att effektivt hantera data har stor potential att göra både utvecklingen och exekveringen av inbyggda system enklare och mer kostnadseffektiv.

Denna avhandling introducerar nya koncept för hantering av data under utveckling, underhåll och exekvering av inbyggda komponentbaserade realtidssystem. Resultaten i denna avhandling baserar sig på en fallstudie som visar på stora problem med att hantera data inom industrin. Dessa resultat visar tydligt att hanteringen av data måste prioriteras mer och ingå som en integrerad del av utvecklingen av hela systemets arkitektur.

För hantering av data under utvecklings- och underhållsfaserna introducerar vi konceptet *data entity*. En *data entity* möjliggör för utvecklare att modellera och dokumentera varje dataelement i systemet korrekt redan i ett tidigt skede av utvecklingsfasen. Därutöver är det också viktigt att på ett enkelt sätt kunna skapa dokumentation och bedöma egenskaper, samt att visualisera dataflöden och beroenden mellan data för att öka den totala kunskapen om systemet. Tekniker för att hantera stora och komplexa datamängder i ett inbyggt system finns tillgängliga i form av databaser. Problemet är att de komponentbaserade teknikerna och databaserna är fundamentalt olika. Här finns ett tydligt glapp, vilket vi försöker överbrygga i denna avhandling. För hantering av data under exekvering introducerar vi konceptet *database proxy*, som möjliggör användandet av en databas utan att bryta mot grundläggande principer inom komponentbaserad utveckling. Syftet med detta är att komplettera den bristande datahanteringen inom komponentbaserad utveckling genom att utnyttja de beprövade tekniker som finns tillgängliga i en databas. Avhandlingen innefattar även ett antal implementationer av verktyg samt evalueringar av de ingående koncepten för hantering av data. Embedded Data Commander (EDC) innehåller en samling verktyg för att integrera och hantera "data entities" i en komponentmodell. Vidare har verktyg för konfigurering samt generering av "database proxies" i komponentmodellen SAVE har implementerats och evaluerats. Slutligen så har "database proxies" implementerats och evaluerats på hårdvara i ett AUTOSAR kontext.

To my Beloved Family

# Acknowledgements

This thesis marks the end of a great journey and at the same time the beginning of something new. To say that this has been an entirely smooth ride would be to lie to myself and others. There have been ups and downs, although the up side has by far exceeded the downside.

Two people have been by my side this entire journey, my supervisors Dr. Dag Nyström and Prof. Mikael Sjödin. Thanks for your excellent support and guidance, both in your role as my supervisors as well as "offline"! Dag, it has been a privilege to work with you. You have always been there for me (probably more than required) and when problems have arisen, your vision and never ending stream of new input and positive thinking carried me forward. Mikael, your ability to concretize and guide me to bring out the essence of my research and paper writing, is amazing.

A special thanks to my friend Peter Wallin. If you would not have started your PhD studies and so warmly recommended it, I would probably have missed this great opportunity. An additional thanks to Mimer Information Technology AB and ArcCore AB for the cooperation and input to the project.

I would also like to thank Jörgen Lidholm for the good discussions and being a great friend. Many people at the department have made this journey more enjoyable, thanks to Fredrik Ekstrand, Karl Ingström, Lars Asplund, Mikael Ekström, Kaj Hänninen, Stefan Cedergren, and all the other wonderful people. In addition, a special thanks to all the administrative people that have helped me with traveling arrangements, paper work, and being great companions.

To the whole Progress gang, Hans Hanson, Tomas Nolte, Ivica Crnkovic, Paul Pettersson, Hüseyin Aysan, Farhang Nemati, Moris Behnam, Mikael Åsberg, Severine Sentilles, Johan Kraft, Yue Lu, Stefan Bygde, Jan Carlsson, Aneta Vulgarakis and all others who have been great traveling companions, friends, and that have provided a lot of input to my work.

Most important, I thank my loving family, Anna, my son Felix, and my daughter Livia for supporting me and making my life wonderful. I love you. You are my everything!

*This work has been supported by the Swedish Foundation for Strategic Research within the PROGRESS Centre for Predictable Embedded Software Systems.*

Andreas Hjertström
Västerås, June, 2012

# List of Publications

## Papers Included in the Thesis

**Paper A:** Design-Time Management of Run-Time Data in Industrial Embedded Real-Time Systems Development, Andreas Hjertström, Dag Nyström, Mikael Nolin and Rikard Land, *13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Hamburg, Germany, September, 2008*

**Paper B:** A Data-Entity Approach for Component-Based Real-Time Embedded Systems Development, Andreas Hjertström, Dag Nyström and Mikael Sjödin, *14th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Palma de Mallorca, Spain, September, 2009*

**Paper C:** Data Management for Component-Based Embedded Real-Time Systems: the Database Proxy Approach, Andreas Hjertström, Dag Nyström and Mikael Sjödin, *Journal of Systems and Software, vol 85, nr 4, p821-834, Elsevier, April, 2012*

**Paper D:** Introducing Database-Centric Support in AUTOSAR, Andreas Hjertström, Dag Nyström and Mikael Sjödin, *7th IEEE International Symposium on Industrial Embedded Systems (SIES), Karlsruhe, Germany, June, 2012*

**Paper E:** Data Management in AUTOSAR: a Tool Suite Extension Approach, Andreas Hjertström, Dag Nyström and Mikael Sjödin, *MRTC Report, submitted for conference publication*

# Additional Papers by the Author

INCENSE: Information-Centric Run-Time Support for Component-Based Embedded Real-Time Systems, Andreas Hjertström, Dag Nyström, Mikael Åkerholm and Mikael Nolin, *Proceedings of the Work-In-Progress (WIP) Session, 14th IEEE Real-Time and Embedded Technology and Applications Symposium, p 4, Seattle, United States, April, 2007*

Information Centric Development of Component-Based Embedded Real-Time Systems, Andreas Hjertström, *Licentiate Thesis, Mälardalen University Press, December, 2009*

Database Proxies for Component-Based Real-Time Systems, Andreas Hjertström, Dag Nyström, Mikael Sjödin, *22nd Euromicro Conference on Real-Time Systems, p 79 - 89, Brussels, Belgium, July, 2010*

Database Proxies: A Data Management Approach for Component-Based Real-Time Systems, Andreas Hjertström, Dag Nyström and Mikael Sjödin *MRTC, technical report*

# Contents

# I

# Thesis

# Chapter 1

# Introduction

The evolution of embedded systems affects us all. Embedded systems are nowadays included in almost everything that surrounds us in our daily life. This has mostly to do with our increased demand for new functionalities that cannot be built, or are not practical to build, using traditional mechanics. Mobile phones, medical equipment, kitchen appliances, home entertainment systems, cars, and cameras are examples of such systems. Some of these are highly complex, and huge amounts of software are used to realize the different functionalities. In addition, the current trend is that systems are evolving from closed stand-alone devices to highly dynamic systems interconnected with the surrounding environment.

Developing these kinds of systems is a challenging task. Today, 90% of the innovations in a premium car are related to electronics and software. In addition, as many as 2500 software functions, sometimes dependent on each other, are distributed throughout a highly interconnected architecture with up to 80 Electrical Control Units (ECU) [1]. Furthermore, there is a more frequent mix of critical functionalities, such as breaking assistance, and non-critical functionalities, such as an infotainment system. Developing these kinds of systems is often associated with high cost [2].

From this evolution towards more complex and interconnected systems arises the need for more efficient means to manage data, perform diagnostics, and to provide predictable and dynamic data access. However, this thesis shows that the current state-of-practice of data management is not sufficient to cope with tomorrow's embedded systems. Therefore, the development of new techniques that deal with/can control data management are needed.

*This thesis contributes to the future of embedded-systems development by identifying problem areas in the current state-of-practice, and by introducing new techniques for the development of component-based embedded real-time systems. These techniques comprise a holistic approach to data management by providing design-time support for modeling, management, documentation and analysis of run-time data, as well as run-time mechanisms for extracting, structuring, and the secure sharing of data.*

**Design-time data management:** A case-study, presented in this thesis, shows that developers are not provided with adequate techniques that enables efficient and up-to-date management of documentation of run-time data. The growing information volume, lack of tool support, poor routines, and the sometimes inadequate documentation, especially concerning internal data in nodes, are becoming an increasing problem. This has for example led to (i) obsolete documentation, (ii) redundant and stale data (data that is not removed due to unknown dependencies), and (iii) companies becoming highly dependent on the undocumented knowledge of individual developers.

The *data-entity approach* is presented in this thesis as a solution to facilitate efficient design-time management of run-time data. This approach has been evaluated and implemented into a tool-suite.

**Run-time data management:** In the development of functions, elevating the abstraction level and providing efficient tool support, is commonly used approaches to manage complexity. One such approach, which is increasingly used by industry today to raise productivity and reduce complexity, is Component-Based Software Engineering (CBSE). Structured development, standardized architectures, and reuse are mentioned as key factors for success. The CBSE focus lies on specifying and developing a component or a set of reusable components with certain functionality. However, CBSE does, so far, not provide structured support for managing data. This has in turn led to highly uncoordinated and ad-hoc management of data in many complex distributed systems [1, 2, 3].

Instead of reinventing data management techniques or developing ad-hoc solutions using internal data structures, the use of existing techniques should be promoted.

Outside the embedded community, a well-proven data management technique that offers standardized interfaces, efficient data management, dynamic access to data, user access control, and effective tool support is available, namely DataBase Management Systems (DBMS). Over the last few years, the use of DBMSs in embedded systems has increased. For example, many control-systems, and virtually all premium mobile phones, such as Apples iPhone and Android-based phones contain databases. However, the use of Real-Time DataBase Management Systems (RTDBMS) within industrial embedded real-time systems is still quite limited, even though there are a few commercial RTDBMSs available [4, 5]. Moreover, this is especially true for component-based systems.

Although both CBSE and RTDBMSs aim to reduce complexity, the coexistence between the techniques is non-trivial since their design goals are fundamentally different (i) within CBSE, decoupling of components from the context in which they are deployed is vital, whereas an RTDBMS provides a blackboard architecture that requires specific database knowledge to be embedded within components in order to access data, (ii) direct access to shared data introduces hidden dependencies between components, thereby violating a fundamental aim of CBSE.

The combined approach, to not only manage the functional complexity of the application and specifying components, but to also utilize the available tools and techniques offered by an RTDBMS, is a research area that is **not well established**.
*Database proxies* are presented in this thesis as a technique to close the gap between CBSE and RTDBMSs. Furthermore, database proxies have been implemented and evaluated as an approach to manage run-time data in the automotive initiative, AUTomotive Open System ARchitecture (AUTOSAR).

The research results presented in this thesis are applicable to many industrial application areas which depend on the efficient development of complex embedded real-time systems with a mix of critical and less critical functions. However, the focus of this thesis is automotive systems from which we borrow the technical background and terminology and apply our results to.

## 1.1 Problem Description

The continuous increase of complexity and new requirements on data management enhances the challenges with respect to performing design-time and run-time data management in a predictable, efficient and structured manner. Developers need new tools and techniques to aid them with the problems of today and tomorrow.

In an effort to understand the problematics concerning data management, this thesis investigates (i) the current problems within industrial embedded systems development, (ii) what tools and techniques could facilitate the development, as well as how and in what contexts they could be deployed.
To be precise, the thesis focuses on the following:

**F1** How is data currently managed in the industry and what are the main problems concerning design-time and run-time data management?

**F2** How can we support design-time data management within CBSE?

**F3** How can we support run-time data management within CBSE by utilizing state-of-the-art RTDBMS technology?

**F4** How can real-time data management techniques be integrated into an industrial development setting?

## 1.2 Thesis Outline

This thesis consists of two main parts. The first part presents an introduction, problem description and background to the scientific work carried out. The second part comprises a collection of published papers, papers A-E.
The remainder of the thesis is structured as follows:

**Chapter 2** presents the background to the research including the techniques and tools that have been used.

**Chapter 3** presents the main technical contributions, the research methodology, the research process, the problem definition, and a summary of the contributions. In addition, a summary of the included papers and my contribution to the results are presented.

**Chapter 4** complements Chapter 2 in that we describe the relevant state-of-the-art, which is related to the work carried out in this thesis.

**Chapter 5** concludes the introductory part of thesis and discusses possible contingency directions.

**Chapter 6-10** correspond to the papers that form the basis of this thesis.

# Chapter 2

# Background and Utilized Techniques

This chapter presents technical information about relevant areas within the scope of this thesis, such as embedded systems, real-time systems, component-based software engineering, and real-time database management systems. In addition, this chapter presents the major tools and techniques that have been used within the scope of this work, e.g. Save CCT, ProCom, AUTOSAR and Arctic Core, COMET, and Mimer SQL Real-Time.

## 2.1 Embedded Systems

An embedded system is a computer system, typically custom-made to perform a certain task or small set of tasks by interacting through sensors and actuators. Nowadays, these embedded systems can be found almost everywhere. They are used in watches, vehicles, robots, airplanes, and even toothbrushes. Their purpose is most often to reduce the number of mechanical parts by replacing them with electronics, in order to add functionality and/or to save costs. Most of these systems that we encounter in our everyday life are static, i.e. the software is never modified. However, there is an increase of devices that are more dynamic and where software can be continuously updated or replaced. An embedded system is characterized by limited hardware resources such as memory size and processor performance. Traditionally, embedded systems are either insolated devices or a part of a larger interconnected system. The current

trend driven by new demands on functionality and features is to change embedded systems from being stand-alone systems to being interconnected with other systems. An example of such a system of systems is Car-to-Car (C2C) communication [6], which allow cars to interact with each other to share information about, for example, a possible nearby hazard or to access the internet for infotainment services. This entails new requirements on how data is accessed and shared. The important aspects include flexibility, dependability and security.

## 2.2 Embedded Real-Time Systems

An embedded real-time system has additional requirements, compared to more general embedded systems, namely not only to perform its task correctly, but also to perform it predictably and within a predefined time interval: not too soon and not too late. Real-time systems are not only about performing a task as fast as possible. I general, real-time embedded systems interact with the environment where external events are perceived by sensors. These events are then analyzed and actuated upon, based on the result of the analysis. A typical example of a real-time system in a vehicle is an air-bag which has to be inflated within a certain time frame if activated by a collision. If the inflation is triggered too soon or too late the air-bag could cause the passengers even more harm than a complete lack of inflation.

Traditionally, real-time systems are divided into two main classes: hard and soft real-time systems. A *hard real-time system* should perform its actions before a defined deadline. A failure in meeting the deadline can have catastrophic consequences if the system is safety-critical. An air-bag is a typical example of such a system.

A *soft real-time system* usually manages less critical applications where a missed deadline can have a negative, but tolerable, effect on the performance of the system. Examples of such systems may concern the display of statistical information, the control of power windows, or to perform diagnostics.

In many applications, combinations of both hard and soft real-time tasks are used.

## 2.3 System Modeling and Development

Developing any type of complex software is most often a difficult and time consuming task. Nowadays, a common solution to manage this problem, is to develop tools and techniques to raise the level of abstraction, build models,

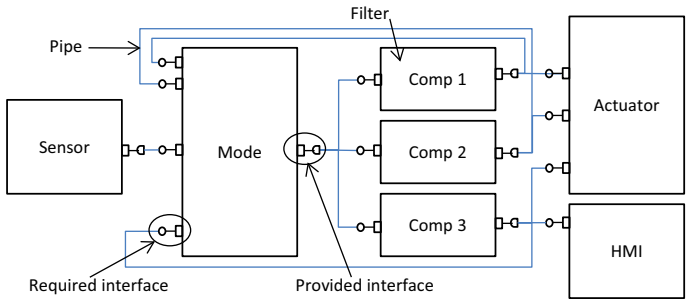


Figure 2.1: CBSE architectural example

generate code, and to reuse as much as possible. A frequently used technique within automotive software development is Component-Based Software Engineering (CBSE).

### 2.3.1 Component-Based Software Engineering

Component-Based Software Engineering aims to achieve a high level of abstraction when designing systems by dividing systems into well-defined and encapsulated building blocks called components. These components have well-defined communication interfaces that make them reusable entities that can be assembled to form entire systems. CBSE introduces a possibility to maintain and improve systems by replacing individual components. In this way, a significant amount of development effort and costs can be saved [7].

Figure 2.1 shows an example of a pipe-and-filter [8] component model where data is passed between components (filters) using connections (pipes). The entry point for the connection to the components is the interface (port). No communication outside of its interface is allowed.

A component can have two types of interfaces: required and provided. The required interface specifies what is needed as input to be able to process (filter) the data and output the result to the provided interface. Furthermore, a component can be either a white-box or a black-box component. A white-box component reveals its internal composition. This can enable developers to directly change the source code if needed. However, a changed behavior of the component, i.e., new versions emerges, can make it difficult to propagate, for instance, bug fixes. A black-box component is typically already compiled and does not reveal any internal details.

There is a great variety of component models which are suitable for different types of systems. COM [9], EJB [10] and .NET [11] are typically used for PC applications since they are not sufficiently taking important embedded systems requirements into account, such as timing properties, safety-criticality and the limited amount of resources available. Examples of component models aimed to at satisfying the requirements of embedded systems are the Rubus component model [12], SaveCCT [13], Koala [14], ProCom [15] and AUTOSAR [16].

In the following sections we describe component technologies which are used in papers B and paper C, namely SaveCCT, ProCom, and AUTOSAR.

### 2.3.2 SaveCCT

The SaveComp Component Technology (SaveCCT) [13] is focused on embedded control software for vehicle systems, with the aim to be predictable and analyzable. The applications are built by connecting input and output ports of components by using their interfaces (see Figure 2.2). Components are then executed using a trigger-based strict "read-execute-write" semantics.

A component is always inactive until triggered. Once triggered it starts to execute by reading data from its input ports to perform the computations. Data is then written to its output ports and outgoing triggering ports are activated. This allows the execution of a component to be functionally independent of any concurrent activity, once it has been triggered. SaveCCT also supports composite components. A composite component is a collection of components that are encapsulated into a single component with the same type of interface and behavior as a primitive component.

Figure 2.2 illustrates an example of a SaveCCT graphical representation of a component. There are two inports into the Engine Controller component, one data port and one trigger port. Data is read by the oilTempIO component from the oilTempSensor inport which is triggered with a frequency of 50Hz. Computations are done and results propagated onto the output port. In this case the output port is a combined trigger and output port.

SaveCCT supports manual design, integrated analysis tools, and automated activities such as task and code generation which transforms the component model into the execution model. In addition, an Integrated Development Environment (IDE) tool is provided, from which developers can develop components and graphically design the system. A number of tools are also available in the IDE for the automated formal analysis of components and architectures.

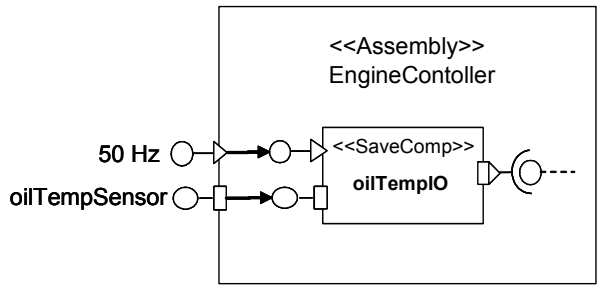


Figure 2.2: Save graphical application design

In the SaveIDE, component development as well as architectural and system modeling, is performed manually while system synthesis, glue-code generation and task allocation are fully automated. Resource usage and timing are resolved statically during the synthesis.

### 2.3.3 ProCom

The ProCom component model [15] extends SaveCCT by addressing key concerns in the development of control-intensive distributed embedded systems. ProCom provides a two-layer component model and distinguishes between a component model used for modeling independent distributed components with complex functionality (called ProSys) and a component model used for modeling smaller parts of control functionality (called ProSave).

In ProSys, a system is modeled as a collection of concurrent, communicating subsystems. Distribution is modeled explicitly, meaning that the physical location of each subsystem is not visible in the model. ProSys is a hierarchical component model where composite subsystems can be built out of other subsystems. This hierarchy ends with the so-called primitive subsystems, which are either subsystems coming from the ProSave layer or non-decomposable units of implementation (such as Commercial-Off-The-Shelf (COTS) or legacy subsystems) with wrappers to enable compositions with other subsystems.

A subsystem is specified by typed input and output message ports, expressing what type of messages the subsystem receives and sends. Message ports are connected through message channels. An example of this is illustrated in Figure 2.3, where a message channel is connected to three subsystems. A message channel is an explicit design entity representing a piece of information


Figure 2.3: ProSys Component Model

that is of interest to one or more subsystems. The message channels make it possible to express that a particular piece of shared data will be required in the system, before any producer or receiver of this data has been defined. This will in addition allow information to remain in the design even if, for example, the producer is replaced by another subsystem.

### 2.3.4 AUTOSAR

The Automotive Open System Architecture (AUTOSAR) [16] is a consortium, where several of the main Original Equipment Manufacturers (OEM), suppliers and software developers within the automotive domain, are members. AUTOSAR defines a standard component model and middleware platform for the automotive electronic architecture. One of the fundamental characteristics of AUTOSAR is the layered architecture that separates the underlying infrastructure from the applications which consist of interconnected software components. Among other things, these abstraction layers enable hardware to be replaced without the need for software updates.

The strategy is to achieve a competitive market for vendors where an OEM can use components and whole applications from "any" supplier. The idea is also that as much as possible can and should be reused to save cost and to reduce time-to-market.

AUTOSAR employs the CBSE approach, where software is encapsulated as components which communicate via well-defined interfaces. The communication between components is managed by a Virtual Function Bus (VFB), which acts as a virtual abstraction of the underlying hardware. This enables early component integration in the development process as they are independent of the ECU hardware. The realization of the VFB when configuring the final target system is the Run-Time Environment (RTE). The RTE represents the concrete implementation of the VFB. The RTE acts as a communication center for both internal Electronic Control Unit (ECU) communication and information exchange between ECUs in the system.

### 2.3.5 ArcCore

Arccore AB [17] is a provider of the open-source Arctic Core embedded AUTOSAR platform developed in Eclipse [18]. The open-source solution, to be used for education and testing, includes Arctic Core and Arctic Studio which is an Integrated Development Environment (IDE). The commercial solution offers a number of licensed professional graphical tools to facilitate development of a complete AUTOSAR system. Arctic Core includes build scripts and services such as, network communication, memory, and operating system. In addition, drivers for different microcontroller architectures are also provided.

Components and their port-based interfaces are developed using the SoftWare Component Builder tool. The Extract Builder tool is used to add selected components to the ECU, connect ports and to validate the extract. The RunTime Environment Builder models the VFB and generates a run-time implementation of the component communication. The configuration of the target platform is done in the Basic Software Builder tool which also generates the configuration files. Since Arctic Core is provided as open source, it is possible to extend it to also include additional functionalities.

## 2.4 Data Management

Data management is defined by the Data Management Association (DAMA) as:

> "the development, execution and supervision of plans, policies, programs and practices that control, protect, deliver and enhance the value of data and information assets" [19].

All computer systems involve the usage of data in some way. As both the amount of data and its use increase in an area, an increase of complexity is often unavoidable. Routines for the documentation, storage, retrieval and security of data thus become particularly important.

In this thesis we distinguish between two types of data management: design-time data management and run-time data management. This can be exemplified by an embedded system, where design-time data management refers to how run-time data is organized and documented during the design and development phase. Run-time data management refers to how data is organized and accessed in memory during execution of the system.

### 2.4.1 Design-Time Data Management

Design-time data management is the interactive link between a designer and the underlying data management system. Management of data at design-time has been and still is a fundamental part for managing the complexity of large scale and data intensive systems in order to decrease time-to-market, costs, and to increase the quality of the system. A key factor is having up-to-date and correct information about data residing in the system available during the whole development cycle. Proper documentation and structure allow for easy access to information, such as properties that can specify unique naming, type, size and where the data is used. In addition, this usually includes version handling of all design information and providing support for multiple user interactions.

The number of dedicated design-time tools for managing data in embedded real-time systems is quite limited. Most tools focus on the properties of individual data elements and how to create or define new data types. They do to a limited extent present an overview of detailed information on how and where data is used in the system during development [20, 21].

### 2.4.2 Run-Time Data Management

Run-time data management concerns how data is managed during execution of the system. Traditionally, most embedded systems developers handle data ad hoc and/or reinvent new solutions in an effort to meet the requirements of the system. This is often done using internal data-structures. Many of today's systems are developed in a distributed manner, which in turn could lead to many different solutions and strategies residing in the same system. A result of this is that large complex systems become less flexible, difficult, and demanding to maintain and extend.

Outside the embedded community, powerful tools and techniques are well established and have facilitated data management in complex data-intensive systems, such as financial markets, where they have been used for decades.

Similar as the techniques used for modeling a system or for the development of functions with a component-based approach to accomplish a higher level of abstraction, techniques to achieve a more dynamic, structured, and maintainable data management is available [22].

Database Management Systems (DBMS) are used to organize large amounts of data. Figure 2.4 shows a high level picture of a DBMS system. To put it simply, a DBMS is an interface and abstraction layer that manages access to

the physical data stored in memory. A typical application area has so far been large enterprise systems such as libraries, commercial web-sites and financial markets. Examples of enterprise mainstream DBMS are Oracle [23], Microsoft Access [24] and MySQL [25].

One of the main benefits with a DBMS is the ability to access data using a standard language. The Standard Query Language (SQL) [26] is the most common database access language, which in addition is supported by many high-level tools, for uniform data access. In order to access data or manipulate data in the database, a number of operations such as, SELECT, INSERT, and DELETE are used. One or several operations that is executed, as a single logical block of work in the database, is called a transaction. A transaction is either performed completely by ending its block of operations with a COMMIT. If the transaction is aborted before the COMMIT, a rollback to its original state is performed. A successful COMMIT makes the changes permanently stored in the database and must take the database from one consistent state to another.

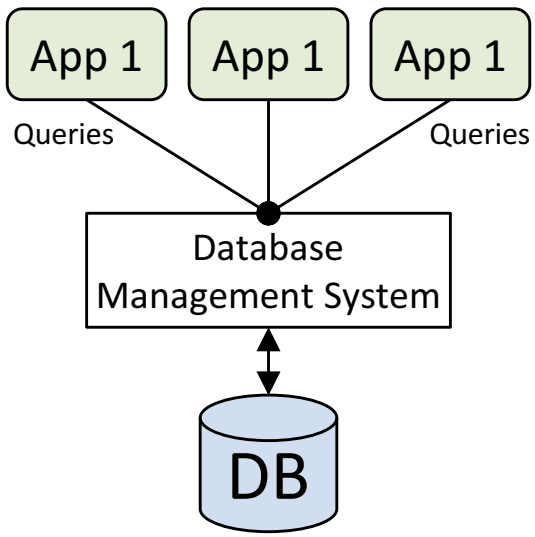


Figure 2.4: DBMS overview

To ensure a correct behavior and safe sharing of data, it is often required that a database transaction should conform to the ACID properties [27]:

- **A**tomicity: either all information in a database transaction is updated or none at all.
- **C**onsistency: after a transaction is completed the database will be in a valid state. If not, the transaction must be rolled back.
- **I**solation: changes that are made to the database will not be revealed to other users until the transaction is committed.
- **D**urability: any change to the database is permanent. The result of a committed transaction cannot be reverted.

Most DBMSs use concurrency control in order to enforce the ACID properties while handling concurrent operations, in order to avoid transaction conflicts to achieve logical correctness. The most commonly used algorithm is Two-Phase-Locking (2PL) [28] and optimistic concurrency-control [29].

The increasing amount of data and growing data complexity have increased the need for a DBMS also in embedded systems. There are now several commercial embedded DBMSs available that are suited for the specific needs, such as a small footprint, of embedded systems [4, 5, 30].

### 2.4.3 Real-Time Database Management Systems

Embedded real-time systems have different requirements compared to large enterprise systems. CPU usage, footprint and availability are highly important. Embedded Real-Time DataBase Management Systems (RTDBMSs) is developed to support real-time constraints in order to provide a deterministic timing behavior management of data in complex embedded real-time systems. For safety-critical embedded real-time systems, predictable access to data is one of the most important requirements [31].

Compared to the concurrency control algorithms used in a general-purpose DBMS, most RTDBMSs relax the semantics of the ACID properties in order to fulfill the real-time properties. This is sometimes necessary in order to comply with domain-specific requirements [32].

A commonly used concurrency control algorithm that enforce real-time properties is the Two-Phase-Locking, with High Priority abort (2PL-HP) [33] which favors transactions with high priorities, thus aborting lower prioritized transactions, in case of a conflict.

### 2.4.4 COMET RTDBMS

The COMponent-based Embedded real-Time database system [34] (COMET RTDBMS) is the result of a research cooperation between Linköping and Mälardalen University. The focus was on real-time systems in general and vehicle systems in particular. COMET is a real-time database management system is intended to be used as a tightly integrated part of the control-system, providing new techniques and functionalities such as, providing applications with support for a mix of hard and soft real-time requirements.

COMET implements the database pointer interface [35] which is a hard real-time database access-method which uses an application pointer variable to access individual data in an RTDBMS. A key property of the database-pointer concept is that reads and writes through database-pointers have deterministic execution-time with bounded and negligible blocking [36]. They also allow SQL-based soft real-time database transactions to be executed in the background without any predictability loss due to any concurrent database-pointer accesses (i.e. no starvation, conflicts, or restarts of transaction can be caused by database pointers [35]).

To guarantee hard real-time predictability for database accesses while eliminating starvation issues for soft real-time SQL queries, COMET uses the 2V-DBP concurrency-control algorithm [36] that combines versioning and pessimistic concurrency control. 2V-DBP is suited for resource-constrained, safety-critical, real-time systems that have a mix of hard real-time control applications and soft real-time management, maintenance, or user-interface applications.

Some of the technologies developed for COMET, including the database pointer concept, has later been adopted by the commercially available real-time database system Mimer SQL Real-Time Edition [4].

### 2.4.5 Mimer SQL Real-Time

Mimer SQL Real-Time (Mimer RT) [4] is a commercial RTDBMS intended for applications such as vehicle systems, process automation and telecommunication systems. Mimer RT supports applications with both hard real-time and non-real-time requirements to safely share data without putting real-time predictability at risk. Hard real-time applications utilize the RTAPI interface to access data using database pointers while non-real-time applications use standard SQL interfaces. Mimer RT combines the standard client/server architecture for SQL queries with an embedded library architecture for real-time access. The client/server architecture allows standard interfaces and tools to be used to access data both locally and remotely.

# Chapter 3

# Research Summary

This thesis presents a number of scientific contributions to facilitate design-time and run-time data management within the area of component-based embedded real-time systems. This chapter presents the technical contributions, presents the research methodology and research process, restates the problem definition, outlines the thesis contributions, and gives a résumé of the included papers.

## 3.1 Technical Contributions

A **Data entity** is a design entity that encapsulates metadata into a compilation of knowledge for run-time data items in the system.

Developers are provided with an additional architectural view, the data architectural view, which complements the traditional component-based design approach. The approach enables run-time data to be acknowledged as design objects during development, as each data item is tightly coupled with proper documentation and where properties such as usage, validity and dependency can be modeled. This enables developers to have an increased knowledge and understanding of the system.

Furthermore, as data entities are defined completely separate from the development of components and functions, data entities persist in the system regardless of any component, function or design changes. Figure 3.1 shows the metadata that is associated with a data entity.

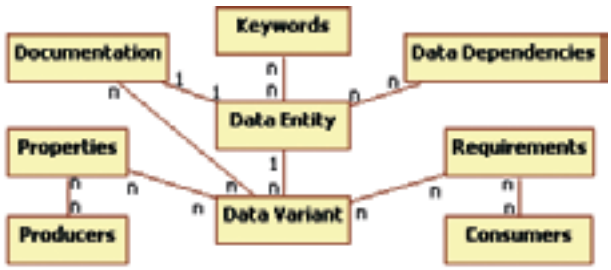


Figure 3.1: Data entity description

Figure 3.2 illustrates how our approach (right-hand-side) complements the traditional component-based design approach represented by dotted lines on the (left-hand side). The component-based approach includes tools for setting up the system architecture, developing components, and to perform analysis. The central database in the middle of the figure acts as the communicating link between the two approaches.

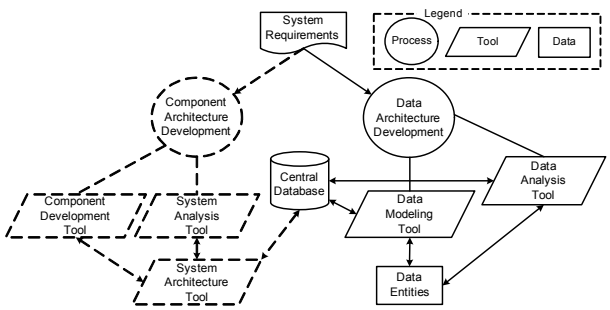


Figure 3.2: The data entity approach

We have developed a tool suite named, the Embedded Data Commander, that provide support for modeling of data entities to keep track of system data, present accurate documentation, and a data analysis tool to perform early analysis on data items. The data entity approach and tool suite serves a direct remedy to some of the problems identified in Paper A where one of the investigated systems suffered from as much as a 15% overhead because of unused and

stale data was being produced. This was due to unknown dependency issues where hardly anything could be removed due to a lack of knowledge.

The goal is to achieve higher software quality, lower development costs, and to provide higher degree of control over the software evolution process.

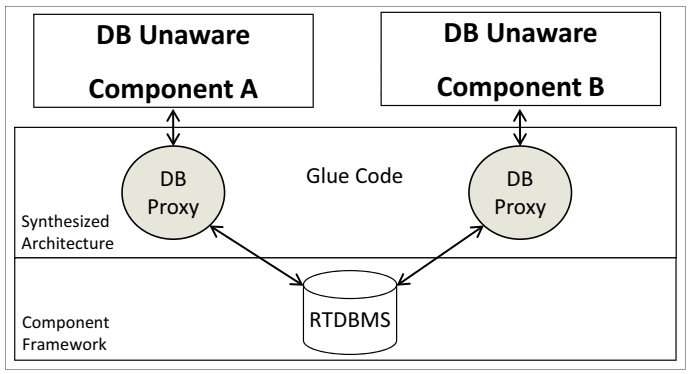


Figure 3.3: Database proxies connecting components to an RTDBMS

The **Database proxy** concept enables a successful integration of an RTDBMS into a component-based system. As illustrated in Figure 3.3, a database proxy is part of the component framework, thus external to the component. The task of the database proxy is to enable for components to interact with an RTDBMS using their normal interfaces. The database proxy is placed between the component and the RTDBMS and includes pieces of code that translates data from a components port to a database call and further on to an RTDBMS residing in the component framework and vice versa. These pieces of code are neither a part of the component nor a part of the RTDBMS; instead database proxies are automatically generated glue-code synthesized from the system architecture.

*Hard proxies* use state-of-the-art database pointers provide predictable access to individual data elements, and *soft proxies* use an SQL interface to provide flexible access to data. A hard real-time database-pointer provides direct access to a data element in memory without calling the database server. In addition, that a hard proxy only translates native data types such as an integer, character or float, implies that no unpredictable type conversions or translation of complex data types that require unbounded iterations are allowed.

```
/*** Original code example ***/
void function(){
DisableAllInterrupts();
Read_Value_Port_1(&Port_1_data_1->value);
EnableAllInterrupts();
}
/*** Database proxy code ***/
void function_DBProxy(){
MimerRTGetInteger(DBP_Actuator, &Port_1_data_1->value);
}
```

Figure 3.4: Differences between regular code and database proxy code

Figure 3.4 illustrated the code differences, using c-code, between an implementation not using, and using hard database proxies. In the original code example, all interrupts are disabled before the call to read the value is made. After the value has been read, interrupts are enabled. When using a database proxy to read the value from the database using a database pointer, the difference to the original code, is that the interrupt disable is not needed within the database proxy, since this is managed by the database.

A soft proxy is typically used for graphical interface components, logging components, and diagnostics components. Therefore, soft proxies emphasize support for more complex data structures by using an SQL interface, towards the RTDBMS.

## 3.2 Research Process

The methodology that has permeated all of the research presented in this thesis is based on the *technology transfer model* presented by Gorschek et al. [37]; see Figure 3.5. However, since this thesis is not a fully integrated industrial project, steps 5 and 7 have not been included and are left for future work. In addition, we have used research approaches: techniques and descriptive models, as well as the validation techniques: implementation, evaluation, and experience techniques described by Shaw [38].

Our research has been guided by the following process (see Fig 3.6), where each item corresponds to specific elements of the *technology transfer model*:

- **Identifying Problems:** A literature study of the state-of-the-art and a case-study conducted at five industrial companies identified that the current status within data management in component-based embedded real-

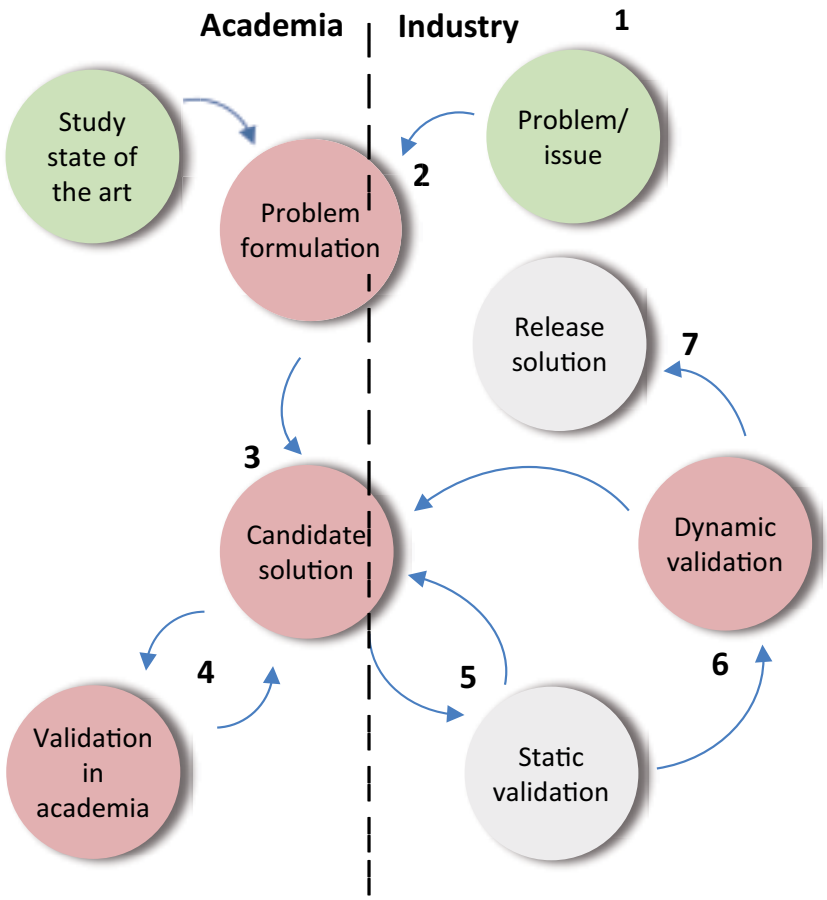


Figure 3.5: The technology transfer model

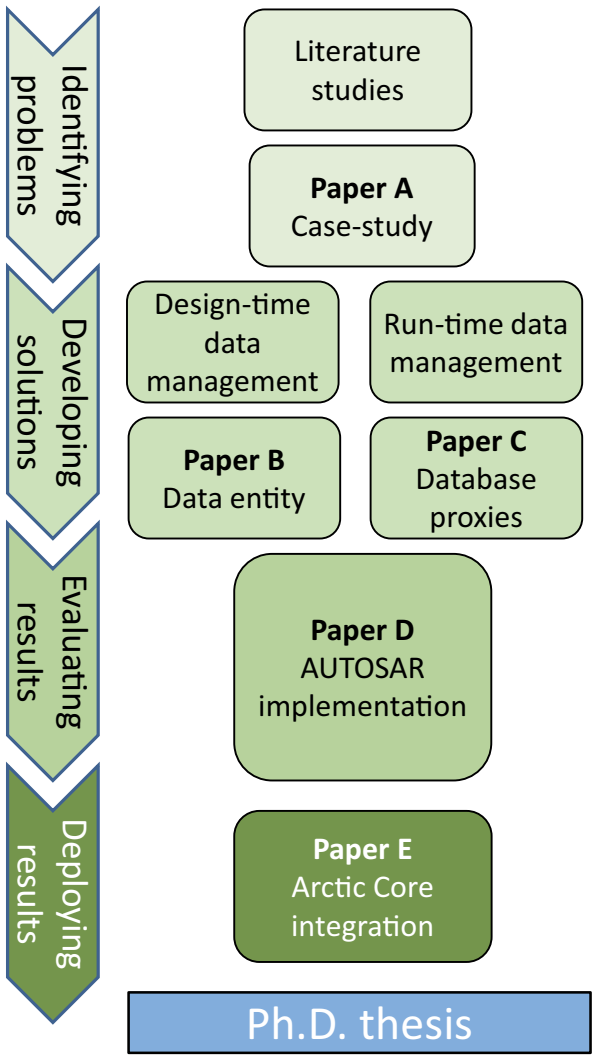


Figure 3.6: Research progression overview

time systems is indeed becoming an increasing challenge for developers and system architects. The case-study, which constitutes **Paper A**, identifies a number of problem areas and possible remedies. These research advances correspond to steps 1 and 2 in Figure 3.5.

- **Developing Solutions:** The continued research, directly targeting the identified problem areas, was sub-divided into two research directions, design-time and run-time data management, which resulted in **papers B** and **C**. **Paper B** presents the data-entity approach that complements design-time tools and techniques with an additional architectural view as well as tools for data management. **Paper C** presents a solution, denoted database proxies, for a successful integration of an RTDBMS into a CBSE setting. Both papers **B** and **C** correspond to steps 3 and 4 in Figure 3.5.
- **Evaluating results:** In the next phase, to validate our approach in an industrial setting, the implementation and evaluation of database proxies in AUTOSAR, a state-of-the-art component-based development architecture, was carried out. An authentic automotive hardware node was used in the evaluation. This resulted in **Paper D**, which corresponds to step 6 in Figure 3.5.
- **Deploying results: Paper E** presents techniques for how to integrate our approach into the commercially available development tool suite, Arctic Core. The use of an RTDBMS in conjunction with database proxies will be included in the meta-model and presented in the graphical user interface, as an additional application design option. This final step corresponds to step 6 in Figure 3.5.

## 3.3 Problem Description, Restated

The continuous increase of complexity and new requirements on data management enhances the challenges with respect to performing design-time and run-time data management in a predictable, efficient and structured manner. Developers need new tools and techniques to aid them with the problems of today and tomorrow.

In an effort to understand the problem concerning data management, (i) this thesis investigates the current issues within industrial embedded systems development, and (ii) what tools and techniques could facilitate that development, i.e. how and in which contexts such systems/tools could be deployed.

To be precise, the thesis focuses on the following:

**F1** How is data currently managed in the industry and what are the main problems concerning design-time and run-time data management?

**F2** How can we support design-time data management within CBSE?

**F3** How can we support run-time data management within CBSE by utilizing state-of-the-art RTDBMS technology?

**F4** How can real-time data management techniques be integrated into an industrial development setting?

## 3.4 Thesis Contributions

The present thesis makes the following major contributions to the area of complex component-based embedded real-time systems:

1. A case-study that emphasizes the importance of data management in order to increase the knowledge and understanding of the system. Ten problem areas within documentation, tool support and routines, as well as remedies, are presented to achieve a more data-centric development strategy. This contribution corresponds to research focus **F1**.

2. The concept of *data entity*, which enables design-time modeling, management, documentation and analysis of run-time data. This contribution corresponds to research focus **F2**.

3. A technique denoted *database proxies*, which enables the integration of an RTDBMS into a component technology. Database proxies are automatically generated glue-code that translates data between component ports and an RTDBMS that resides in the component framework. This contribution corresponds to research focus **F3**.

4. An implementation of tools and techniques for the realization of *data entities* into a component-based development suite named Save CCT. This contribution serves as a validation of contributions 2 and 3.

5. An implementation and evaluation of *database proxies* in AUTOSAR, using industrial tools and hardware. This contribution serves as a possible technology transfer of contribution 3 and corresponds to research focus **F4**.

Part II of the thesis contains five papers, denoted Paper A to Paper E. Each of these papers is summarized below.

**My contribution** to each of the papers has been to define the different concepts, implement the tools, perform the evaluations and be the main author.

### 3.4.1 Paper A

**Paper A:** Design-Time Management of Run-Time Data in Industrial Embedded Real-Time Systems Development, Andreas Hjertström, Dag Nyström, Mikael Nolin and Rikard Land, *13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Hamburg, Germany, September, 2008*

In this paper, we present the results of an industrial case-study conducted at five companies where we have studied the current state of practice in data management and documentation in embedded real-time systems. The case-study identifies a lack of design-time data management, which often results in costly development and maintenance. It confirms that new processes and techniques for achieving an efficient, up-to-date and satisfactory documentation are needed. Furthermore, inadequate tools and routines for data management of internal ECU data results in costly development and maintenance, which is often entirely dependent on the know-how of single individual experts. Ten specific problems are identified, four key observations and six suggested remedies are presented.

### 3.4.2 Paper B

**Paper B:** A Data-Entity Approach for Component-Based Real-Time Embedded Systems Development, Andreas Hjertström, Dag Nyström and Mikael Sjödin, *14th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Palma de Mallorca, Spain, September, 2009*

This paper presents our design-time data management approach, denoted the *data entity* approach. The motivation for this approach stems from identified problems presented in Paper A.

The approach allows efficient design-time management of run-time data to be included in component-based real-time embedded systems development as an additional architectural view that complements the traditional architectural component inter-connections and development view. The *data entity* approach elevates run-time data to be acknowledged as first class objects of the architectural design, and allows data to be modeled and analyzed in an early phase of the development.

The paper also presents a design-time data management tool suite called Embedded Data Commander (EDC), where data entities can be created, retrieved and modified. Furthermore, they can be associated with design entities such as message channels created from the ProCom component architecture development. In addition, the tool allows documentation to be generated from an ongoing project as well as presenting data dependencies, e.g., who the producers and consumers of a data item are. EDC provides tools for data modeling and analysis.

### 3.4.3 Paper C

**Paper C:** Data Management for Component-Based Embedded Real-Time Systems: the Database Proxy Approach, Andreas Hjertström, Dag Nyström and Mikael Sjödin, *Journal of Systems and Software, vol 85, nr 4, p821-834, Elsevier, April, 2012*

To close the gap between two existing techniques used by the industry to manage the complexity and increase the flexibility, of component-based embedded real-time systems development, we introduce the concept of database proxies. Database proxies are automatically generated glue-code synthesized from the system architecture and used to decouple components from the underlying database in order for components to preserve the components encapsulation and possibility of reuse. A component with direct access to an RTDBMS is dependent on that specific RTDBMS and may not be useable in an alternative environment.

The use of an RTDBMS in the component-based setting provides a new range of possibilities, such as structured data management, as well as flexible and predictable access to both critical and non-critical data. By using database proxies in conjunction with state-of-the-art database pointer techniques, developers can employ the full potential of both CBSE and an RTDBMS. With this approach, developers can focus on application development instead of reinventing data management techniques or develop solutions using internal

data structures. As proof of concept this work has been implemented in the SaveCCT framework where a system can be designed with or without a database. In addition, the database proxy properties are generated to glue-code from its specifications and further to target c-code. Furthermore, an evaluation of the execution time overhead and additional memory overhead is in the order of 1-2%.

### 3.4.4 Paper D

**Paper D:** Introducing Database-Centric Support in AUTOSAR, Andreas Hjertström, Dag Nyström and Mikael Sjödin, *7th IEEE International Symposium on Industrial Embedded Systems (SIES), Karlsruhe, Germany, June, 2012*

In this paper we take the database proxy concept from research-oriented techniques to an industrial setting by showing how a real-time database management system can be integrated into the basic software of AUTOSAR by using database proxies. The aim with the approach is to show how a database-centric strategy can facilitate the development and maintenance of an automotive system by providing the proven capabilities of an RTDBMS. Database proxies are used to manage the communication between components on the AUTOSAR Virtual Function Bus (VFB). The COMET RTDBMS is successfully integrated into the AUTOSAR Basic Software (BSW), and evaluated on an authentic automotive hardware node. The evaluation shows that our approach can be used without components being aware of it, jeopardizing system performance or safety. Moreover, this greatly simplifies the development of soft real-time functions that process large data volumes, e.g., for statistics and logging. Our measurements show that the concept only introduces a CPU overhead in the order of 4% under typical workload conditions.

### 3.4.5 Paper E

**Paper E:** Data Management in AUTOSAR: a Tool Suite Extension Approach, Andreas Hjertström, Dag Nyström and Mikael Sjödin, *MRTC Report, submitted for conference publication*

In this paper, our research is transferred from academia to industry as a proof of concept and to demonstrate the usefulness of our research results. We present how a database proxy can be integrated into the development of automotive systems using industrial tools. Our approach enables a clear separation of

concerns between the system architect, component developer, and the Data-Base Administrator (DBA). This separation of concerns allows each part to be managed and reconfigured independent of each other. Furthermore, a plug-in approach, developed for the Arctic Core tool suite and an integration of the Mimer SQL Real-Time [4] database into the basic software of AUTOSAR is presented.

# Chapter 4

# State-of-the-Art

The aim with this chapter is to present relevant background information regarding the development of automotive systems and we introduce some tools and techniques that are important in this respect. This chapter complements Chapter 2 in that we describe the related areas that are mostly orthogonal to the work performed in this thesis.

## 4.1 Automotive Systems

Vehicles have in recent years evolved from mechanical systems to advanced computer-controlled systems where mechanical parts are continuously replaced by computer-controlled functions to achieve higher safety, less pollution, and more comfort. In the early phase of this technological transformation, non-critical tasks such as central locking and parts of the engine-control were handled by small embedded computers. In today's automotive systems, more and more safety-critical functionality is replaced by computers that control breaks, steering, airbags, etc.

In addition, the trend is that automotive systems are evolving from closed stand-alone systems to highly dynamic systems interconnected and communicating with the surrounding environment. There is a lot of research on new technologies such as Car-to-Car (C2C), and Car-to-Infrastructure (C2I) [6] communication. As an example, the system can be used to inform nearby vehicles of possible dangers that have been discovered and even of its own location to avoid a possible collision. In addition, the user demand for integrating third-party applications, such as smart phones and internet connectivity, poses

a range of new challenges concerning areas such as secure data access and handling shared data between safety-related and non-safety-related functionalities.

The amount of software in high-end automotive systems is increasing and is estimated to have reached 1 GB [39]. In addition, an advanced vehicular system can include more than 80 Electrical Control Units (ECUs) which exchange in excess of 2500 signals [40] on several separate bus systems such as CAN, FlexRay, LIN and MOST [41, 42, 43, 44]. To complicate matters further, a high-end system can have more than 2000 functions that often are highly dependent on each other. Consequently, the costs related to software development and electronics have surged and can reach as much as 40% of the total development costs of a car [2]. This has increased the need for flexible platforms that can accommodate entire product lines for several years, in an effort to reduce development costs [45].

The development of functionality in these complex automotive systems requires expertise within the areas of infotainment, engine control, safety-critical applications, etc. As a result of this, Original Equipment Manufacturers (OEM), in-house development is increasingly replaced by Commercial-Off-The-Shelf (COTS) parts from various suppliers with expertise in certain areas [2]. Integrating heterogeneous subsystems from different sources while managing their evolution and maintenance constitutes a great challenge [46]. In addition, as stated by Schulze et al. [39] and Saake et al. [3], the ad-hoc and/or reinvented management of data for each ECU with individual solutions using internal data structures can lead to concurrency and inconsistency problems. A standardized and overall data model and management system has great potential as a solution to deal with the distributed and uncoordinated data in these complex systems [1].

A lot of focus within the automotive industry is the use of a standardized software architecture such as the AUTOSAR [16]; see section 2.3.4. Another approach targeting complexity, cost, time-to-market, etc. when developing automotive systems is Model Driven Engineering (MDE).

### 4.1.1 Model Driven Engineering

Model Driven Engineering (MDE) supplies an abstraction of reality by providing a model of reality that relates to a given aspect of the system. It often does this by only representing a selected part of the system, thus simplifying the overall view. To build a model that represents all aspects of reality or of a system would not only be hard, it would in many respects be impossible to understand [47]. Within computer science, systems are often divided into mod-

els that represent aspects such as requirements, system architecture, validation, etc. MDE has also evolved modeling from being a quite rudimentary form of documentation to serving as formal interchange formats used by tools for precise implementation purposes within computer engineering.

A model must conform to some specified (language and grammar) rules called meta-model in order to be interpretable. There is also the possibility to build hierarchical models, i.e., models of models. An important feature of MDE are to transform one model into another or to generate e.g. code or reports [48].

In the automotive sector, MDE often uses several different modeling languages such as EAST-ADL2 [49], TADL [50], MARTE [51] or SysML [52], which are based on and/or extend concepts from UML.

### 4.1.2 EAST-ADL2

EAST-ADL2 [49] is an automotive architecture description language, developed as a UML 2.0 profile [53] within the ATESST project [54]. EAST-ADL2 aims to support the development of complex automotive software by providing structured system information management at five levels: vehicle level, design level, analysis level, implementation level and operation level.

1. **Vehicle level**: focuses on the features visible to the end users, such as breaks or collision warning. A feature is specified by use cases and requirements to meet, for instance, the configuration of a specific vehicle variant.

2. **Analysis level**: provides analysis support of what the system shall do and describes the functions that enable the available vehicle features. This allows functions to be integrated and validated before the actual software and hardware are developed.

3. **Design level**: includes a behavior description of the functionalities without any implementation constraints in order to meet non-functional constraints such as specific supplier concerns or reusability issues. The focus is on the interaction and behavior of functions.

4. **Implementation level**: a system description of software components, middleware etc.

5. **Operation level**: describes low-level details concerning the deployment on to hardware.

The implementation and operation levels are highly related and complement the AUTOSAR basic software description. In short, AUTOSAR defines the final implementation details and EAST-ADL2 defines the logical and functional architecture aspects.

Neither EAST-ADL2 nor any other of the previously mentioned modeling languages provides specific techniques or support for data management.

## 4.2 Design-Time Tools for Automotive Data Management

Design-time data management is a recognized problem in the automotive industry. Hence, a couple of tools that provide partial solutions to the problem have been developed. To provide data management support at design-time, the dSpace Data Dictionary tool [20] holds information about an ECU for calibration and code generation. It is a central data container for model-independent data management that is used to share information such as interface variables, their scalings, typedefs, etc. throughout an entire project.

A data dictionary is also used for managing AUTOSAR properties, alongside AUTOSAR specification properties at block level in Targetlink [20]. The input to the dSpace data dictionary is templates generated from Simulink [55]. The data dictionary provides access to information such as specifics on C modules, function calls, tasks, variable classes, and data variants. In addition, developers are provided with support to import and export AUTOSAR software-component XML description-files, which can be used by other tools. The information included in the dSpace data dictionary reflects the information included in the software component templates and does not include information about the overall system and what data and signals are included. It is also possible to specify and produce signal lists and spreadsheets with information regarding data. The start of the development process in this tool is to model components and their structure. In contrast to the data entity approach presented in this thesis, the tool does not focus on managing or visualizing the data flow in the system. Neither does it include analysis techniques to make data dependencies visible.

The Automotive Data Dictionary (ADD [21]), is an additional tool that provides a repository solution to centralize data declarations and ensure label and variable uniqueness for companies. ADD has an interface towards MATLAB and Simulink and is used to develop ECUs within the automotive industry. The main goal is to close the gap between software development and

requirements engineering in order to avoid inconsistency throughout the whole development process. It gives the developers an overview of the data specification but does not include any implementation details. Contrary to our data entity approach, ADD mostly focuses on requirements engineering and unique labeling and does not cover information about data flow and data dependencies.

# Chapter 5

# Conclusions and Contingency

In this thesis, we bring together new techniques in order to take a holistic approach to data management in the development of component-based embedded real-time systems.

## 5.1 Conclusions

This research stems from the rapidly growing complexity when it comes to the amount of data and data flow between components in today's embedded real-time systems. This has so far not been addressed by contemporary development techniques. Instead, the focus tends to be on achieving a higher level of abstraction by encapsulating functionality. Current research shows that the state-of-practice for managing data on the system level and internally in individual nodes is not adequate to meet the increasing complexity when building the embedded systems of tomorrow. This was also confirmed by the case-study presented in this thesis.

A starting point was to develop techniques that would enable a Real-Time Database Management System (RTDBMS) to become a native part of the design and to manage the data flow between components. The use of an RTDBMS within data-intensive applications, with high demands on flexibility and structured data management, is not new. It has been a natural next step as systems have evolved and the requirements on data have become increasingly complex. However, the use of an RTDBMS in an embedded setting, and partic-

ular in real-time systems, is still somewhat unconventional, even though today there are several commercial RTDBMSs tailor-made for resource-constrained real-time embedded systems.

To manage data complexity, the use of an RTDBMS in conjunction with Component-Based Software Engineering (CBSE) is an interesting challenge, which we have tackled in this thesis. This approach is not obvious since the design goals of CBSE and RTDBMSs stand in opposition to each other. To overcome these contradictions we have introduced the concept of *database proxies*. In addition, we have shown, through implementations and evaluations using AUTomotive Open System ARchitecture (AUTOSAR) compliant tools and automotive hardware, that this approach offers a number of new possibilities at limited cost with respect to execution time overheads and resource consumption.

Furthermore, this thesis presents a new design-time artifact named *data entity*. The idea behind a data entity is to elevate run-time data to becoming a first-class citizen in the system architectural design and to introduce a data architectural view. Data entities allow data to be documented, modeled and analyzed separately from the actual component implementation. Since data entities are designed in separate, they could be used in other component models, with channels, regular connections or other design approaches then those investigated within this thesis.

In this thesis we have implemented support for data-entities within the SaveComp component technology as well as support for database-proxies with the ProCom and AUTOSAR component technologies. However, the two concepts have not been designed for use with any particular underlying component technology and we believe that results could be generalized to most component technologies that are based on statically configured pipes-and-filter style components. As a run-time backend we have use Mimer Real-Time edition; however any underlying storage-technology could be used with appropriate modification of our glue-code generators. One should note though, that in order to achieve hard real-time support, the storage-technology needs to hard real-time primitives.

It is our firm belief that new, adequate data management techniques are crucial to meet future requirements and to contribute to the evolution of component-based embedded real-time systems development.

## 5.2 Contingency

There is much work to be done in the area of data management in component-based embedded real-rime systems. Existing techniques must incorporate data management as a natural part of the development. In addition, new techniques and tools have to be developed to keep up with the evolution of systems. In this thesis we present techniques that could be of use with respect to some aspects of the development of large complex systems. However, a lot more research and many more solutions are needed in an effort to cover the whole area.

Although we have implemented and evaluated database proxies using commercial tools, this has been done in a "controlled" research environment. A natural next step would be to test the approach in an industrial project to evaluate its usefulness in practice and in the development process. This would be a final step corresponding to steps 5 and 7 in the methodology presented in section 3.2. This would require full-scale integration, in for instance Arctic Core, complete with automatic code generation and validation procedures. Since our implementation is not entirely complete with all functionalities, additional implementation efforts would be necessary.

Some work has been done on the visualization of data entities, as a contingency of the Embedded Data Commander (EDC) tool, which displays data dependency graphs and analytic information. This is an interesting continuation that should be developed and evaluated against other tools and approaches to evaluate its usefulness in an industrial development project.

An additional venue of research, which we have not yet touched upon, is the relation between our proposed techniques and contemporary techniques for model-based development (MBD). While we don't anticipate any conflicts between MBD and our proposals, it remains to be studied, e.g., how and where our data-modeling with data-entities fits into the workflow of an MBD-process.

EDC allows timing requirements to be modeled. However, to validate this type of requirements it would be useful to relate these requirements to timing requirements onto the execution of producers and consumers of data. It would therefore be interesting to map data-timing requirements to Timing Augmented Description Language (TADL) [50] descriptions to allow automated validation through scheduling-analysis.

# Bibliography


[1] Alexander Pretschner, Manfred Broy, Ingolf H. Kruger, and Thomas Stauner. Software Engineering for Automotive Systems: A Roadmap. *Future of Software Engineering*, pages 55–71, 2007.

[2] Manfred Broy. Challenges in Automotive Software Engineering. In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, pages 33–42. ACM, 2006.

[3] Gunter Saake, Marko Rosenmuller, Norbert Siegmund, Christian Kästner, and Thomas Leich. Downsizing Data Management for Embedded Systems. *Egyptian Computer Science Journal*, pages 1–13, 2009.

[4] Mimer SQL Real-Time Edition, Mimer Information Technology, Uppsala, Sweden. http://www.mimer.se, 2012.

[5] eXtremeDB in-Memory Database, McObject. Issaquah, WA USA. http://www.mcobject.com/extremedbfamily.shtml.

[6] AUTOSAR Open Systems Architecture. http://www.car-to-car.org.

[7] Ivica Crnkovic. Component-based Software Engineering - New Challenges in Software Development. In *Software Development. Software Focus*, pages 127–133. John Wiley and Sons, 2001.

[8] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley and Sons, 1996.

[9] D. Box. *Essential COM*, chapter Microsoft Component Object Model (COM.

[10] EJB 3.0 Expert Group. Enterprise JavaBeansTM,Version 3.0 EJB Core Contracts and Requirements Version 3.0. *Final Release*, 2006.

[11] .NET Framework. Microsoft Visual Studio Developer Center. http://www.microsoft.com/NET/.

[12] Kaj Hänninen, Jukka Mäki-Turja, Mikael Nolin, Mats Lindberg, John Lundbäck, and Kurt-Lennart Lundbäck. The Rubus Component Model for Resource Constrained Real-Time Systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[13] Mikael Åkerholm, Jan Carlson, Johan Fredriksson, Hans Hansson, John Håkansson, Anders Möller, Paul Pettersson, and Massimo Tivoli. The Save Approach to Component-Based Development of Vehicular Systems. *Journal of Systems and Software*, 2006.

[14] Rob van Ommering. *The Koala Component Model for Consumer Electronics Software*, volume 33, chapter 3, pages 78 – 85. IEEE Computer Society, Computer archive, 2000.

[15] Tomas Bures, Jan Carlson, Ivica Crnkovic, Severine Sentilles, and Aneta Vulgarakis. ProCom - the Progress Component Model Reference Manual. Technical Report, Mälardalen University, 2008.

[16] AUTOSAR Open Systems Architecture. http://www.autosar.org.

[17] ArcCore. Open Source AUTOSAR Solutions, Göteborg Sweden. http://www.arccore.com.

[18] The Eclipse Foundation, Ottawa, USA. http://www.eclipse.org/.

[19] DAMA International. *The DAMA Guide to the Data Management Body of Knowledge*. Technics Publications, 2009.

[20] dSPACE Data Dictionary, dSPACE Tools. http://www.dspaceinc.com.

[21] Visu-IT, Automotive Data Dictionary. http://www.visu-it.de/ADD/.

[22] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[23] ORACLE, Database Solutions. http://www.oracle.com.

[24] Access - Database Management System, Microsoft. http://www.http://office.microsoft.com/en-us/access/.

[25] MySQL Database, Oracle. http://www.oracle.com/us/products/mysql.

[26] ISO SQL 2008 standard. Defines the SQL language. http://www.iso.org/iso/home.htm, 2009.

[27] Fred R. McFadden, Mary B. Prescott, and Jeffrey A. Hoffer. *Modern Database Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[28] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The Notions of Consistency and Predicate Locks in a Database System. *The Communications of the ACM*, 19(11):624–633, November 1976.

[29] H. T. Kung and John T. Robinson. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems*, 6:213–226, 1981.

[30] Polyhedra In-Memory Database. http://www.enea.com, Sept 2011.

[31] Dag Nyström. *Data Management in Vehicle Control-Systems*. PhD thesis, Mälardalen University, October 2005.

[32] Barbara Gallina and Nicolas Guelfi. SPLACID: An SPL-Oriented, ACTA-Based, Language for Reusing (Varying) ACID Properties. *Software Engineering Workshop, Annual IEEE/NASA Goddard*, 0:115–124, 2008.

[33] Robert K. Abbott and Hector Garcia-Molina. Scheduling Real-Time Transactions: a Performance Evaluation. *ACM Trans. Database Syst.*, 17:513–560, September 1992.

[34] Dag Nyström, Aleksandra Tešanović, Mikael Nolin, Christer Norström, and Jörgen 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. The IEE, June 2004.

[35] Dag Nyström, Aleksandra Tešanović, Christer Norström, and Jörgen Hansson. Database Pointers: a Predictable Way of Manipulating Hot Data in Hard Real-Time Systems. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 623–634, 2003.

[36] Dag Nyström, Mikael Nolin, Aleksandra Tešanović, Christer Norström, and Jörgen Hansson. Pessimistic Concurrency Control and Versioning to Support Database Pointers in Real-Time Databases. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 261–270. IEEE Computer Society, 2004.

[37] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson. A Model for Technology Transfer in Practice. *Software, IEEE*, 23(6):88 –95, nov.-dec. 2006.

[38] Mary Shaw. The Coming-of-Age of Software Architecture Research. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 656–, Washington, DC, USA, 2001. IEEE Computer Society.

[39] Sandro Schulze, Mario Pukall, Gunter Saake, Tobias Hoppe, and Jana Dittmann. On the Need of Data Management in Automotive Systems. In Johann Christoph Freytag, Thomas Ruf, Wolfgang Lehner, and Gottfried Vossen, editors, *BTW*, volume 144 of *LNI*, pages 217–226. GI, 2009.

[40] Nicolas Navet. Trends in Automotive Communication Systems. In *Proceedings of the IEEE*, volume 93, pages 1204–1223, June 2005.

[41] Robert Bosch GmbH. *CAN Specification*. Bosch, Postfach 30 02 40 Stuttgart, version 2.0 edition, 1991.

[42] FlexRay Consortium. http://flexray.com.

[43] Local Interconnect Network. http://www.lin-subbus.org.

[44] Media Oriented Systems Transport (MOST). http://www.mostcooperation.com/home/index.html.

[45] Håkan Gustavsson and Jakob Axelsson. Evaluating Flexibility in Embedded Automotive Product Lines Using Real Options. In *SPLC '08: Proceedings of the 2008 12th International Software Product Line Conference*, pages 235–242, Washington, DC, USA, 2008. IEEE Computer Society.

[46] Alexander Pretschner, Manfred Broy, Ingolf H. Kruger, and Thomas Stauner. Software Engineering for Automotive Systems: A Roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 55–71, Washington, DC, USA, 2007. IEEE Computer Society.

[47] J. Rothenberg. The Nature of Modeling. pages 75–92. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[48] K. Czarnecki and S. Helsen. Feature-Based Survey of Model Transformation Approaches. *IBM Syst. J.*, 45(3):621–645, July 2006.

[49] The ATESST Project, East-ADL Specification. http://www.atesst.org. March, 2012.

[50] The TIMMO Consortium. TADL: Timing Augmented Description Language, Version 2. *TIMMO (TIMing MOdel), Deliverable 6*, October 2009.

[51] MARTE Specifcation Version 1.0 (formal/2009-11-02). http://omgmarte.org. March 2012.

[52] OMG: UML Profile for SysML. http://www.omgsysml.org. March, 2012.

[53] The Object Management Group. Unifed Modeling Language:Superstructure. Version 2.0, OMG document formal/05-07-04, 200.

[54] Advancing Traffic Efficiency and Safety through Software Technology (ATESST). http://www.atesst.org. March, 2012.

[55] The MathWorks. http://www.mathworks.com.