Dipl.- Ing. Andreas Doblander

A Novel Software Framework for Multi-Processor Embedded Smart Cameras

Dissertation

vorgelegt an der
Technischen Universität Graz

zur Erlangung des akademischen Grades
Doktor der technischen Wissenschaften
(Dr. techn.)

durchgeführt am Institut für Technische Informatik
Vorstand: O. Univ.- Prof. Dr. techn. Reinhold Weiß

Oktober 2006

I dedicate this work to my wife Anna.
Your unquestioning support, your understanding,
and your enduring encouragement made this work possible.

Abstract

In traffic surveillance there is a trend towards embedded smart cameras. These cameras provide on-site video analysis to detect dangerous traffic situations and compute traffic statistics that can be used for traffic management. Typically, multi-processor platforms comprising several DSPs and other specialized processors are used. In this thesis the ***Smart****Cam* is considered as target platform.

This thesis presents a novel middleware for loose coupling of DSP algorithms. Because of tight resource constraints standard solutions are not appropriate and, therefore, an efficient and light-weight middleware layer is suggested for the ***Smart****Cam*. By introducing only minimal indirection it also provides little transfer time overhead. Transparent communication within a single processor and between different processors via the local PCI bus is supported by a special proxy mechanism transparent to the application developer. For easy service discovery a simple directory service is provided that allows algorithms to find data sinks and sources dynamically according to their requirements.

For wide-spread use of smart cameras they require basic fault-tolerance mechanisms. The system has to stay operational in case of unexpected events without external intervention. Because of tight device cost limitations in the industry it is especially interesting to focus on software fault-tolerance. Redundant hardware would result in increased product costs and further to reduced market success.

Currently, in most systems ad hoc mechanisms for software fault-tolerance are employed separately for each application. In contrast to that the aim of this work is to provide dedicated services for fault-tolerance in the middleware. An important advantage of a middleware solution is that program code for fault-handling is shared by all applications so that scarce system resources are saved. Based on the flexibility with respect to QoS and dynamic reconfigurability introduced by the publisher-subscriber middleware a graceful degradation approach is the most reasonable approach for software fault-tolerance. Video surveillance applications can mostly be stripped down to an essential set of algorithms so that a degraded state also satisfies minimum requirements—at least for short time periods or in some parts of the system, respectively. To decide on which algorithms to degrade the framework relies on application-specific knowledge for their prioritization.

In the middleware framework for the DSPs in the ***Smart****Cam* a special algorithm component model is derived that supports high-level model-based development of video analysis algorithms. That is, algorithms are developed using a model-based environment like Matlab/Simulink. Code for the DSP framework is then directly generated from the model. An evaluation of the code generation tools in Matlab/Simulink suggests this approach to be feasible.

A prototype of the presented framework has been implemented on the ***Smart****Cam* platform. Experimental evaluation shows that the overhead introduced by the middleware mechanisms is very small. This efficiency is substantial for use with video analysis algorithms on our embedded smart camera platform.

Kurzfassung

In der Videoüberwachung ist zunehmend ein Trend hin zu eingebetteten intelligenten Kameras zu beobachten. Diese Smart Cameras ermöglichen Videoanalyse direkt vor Ort. Damit können gefährliche Situationen erkannt werden und Verkehrsstatistiken für das Verkehrsmanagement berechnet werden. Die für die Videoverarbeitung nötige Rechenleistung wird meist mittels Multi-Prozessor Plattformen aus DSPs und anderen spezialisierten Prozessoren zur Verfügung gestellt. In der vorliegenden Arbeit wird die ***Smart****Cam* als Zielplattform verwendet.

Diese Arbeit stellt eine neuartige Middleware für lose gekoppelte DSP Algorithmen vor. Wegen der begrenzten Ressourcen sind dafür Standardlösungen nicht geeignet. Deshalb wird für die ***Smart****Cam* eine effiziente und schlanke Middleware-Schicht vorgestellt. Durch minimale Indirektion ist der Kommunikations-Overhead sehr gering. Transparente Kommunikation innerhalb eines Prozessors und zwischen mehreren Prozessoren erfolgt über einen speziellen Proxy-Mechanismus. Für das Auffinden von Services wird ein einfaches Directory Service zur Verfügung gestellt. Damit können Algorithmen Datenquellen und -senken dynamisch nach ihren Anforderungen auffinden.

Damit intelligente Kameras auch in größerem Umfang eingesetzt werden können, benötigen sie zumindest grundlegende Mechanismen zur Fehlertoleranz. Bei unvorhergesehenen Ereignissen soll zumindest ein Notbetrieb aufrechterhalten werden und zwar möglichst ohne menschliche Intervention. Aus Kostengründen ist in der eingebetteten Videoüberwachung auch Software Fehlertoleranz vorzuziehen, da Hardware-Redundanz die Produktkosten erhöht und damit den Markterfolg senkt.

Derzeit werden meist eigens in jeder Anwendung ad-hoc Mechanismen zur Fehlertoleranz eingesetzt. Im Gegensatz dazu ist es das Ziel dieser Arbeit spezielle Fehlertoleranz-Dienste in der Middleware anzubieten. Der Vorteil einer solchen Lösung ist, dass der Programm-Code für die Fehlerbehandlung von allen Anwendungen geteilt werden kann, wodurch Ressourcen geschont werden. Durch die Flexibilität in der dynamischen Rekonfiguration von QoS Einstellungen und Algorithmenzusammensetzugen der Publisher-Subscriber Middleware wird ein Graceful Degradation Ansatz gewählt. Tatsächlich können Videoüberwachungsanwendungen, zumindest für eine bestimmte Zeit, mit reduzierter Qualität betrieben werden. Welche Algorithmen degradiert werden, wird vom Framework aufgrund anwendungsspezifischer Information über deren Priorisierung entschieden.

Das präsentierte Middleware-Framework stellt auch ein Komponentenmodell für DSP Algorithmen zur Verfügung. Damit wird auch der Einsatz von Modellbasierten Entwicklungsmethoden ermöglicht. Das heißt, Algorithmen werden in einer Umgeben, wie z.B. Matlab/Simulink, entwickelt und der entsprechende DSP Programmcode wird daraus automatisch generiert. In einer experimentellen Evaluierung wurde die Durchführbarkeit dieses Ansatzes bestätigt.

Ein Prototyp des präsentierten Frameworks wurde auf der ***Smart****Cam* implementiert. Experimentelle Untersuchungen zeigen, dass der Overhead durch die Middleware besonders gering ist. Die resultierende Effizenz ist besonders wichtig für Videoanalyse auf eingebetteten intelligenten Kameras.

Extended Abstract

In traffic surveillance there is a trend towards embedded smart cameras. These cameras provide on-site video analysis to detect dangerous traffic situations and compute traffic statistics that can be used for traffic management. To provide enough computing power for the video analysis algorithms high performance embedded computing platforms are required. Typically, multi-processor platforms comprising several DSPs and other specialized processors are used. In this thesis the ***Smart****Cam* is considered as target platform.

Because of the limited resources of the embedded smart camera platform it is not possible to run all intended analysis algorithms simultaneously. In previous work in the ***Smart****Cam* project a framework was established that allows for all algorithms to be loaded and unloaded on demand at runtime. An important restriction of this framework was that only predefined algorithm connections could be realized. In order to extend the platform's flexibility in terms of connecting video analysis algorithms dynamically at runtime this thesis presents a novel middleware for loose coupling of DSP algorithms [1]. Because of tight resource constraints standard solutions are not appropriate and, therefore, an efficient and light-weight middleware layer is suggested for the ***Smart****Cam* [2].

The presented real-time publisher-subscriber middleware (PS-MW) is a very light-weight architecture that supports loose coupling of tasks in the given dynamic application environment. By introducing only minimal indirection it also provides little transfer time overhead. Transparent communication within a single DSP and between different DSPs via the local PCI bus is supported. To abstract from the PCI bus a special proxy mechanism transparent to the application developer is introduced. For easy service discovery a simple directory service is provided that allows algorithms to find data sinks and sources dynamically according to their requirements. In that way it is possible to rearrange algorithm connections at runtime if special surveillance conditions request for an adapted analysis pipeline. Alternatively, it is also possible to rearrange and reconfigure algorithms to make the system more power-efficient if it is running on batteries [3] [4].

[1] Andreas Doblander, Bernhard Rinner, Norbert Trenkwalder, and Andreas Zoufal. A light-weight Publisher-Subscriber Middleware for Dynamic Reconfiguration in Networks of Embedded Smart Cameras. In *Proceedings of the 5th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems*, Madrid, Spain, February 2006. World Scientific and Engineering Academy and Society

[2] Michael Bramberger, Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Distributed smart cameras for surveillance applications. *Computer*, 39(2):68–75, February 2006

[3] Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. A Novel Software Framework for Power-Aware Reconfiguration in Distributed Embedded Smart Cameras. In *Proceedings of the 12th IEEE International Conference on Parallel and Distributed Systems*, volume 1, pages 281–288, Minneapolis, Minnesota, USA, July 2006. IEEE Computer Society

[4] Andreas Doblander, Arnold Maier, Bernhard Rinner, and Andreas Zoufal. An Efficient Middleware for Power-Aware Service Reconfiguration in Multi-DSP Smart Cameras. In *Proceedings of the 2nd IEEE International Conference on Information and Communication Technologies: From Theory to ApplicationsSoftware Engineering*, pages 1093–1094, Damascus, Syria, April 2006. IEEE

In the middleware framework for the DSPs in the ***Smart****Cam* a special algorithm component model is derived that may also be used to support high-level model-based development of video analysis algorithms [5]. That is, algorithms are developed using a model-based environment like Matlab/Simulink and code for the DSP framework is directly generated from the model. An evaluation of the code generation tools in Matlab/Simulink suggests this approach to be feasible although substantial effort has to be made to adapt to custom hardware [6]. However, the increasing complexity of embedded applications drives development of the tools so that they are continuously improved.

As typical future surveillance settings will be based on a large number of smart cameras it is necessary to aid operators and system providers in operating the system and maintenance, respectively. Especially maintenance in such large-scale distributed systems plays an increasing role in the industry. Therefore, it is desirable to provide systems of smart cameras with self-management capabilities. In other words such systems should act as autonomously as possible.

A basic requirement for self-management and autonomous operation is fault-tolerance with its enabling mechanisms fault-detection and fault-diagnosis. It allows a system to stay operational in case of unexpected events without external intervention. Because of tight device cost limitations in the industry it is especially interesting to focus on software fault-tolerance. Redundant hardware results in increased product costs which in turn means reduced market success.

Currently, in most systems today ad hoc mechanisms for software fault-tolerance are employed separately for each application. In contrast to that the aim of this work is to provide dedicated services for fault-tolerance in the middleware. An important advantage of a middleware solution is that program code for fault-handling is shared by all applications so that scarce system resources are saved. Based on the flexibility with respect to QoS and dynamic reconfigurability introduced by the publisher-subscriber middleware a graceful degradation approach is the most reasonable approach for software fault-tolerance [7]. In fact, video surveillance applications can mostly be stripped down to an essential set of algorithms so that a degraded state also satisfies minimum requirements—at least for short time periods or in some parts of the system, respectively.

The presented middleware-based software fault-tolerance approach for intelligent video surveillance applications is organized in two layers [8]. First, the node-

[5] Andreas Doblander, Bernhard Rinner, Norbert Trenkwalder, and Andreas Zoufal. A Middleware Framework for Dynamic Reconfiguration and Component Composition in Embedded Smart Cameras. *WSEAS Transactions on Computers*, 5(3):574–581, March 2006

[6] Andreas Doblander, Dietmar Gösseringer, Bernhard Rinner, and Helmut Schwabach. An Evaluation of Model-Based Software Synthesis from Simulink Models for Embedded Video Applications. *International Journal of Software Engineering and Knowledge Engineering*, 15(2):343–348, April 2005

[7] Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Improving Fault-Tolerance in Intelligent Video Surveillance by Monitoring, Diagnosis and Dynamic Reconfiguration. In *Proceedings of the Third IEEE-Workshop on Intelligent Solutions in Embedded Systems, Hamburg, Germany*, pages 194–201, 2005. ISBN 3-902463-03-1

[8] Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Increasing Service Availability in Intelligent Video Surveillance Systems by Fault Detection and Dynamic Reconfiguration. In *Proceedings of the Telecommunications and Mobile Computing Workshop on Wearable and Pervasive Computing*, Graz, Austria, March 2005

level layer focuses on a single smart camera. That is, only algorithms within a node and their interactions are observed. Second, the system-level layer is concerned with the overall network of smart cameras. Data from neighboring nodes is exploited to derive decisions on the network and the alive-states of whole nodes.

Based on the algorithm component model defined by the framework each algorithm in the system has to provide the middleware framework with information on its QoS capabilities and resource requirements. Furthermore, algorithms have to provide an interface that is used by the middleware to monitor the algorithms's state.

In case of faults graceful degradation is carried out by appropriate reconfiguration of algorithms and their QoS settings. The (surveillance) application, i.e., the developer, has to communicate the minimum QoS requirements of the used algorithms to the middleware. Additionally, a situation-dependent prioritization of algorithms has to be provided. Based on this importance measure the middleware reconfigures the overall application, i.e., all algorithms. Replication of node and system state information among neighboring nodes ensures that reconfiguration is also possible in case of node failures.

A prototype of the framework has been implemented on the ***Smart****Cam* platform. Experimental evaluation shows that the overhead introduced by the middleware mechanisms is very small. This efficiency is substantial for use with video analysis algorithms on our embedded smart camera platform.

Thesis Contribution

- The thesis presents a novel light-weight communication middleware for embedded multi-DSP platforms. It is based on the publisher-subscriber mechanism and supports flexible reconfiguration of DSP algorithms.
- Middleware mechanisms for node-level and system-level fault-tolerance in the domain of video surveillance by smart cameras are introduced. That is, application-specific knowledge is used for reasoning about algorithm reconfigurations to achieve graceful degradation.
- A prototype implementation of the middleware has been realized on a heterogeneous multi-processor platform featuring a network processor (Intel XScale) and multiple DSPs (Texas Instruments C64x).
- Reconfiguration and communication performance have been evaluated on the prototype platform mentioned above. The approach is shown to be very efficient in terms of memory consumption and resulting CPU load.
- High-level software development using the Matlab/Simulink environment has been examined and evaluated concerning its applicability to automatic code generation of video analysis algorithms.

Acknowledgments

This work has been conducted for the ***Smart****Cam* project at the Institute for Technical Informatics and I would like to thank Professor Reinhold Weiß, the head of the institute, and my advisor Bernhard Rinner for their guidance and support during my time at Graz University of Technology. Furthermore, thanks to all colleagues at the institute for their inputs in many valuable discussions.

Very special thanks go to my colleagues and friends Michael Bramberger and Arnold Maier for their priceless help and support, as well as their friendship that made it a real pleasure for me to work on this project. My appreciation also goes to Markus Quaritsch, Norbert Trenkwalder, and Karima Klamminger for their devotional work and commitment to the project.

I am also very grateful for the strong support of my work from the *ARC Seibersdorf research GmbH*. Thank you for the financial support that made this work possible. Especially, I would like to thank the people at the *Video and Safety Technology* group and its former head Helmut Schwabach for their continual support and help.

Finally, my biggest thank to my wife Anna for her understanding and enduring encouragement during the last years; and my parents for their guidance throughout my whole life and their unprecedented confidence in my decisions.

Thank you!

ANDREAS DOBLANDER

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Recent advances in computing, communication and sensor technology are pushing the development of numerous new applications. This trend can especially be observed in pervasive computing. Intelligent video surveillance based on smart cameras is an example for an innovative intelligent infrastructure application.

Smart cameras [WOL02, LLWO04] are equipped with high-performance onboard computing and communication devices. They combine video sensing, processing and communication within a single embedded device. Networks of distributed smart cameras are an emerging technology for a broad range of important applications, including smart rooms, surveillance, tracking and motion analysis. By having access to many views and through cooperation among the individual cameras, these networks have the potential to realize many more complex and challenging applications than single camera systems.

Surveillance applications pose strong requirements on the camera's hardware and software [FMR00]. Typically, the cameras have to execute demanding video processing and compression algorithms. These surveillance tasks running on the cameras offer different QoS-levels and may be adapted in response to events detected in the monitored area. The distributed surveillance architecture has, therefore, to be scalable and flexible.

In previous work [Bra05, BDM+06] a smart camera has been designed—it is called the ***Smart****Cam*—as a fully embedded system focusing on aspects such as dynamic load distribution [BRS05] and power consumption and QoS-management [Mai06]. The smart camera is realized as a scalable, embedded high-performance multi-processor platform consisting of a network processor and a variable number of digital signal processors (DSP).

Several requirements have to be met by the system software to employ this flexible high-performance platform in real-world distributed (surveillance) applications:

- Flexibility of algorithm configuration, i.e., how tasks are composed to build the application.

- Scalability concerning the number and the different types of employed surveillance tasks.
- Low resource consumption so that resources are spared for surveillance tasks and image buffers.
- Low performance overhead to not hinder real-time operation of surveillance tasks.
- Real-time operation to meet requirements of surveillance tasks. At least frame rate requirements of all tasks have to be met.

To meet the above requirements a multi-layer heterogeneous software framework was devised for our smart cameras. Since our smart cameras comprise a network processor and several DSPs the framework is divided into two parts. First, the part running on the network processor—the so-called *SmartCam-Framework* (SC-FW)—hosts a mobile agent system (MAS) and provides system-level task distribution services. Second, the part employed on the DSPs—the so-called *DSP-Framework* (DSP-FW)—is based on a publisher-subscriber middleware approach. The responsibility of this DSP-FW is to provide local task communication where all tasks are dynamically loaded and unloaded. This is in contrast to current practice where functionality is mostly changed by re-programming a device with a new software image. That implies that the device has to be rebooted and cannot provide its service for some time. To support dynamic alteration of surveillance applications loose coupling of tasks is required [HC01]. Therefore, a publisher-subscriber approach was chosen to support this loose coupling. Given the tight resource requirements in our system standard middleware solutions are not appropriate. Special implementations for resource constrained embedded devices could not be used because they are not available for the DSPs and the operating system we use.

This middleware allows to dynamically change the camera's functionality, i.e., various tasks can be loaded and unloaded at runtime or their QoS-level can be adapted dynamically. Based on this reconfiguration capabilities our smart cameras can be combined to a distributed embedded (surveillance) system and support cooperation and communication among the individual cameras.

1.2 Motivation

1.2.1 Dynamic reconfiguration

In embedded computing the general practice for changing features of employed devices or change entire software configurations is to do image swapping. That is, the overall software image is exchanged. Figure 1.1 illustrates the principle of changing the software image. This procedure is not very flexible and means that the whole device has to be rebooted after successful software image updates. Most video surveillance applications require at least the video storage features to run uninterruptedly so changing software images with every change in the camera configuration is not an option.

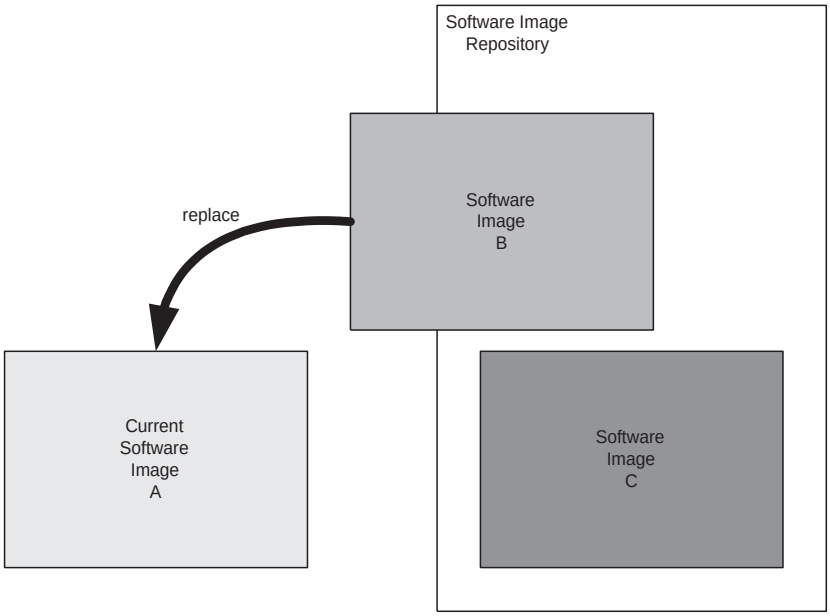

Figure 1.1: The state-of-the-practice in changing features and algorithms in mission-critical embedded devices.

Furthermore, it is not very efficient to use such a brute-force mechanism for changing algorithm configurations because of the unacceptable reaction times and unnecessary down-times. In this context an algorithm configuration means the composition of different algorithms to a whole application as well as QoS levels of individual algorithms if they cannot be changed by runtime parametrization of the respective algorithm. Parameterizations that cannot be done dynamically when the algorithm is running are mostly such that they require substantial reorganization of internal algorithm buffers. A typical example is changing the input image resolution of the MPEG-4 encoder because internal memory management of the algorithm prohibits changing buffers of a running instance.

As there are several of these reconfiguration limitations when relying only on changing overall software images a more flexible approach was introduced in the first ***Smart****Cam* prototype [BBRS04, BRS04]. There it was possible to load and unload several algorithms at runtime. But all algorithms were restricted to be known beforehand. It was not possible to load new algorithms for extending functionality. Another severe restriction was that it was not possible that two instances of the same algorithm were run on the same ***Smart****Cam*. The root cause for these limitations was a static binding of algorithms to communication channels over the PCI bus. In Figure 1.2 the situation concerning algorithm reconfiguration in the first ***Smart****Cam* prototype is illustrated.

Given the above limitations it is an important goal of this work to provide a flexible software framework for dynamic software reconfiguration in the ***Smart****Cam* prototype. In this thesis the notion of *dynamic integration* is introduced for the ***Smart****Cam* software framework to describe the presented reconfiguration approach. Figure 1.3 shows this principle of dynamic integration where algorithms

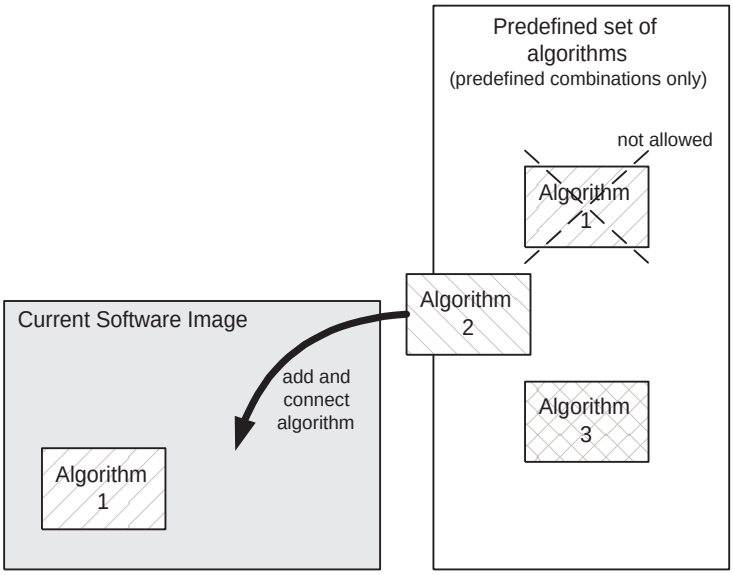

Figure 1.2: A more flexible approach to reconfiguration as it was used in the first ***Smart**Cam* prototype. But still the intended reconfigurations could not be done dynamically. Only predefined algorithms can be exchanged at runtime. Several QoS levels that cannot be applied by runtime parametrization also have to be provided as predefined algorithm instances.

complying to the *DSP algorithm component model* defined in this work can be loaded and unloaded arbitrarily.

1.2.2 Towards Autonomous Smart Cameras

In embedded video surveillance it is a key feature of smart cameras to relieve operators in central surveillance stations from continuously observing hundreds of different monitors. Therefore, it is necessary to provide each smart camera with substantial self-management capabilities and profound detection algorithms. The aim is that a camera generates alarms in case of dangerous situations in the observed scene. Video streams from dangerous scenes can then be displayed to an operator as a reaction to the alarm. In such a scenario only the most relevant video streams are presented to the operators preventing them from information overflow.

But in order to be of real assistance a network of smart cameras has to reach an adequate level of trustworthiness so that operators rely on the systems alarms. Therefore, each ***Smart**Cam* in a future surveillance network has to provide self-monitoring and fault-tolerance mechanisms [DMRS05b, DMRS05a].

Furthermore, it is necessary for a ***Smart**Cam* to keep house with their energy resources. This might not be a big problem for, e.g., tunnel installations or indoor surveillance. But when fielded in rural and remote environments with less infrastructure a ***Smart**Cam* has to be powered by solar energy or by batteries [MRSS06]. To achieve an energy-efficient configuration of a single ***Smart**Cam* and whole net-

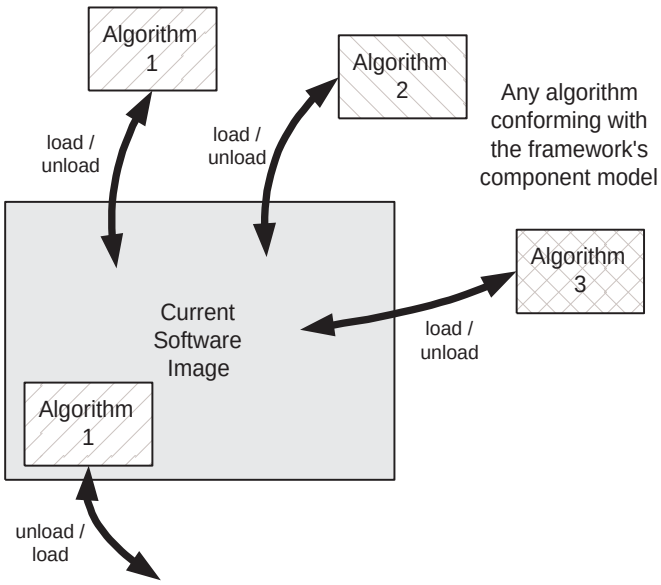

Figure 1.3: The new reconfiguration approach for dynamic algorithm reconfiguration in the ***Smart****Cam* as presented in this thesis. Improved flexibility introduces full dynamic reconfiguration capabilities.

works of ***Smart****Cams* the software framework's dynamic reconfiguration capabilities are used as a basis [DMRZ06, DMRS06].

When pushing the concept of an embedded smart camera further to integrate a multi-sensor device with sensor-fusion capabilities the necessity of dynamic reconfiguration and self-management increases even more [BDJ+06]. More sensors imply more detection algorithms and, therefore, an increased number of reconfiguration events and increased monitoring effort. Efficient inter-task communication as well as a well-defined algorithm model are key for successful employment of complex multi-sensor devices. Especially, with the increased fault complexity due to the increasing number of algorithms and events the dynamic reconfiguration and flexible task communication mechanisms provide the means for gracefully degrading system performance in case of faults.

1.2.3 Raising the Level of Abstraction in Embedded Software Development

To cope with increasing software complexity in distributed embedded multiprocessor architectures such as smart cameras it is necessary to raise the level of abstraction in the software development. In recent years the object-oriented paradigm has been adopted for embedded software but still the level of abstraction is not adequate for the complexity inherent in current designs [Sch06].

In the research community there are efforts to introduce high-level approaches such as model-based software development or model-integrated computing to the embedded software industry [BGK+06, SSBG03].

Especially, in multi-processor architectures it is necessary to aid developers and integrators in their complex tasks. As a smart camera is such a system it is essential for future applications of the ***Smart****Cam* to step towards improved development paradigms. Component-oriented development combined with model-based approaches that also provide rich visualization capabilities and domain-specific component libraries can significantly raise the level of abstraction.

1.3 Scientific Focus

Based on the introductory ideas of the previous sections the research goal of this thesis can be stated as follows:

To develop a software framework for distributed embedded smart cameras focussing on

- *flexibility and scalability with respect to type and number of analysis algorithms.*
- *efficiency in terms of memory consumption and runtime overhead.*
- *fault-tolerance mechanisms to step towards autonomous operation of networks of smart cameras.*
- *support for system integrators and application / algorithm developers by appropriate abstractions.*

so that the demanding requirements imposed by future video surveillance applications can be met.

A prototype implementation of the middleware was realized on a heterogeneous multi-processor platform featuring a network processor (Intel XScale) and multiple DSPs (Texas Instruments C64x). Reconfiguration and communication performance were evaluated on the prototype platform. The approach is shown to be very efficient in terms of memory consumption and resulting CPU load.

Based on the dynamic reconfiguration capabilities of the ***Smart****Cam* graceful degradation of system services is examined as a means for improving service availability in spite of faults. Specialized prioritization of algorithmic services are devised in view of the intended application of the ***Smart****Cam* in video traffic surveillance.

To meet increasingly tight time-to-market demands and increasing software complexity in embedded surveillance systems a high-level development paradigm should be evaluated for their feasibility in this field. As is shown in Figure 1.4 the current development process is inefficient in that it needs the system integrator to recode the algorithm to port it to the actual target hardware. The major problem is that video analysis experts focus on the algorithmic details and are mostly not familiar with the target hardware. They devise their algorithms in high-level modeling environments like Matlab/Simulink or using abstract programming language libraries that cannot be efficiently ported to the embedded target hardware.

To solve this the software framework should be devised such that support for high-level development can easily be integrated in the future. As a typical environment for algorithm exploration the Matlab/Simulink development suite was

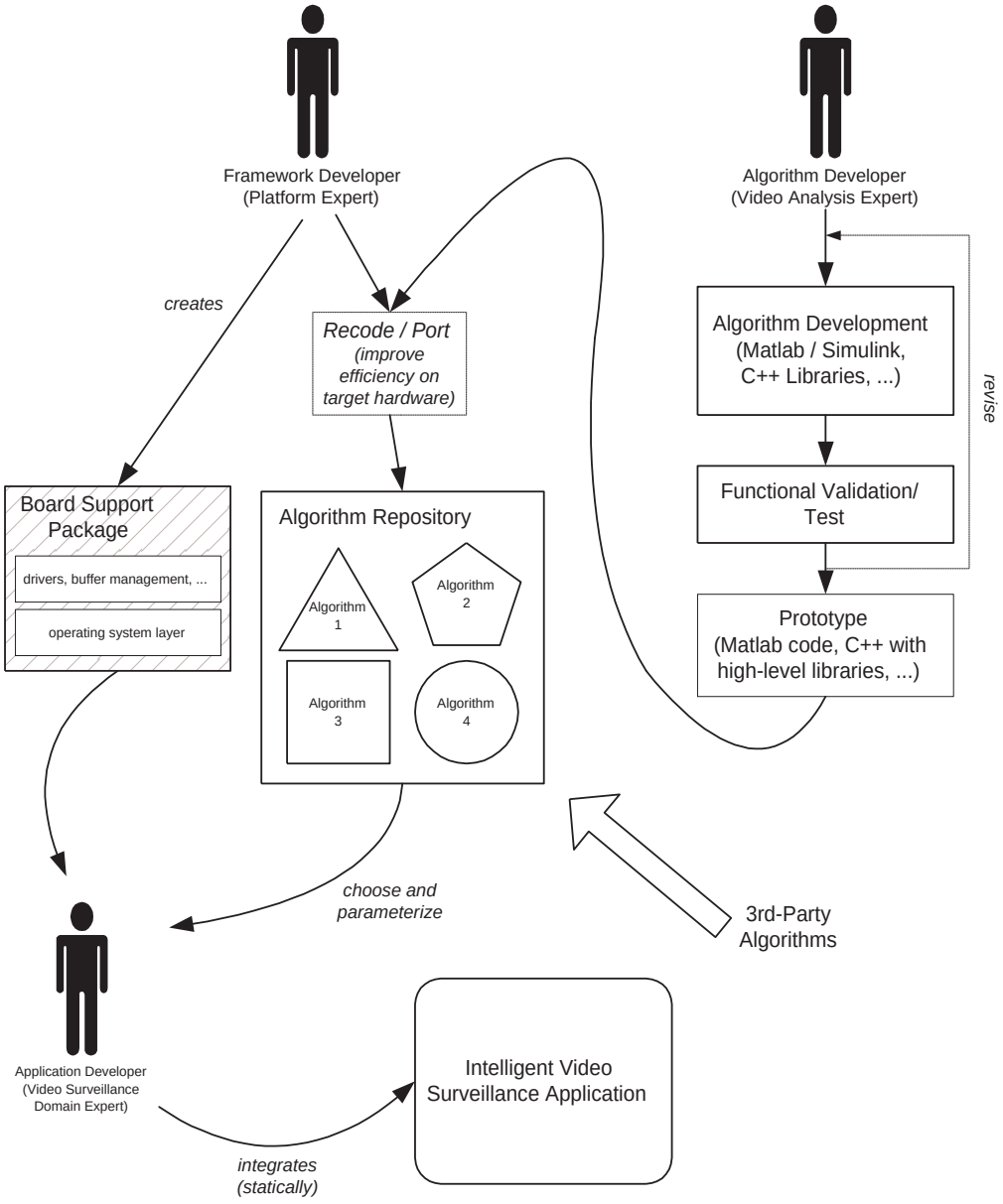

Figure 1.4: The development process as it is currently state-of-the-practice at ARC Seibersdorf research GmbH. As algorithm experts are not familiar with the target hardware platform the system integration is inefficient because integrator have to recode algorithms to adapt to platform specialities.

chosen as an example for a model-based development environment following a model-based approach.

In Figure 1.5 the resulting improved development process is shown in context of a video surveillance application as it would be appropriate, e.g., at the ARC Seibersdorf research GmbH Video and Safety Technology group. This process has the benefit that the algorithm experts are decoupled from the platform by the model-based development environment. Code generation templates and in part also block libraries are created by platform experts. Therefore, automatically generated algorithms are appropriate for the target hardware and efficient in spite of the high-level approach.

1.4 Thesis Outline

The remainder of the thesis is organized as follows.

- Chapter 2 first presents a problem analysis that clarifies the driving forces and motivating aspects for this thesis. The presentation of these different aspects is organized in several sections. Second, it discusses relevant related research work in industry and academia in Section 2.5.
- Chapter 3 discusses the novel middleware fault-tolerance approach for embedded smart cameras. The light-weight publisher-subscriber software architecture as well as the middleware fault-tolerance mechanisms are described.
- Chapter 4 first presents the implementation of the framework as it was used for evaluation. Then it discusses an experimental evaluation of the presented approach. It demonstrates the feasibility and efficiency of the presented approach.
- Chapter 5 concludes the thesis by summarizing the most important ideas and conclusions. In the end possible directions for future work are discussed in Section 5.1.

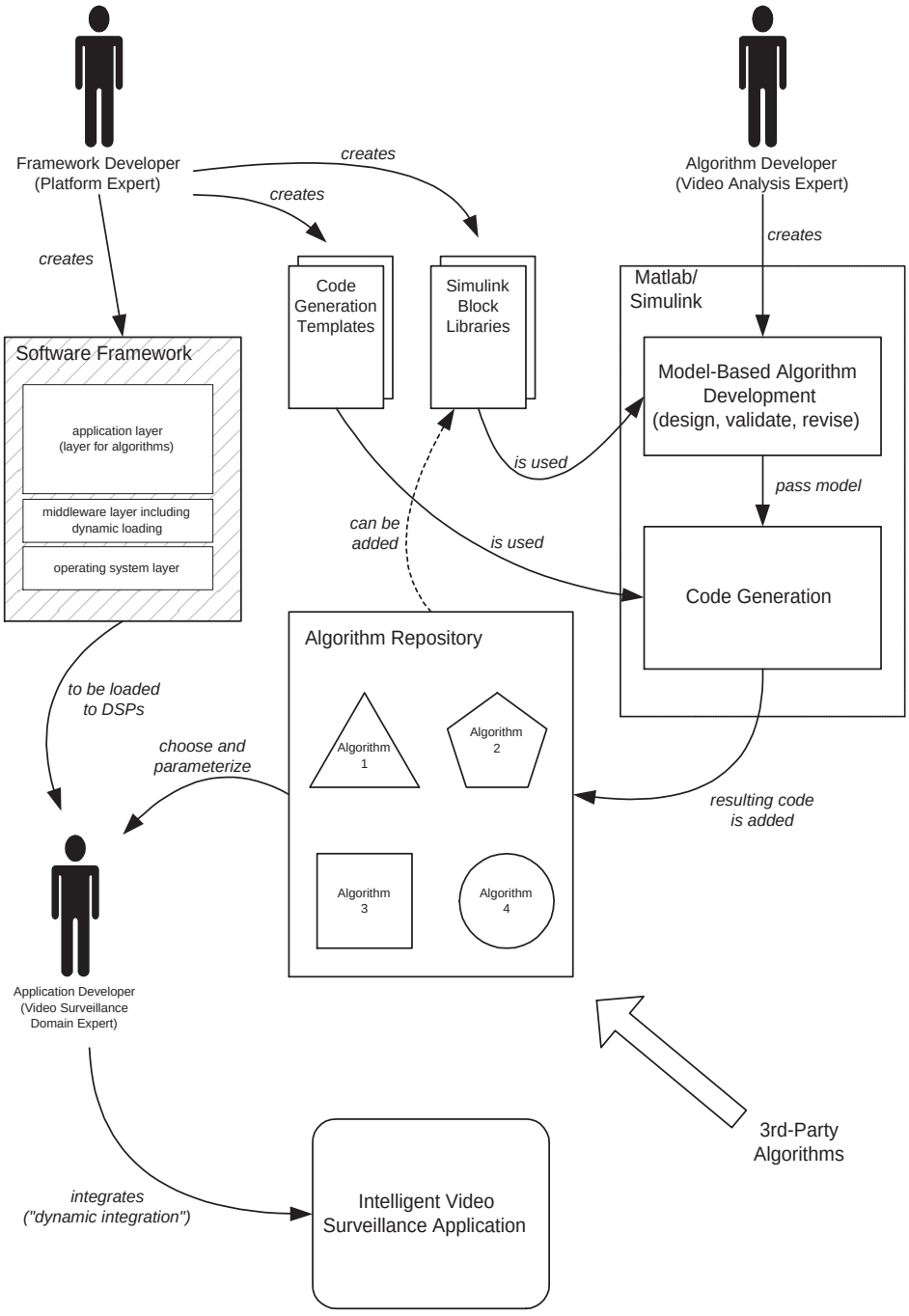

Figure 1.5: The intended development approach based on the software framework presented in this thesis. Involved roles and their relations among each other and the framework are illustrated.

Chapter 2

Problem Analysis and Related Work

In this chapter the main problems in the research field are stated and analyzed in Section 2.1. The intention is to provide some insight into the driving forces in the field of distributed embedded video surveillance. Key aspects such as component composition, fault-tolerance, and high-level development are treated in separate sections. In Section 2.5 of this chapter related work by other researchers in the field is presented. The discussion of related work is organized in different subsections corresponding to the main ideas of the thesis.

2.1 Distributed Embedded Video Surveillance

Video surveillance is an upcoming field in industry and research. Due to the increased demand for security applications video surveillance is employed in many different areas. There are indoor surveillance applications in casinos, hotels, corporate and governmental buildings and outdoor surveillance on parking lots, public places and, of course, traffic surveillance on highways. Given tight time-to-market and cost demands and the vast number of devices needed for a surveillance application there is also a strong trend towards embedded solutions. Therefore, smart cameras are an integral part of future surveillance systems [WOL02].

Another interesting recent development is aerial video surveillance [SLS+04] based on wireless video stream transmission for rural areas where no network infrastructure is available. It is also an embedded solution and would even be an interesting application domain for intelligent cameras because they have the potential to reduce transmission bandwidth that is scarce in wireless settings.

Embedded video Surveillance applications are very demanding with respect to computing and memory resources. Therefore, it is especially important to provide these applications with flexible mechanisms for resource allocation and dynamic loading capabilities. The notion of *runtime configuration capable embedded systems* by Nitsch and Kebschull [NK02] is quite similar to our understanding of a dynamically configurable system. However, we do not consider hardware reconfiguration as suggested in their work. Nitsch and Kebschull also use Enterprise Java Beans as

enabling technology which is too resource intensive for our application in smart cameras.

Modern embedded applications are increasingly complex in that they do not serve a single purpose anymore but fulfill several different tasks. Combined with the limited resources the embedded platforms provide software development is becoming more and more cumbersome and error prone. Better tool support and better methodologies are required to cope with this situation. To provide complex features and major applications special techniques such as dynamic loading and distributed computing are employed. Naively running all possible services becomes impossible and, additionally, consumes extra energy [LWD+06]. It is the aim to provide all the complicated services needed by the application but perform the services only at the right time.

To aid operators and users of highly complex embedded systems several sophisticated mechanisms have to be provided. That is, these systems get more and more intelligent. One major goal is to build the systems in a way that they can manage most of their runtime and maintenance tasks by themselves. Therefore, such systems are often called "autonomic systems". A major requirement for autonomous behavior is a fault-tolerance concept so that the system is able to recover from failures without (substantial) intervention by an operator [KC03].

At least a system state with reduced functionality should be achievable instead of complete system failure. Graceful degradation is a popular and well researched mechanism to achieve prolonged operation in case of a fault. Of course, there is a vast number of other fault-tolerance [1] and high dependability approaches around. Especially, redundant hardware and highly sophisticated software methods like, e.g., n-version programming and recovery blocks [Lyu95] are employed in high dependability systems. However, in the case of embedded surveillance the focus is on price-per-unit which inhibits too sophisticated and expensive solutions. But nevertheless, with the dynamic reconfiguration capabilities and the framework presented in this thesis it is possible to devise a graceful degradation scheme to provide the system with increased survivability with only reasonable overhead.

2.2 Software Components and their Composition

Software components are increasingly used in software development to handle complexity and cost requirements. In contrast to hardware components, there are two distinct variants of components [MPT04] in software. First, there are components that can only be used as provided. These are called Commercials Off-The-Shelf (COTS). Second, there are components that can be modified and customized by the user. Therefore, the second category is called Modifiable Off-The-Shelf (MOTS).

For simplicity in this thesis no distinction is made between the two different types of components. Therefore, both component types are referred to as COTS. However, in the literature often the term Off-The-Shelf (OTS) is used as a single

[1] In this thesis all terms related to dependable and secure systems, i.e., all terms like fault, failure, dependability, fault-tolerance and the like, are understood as defined in [ALRL04].

notion that includes both types of reusable components when it is necessary to differentiate between the two component types.

According to [Wha05] there are several informal definitions for a component depending on the actual field of application. In the following two plausible definitions are stated that intuitively explain what software components are about.

Definition 1. *In programming and engineering disciplines, a component is an identifiable part of a larger program or construction. Usually, a component provides a particular function or group of related functions. In programming design, a system is divided into components that in turn are made up of modules. Component test means testing all related modules that form a component as a group to make sure they work together.*

Definition 2. *In object-oriented programming and distributed object technology, a component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application. Examples of a component include: a single button in a graphical user interface, a small interest calculator, an interface to a database manager. Components can be deployed on different servers in a network and communicate with each other for needed services. A component runs within a context called a container. Examples of containers include pages on a Web site, Web browsers, and word processors.*

Besides the two informal definitions stated above there is another textual definition by Szyperski [Szy97] that is most frequently used in scientific literature. This popular and concise definition for software components is stated in the following.

Definition 3. *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*

Despite the benefits the component-based development paradigm entails there are still several challenges and problems regarding component granularity, component variability, component dependencies, component interfaces, component models, and component certification that have to be taken into account [MPT04].

A very basic problem is that it cannot be guaranteed that several separately designed components work properly when they are combined in a single application [HC01].

A further research issue is dynamic component composition. That is, connecting binary components at runtime. Of course, problems concerning the integration of (maybe third party) components at runtime are highly relevant for fault-tolerance considerations. If two or more components do not work properly when integrated it is important to detect this situation as soon as possible to keep the system in a consistent state. After detection appropriate action has to be taken to minimize unintended system behavior.

To make use of components it is also necessary to define a so-called *component model*. According to [Mic05b] a component model can be defined as stated in the following definition.

Definition 4. *A component model is defined as typically providing these major types of services:*

- Component interface exposure and discovery. *Thus, during application use, one component can interrogate another one to discover its characteristics and how to communicate with it. This allows different companies (possibly independent service providers) to create components that can interoperate with the components of other companies without either having to know in advance exactly which components it will be working with.*

- Component properties. *This allows a component to make its characteristics publicly visible to other components.*

- Event handling. *This allows one component to identify to one or more other components that an event (such as a user pressing a button) has occurred so that the component can respond to it. In Sun's example, a component that provided a button user interface for a finance application would "raise" an event when the button was pressed, resulting in a graph-calculating component gaining control, formulating a graph, and displaying it to the user.*

- Persistence. *This allows the state of components to be preserved for later user sessions.*

- Application builder support. *A central idea of components is that they will not only be easy and flexible for deploying in a distributed network, but that developers can easily create new components and see the properties of existing ones.*

- Component packaging. *Since a component may comprise several files, such as icons and other graphical files, Sun's component model includes a facility for packaging the files in a single file format that can be easily administered and distributed. (Sun calls their component package a JAR (Java Archive) file format.)*

2.3 High-Level Software Development in Embedded Systems

There is an increasing trend towards high-level development in the embedded systems community. It is the aim of these mostly model-based approaches to raise the level of abstraction. Current programming languages are not able any more to cope with tremendously increased complexity in embedded software development. The systems themselves are getting more and more complex. Furthermore, close cooperation among different engineering domains is often necessary to bring an embedded application to the market. That is, many fields are involved in hardware and software development as most embedded devices are in tight interaction with their environment. Therefore, high-level development aims at simplifying the development process by providing a uniform means of communication throughout the different people involved. That is, a system model is used from the beginning of the development process on to the final validation tests. Figure 2.1 illustrates the intended process for model-based approaches as it is implemented in the Simulink development environment.

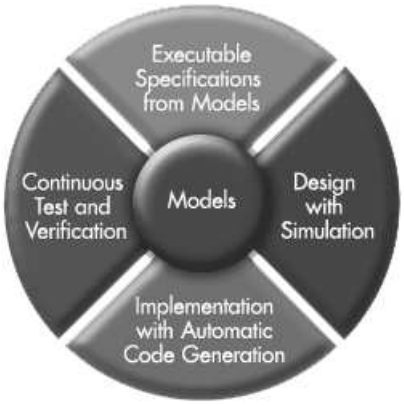

Figure 2.1: The model-based design process according to [The06].

The key idea is to pass models of appropriate levels of detail at the interfaces of different development phases. Models are a better choice than paper work as practiced up to now because models can be kept executable and they can easily be visualized. Figure 2.2 compares the model-based process to the conventional development process. Visualization is an important feature of model-based development because it allows developers to better oversee the complexity inherent in typical embedded system designs.

In the past few years a number of different approaches to the model-based development paradigm have evolved in the literature but also in the industry. Several companies are providing software development environments that support model-based processes (e.g., The Mathworks Matlab/Simulink [The06], National Instruments LabVIEW [Nat06], or Telelogic TAU [Tel06]). Also several research projects are focussing on the use of model-based principles to improve development time, system reliability and the maintenance phase. Interesting projects that are somewhat related to the work presented in this thesis are presented in Section 2.5.4.

2.4 Fault-Tolerance in Distributed Embedded Systems

The increasing complexity of embedded applications results also in an increased number of failures and, therefore, lower customer satisfaction and higher maintenance costs. As a result means for mitigating failure effects are often included in embedded designs. In a network of smart cameras it is especially necessary to improve autonomous fault handling because one of the key aims of smart cameras is to minimize operator interaction.

Nevertheless, the market demands also embedded smart cameras to obey tight device cost constraints. It is, therefore, not practical to include redundant hardware or extra reliable, i.e., expensive, hardware. Another restriction is that extensive software redundancy can also not be implemented because the high computing power requirements of embedded video processing have to be met. Redundant software components, as used in general-purpose computing middleware (e.g. FT-CORBA [Gro04]), would require additional hardware which is not acceptable due to limited

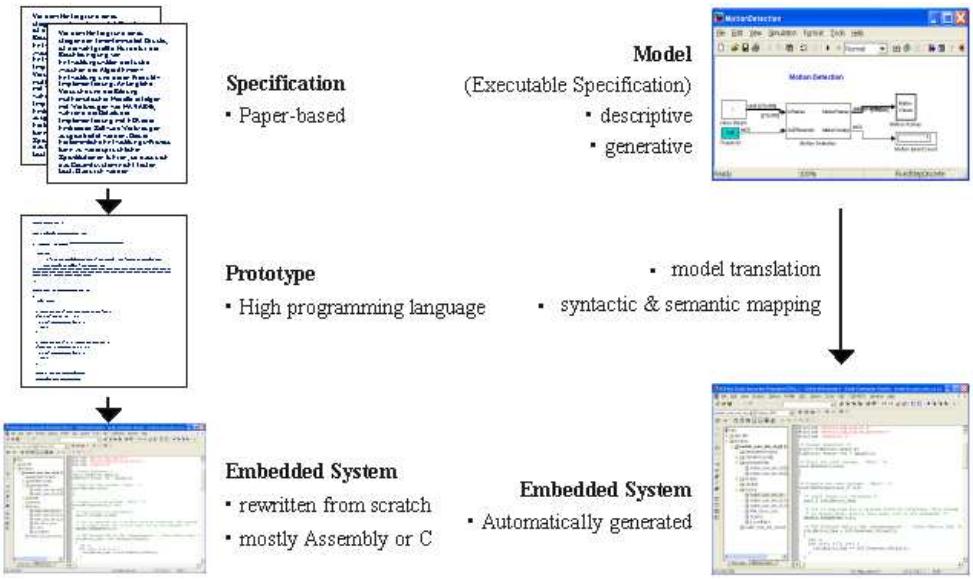

Figure 2.2: The model-based development process compared to conventional development [The06].

allowed unit cost. Unit cost limits also inhibit extensive software fault-tolerance techniques [Pul01] as N-version programming, recovery blocks, time redundancy, and other high-reliability techniques.

However, given the dynamic reconfiguration capabilities of our ***Smart****Cam* it is possible to introduce limited fault-tolerance by employing graceful degradation of executed services. Fault-tolerance in general can be implemented in the application code or in middleware. Nevertheless, it is more efficient to provide it in the middleware because by that not every application has to include fault-tolerance code by itself [SNT04]. Because of the improved code efficiency and to better reflect the goal of taking the ***Smart****Cam* towards more autonomous behavior in our system the graceful degradation mechanisms are provided by middleware services. In that respect fault-tolerance is introduced almost transparent to application developers which eases software development. Framework developers and system integrators are then responsible to provide the intended middleware mechanisms for fault-tolerance. The publisher-subscriber framework together with the dynamic reconfiguration facilities serve as the foundation for realizing these services.

It is important in distributed real-time systems to distinguish between *application-level* fault-tolerance and *system-level* fault-tolerance [HLKK00]. Given this differentiation the available approaches can be categorized. System-level fault-tolerance encompasses redundancy and recovery actions within the system hardware and software. Application-level fault-tolerance, on the other hand, encompasses redundancy and recovery actions within the application software.

Based on these definitions a middleware approach to fault-tolerance would qualify for both of the above-mentioned or for none. However, as middleware fault-tolerance aims at simplifying development by abstracting fault-tolerance

mechanisms as well as to remove fault-tolerance code from applications it may be useful to consider it a system-level technique.

2.5 Related Work

This section presents a discussion of the state-of-the-art in research fields relevant for this thesis. Important research results and related work are summarized. The section is organized in several subsections that reflect the different topics related to this work.

2.5.1 Autonomous and Self-Adaptive Systems

In the late nineteen-eighties the idea of *autonomic computing* emerged. Originally introduced by IBM research it spread into the artificial intelligence and control communities [Lad99]. Autonomic systems can be characterized as systems that can manage themselves given high-level objectives from administrators. They will maintain and adjust their operation in the face of changing components, workloads, demands, and external conditions and in the face of hardware or software failures, both, innocent and malicious [KC03].

According to Kephart et al. [KC03] autonomic computing can also quite well be described by the term *self-management*. As a relatively general term it subsumes the most important attributes of autonomic computing systems which can be identified as

- self-configuration,
- self-optimization,
- self-healing, and
- self-protection.

Therefore, also the notion of a *self-adaptive systems* can be seen as a synonym for autonomic systems. Traditionally, self-adaptation is achieved by techniques introduced by the artificial intelligence (AI) community. The aim has always been to improve robustness of mission critical systems. Another term describing the attributes of autonomous systems is *self-controlling system* [KBE99]. It was derived from the viewpoint of control engineering where not only the controller is implemented in software but also the system under control is a part of software. Then well-known principles like open-loop, closed-loop or adaptive control models can analogously be applied for self-controlling software.

The model-based approach to self-adaptation [KS99] is also based on models. But different to self-controlling systems the model-based approach relies on domain-specific models typically based on graphs. Models represent the system itself, it's environment and the relationship between them. Once these models are available they are used to derive actions for reconfiguration in order to adapt the system to achieve a certain goal.

Quite early NASA introduced self-adaptive techniques in a large-scale project called "Remote Agent" (RA) that was deployed to space with the *deep space one* mission [BDR+99]. This mission was the first to rely on autonomous navigation. Previous missions always were navigated by ground operators. As a novel flight command and control system the RA was directed by goals specified by the operators. It was the RA's responsibility to choose appropriate means to achieve them. To meet these requirements AI technologies for planning/scheduling and model-based fault diagnosis and recovery were employed. To handle the complexity in software development resulting from the AI technologies the designers introduced model-checking and automatic code generation. Especially, the fault protection code was automatically generated from behavioral and structural high-level statechart models to improve development efficiency and code quality.

Another interesting approach towards self-adaptive software is *architecture-based self-adaptation* [OGT+99]. This approach concentrates on an architectural perspective of adaptive software. It assumes relatively coarse-grained components as building blocks of an application. Self-adaptation is then organized in two concurrent processes: *system evolution* and *system adaptation*. System evolution focuses on the consistent application of change over time whereas system adaptation represents the cycle of detecting changing circumstances and planning and deploying responsive modifications. Figure 2.3 sketches the basic relationship between adaptation and evolution. According to the authors the main issue in self-adaptation is to guarantee consistent reconfiguration of components. Therefore, robust observers together with a rigorous change management are vital.

2.5.2 Middleware and Frameworks for Embedded Systems

CORBA-based general purpose middleware Middleware for distributed and embedded systems is a very active research field. A lot of work has been done to support transparent communication and to ease distributed application development. Component-based middleware technologies from general purpose computing, such as, Microsoft DCOM [Ses97], Java RMI [PM01] and OMG CORBA [Pop98] are not suitable for very resource limited devices [MCE02]. To adapt the CORBA technology to resource constrained real-time systems the Real-Time CORBA (RT-CORBA) and Minimum CORBA specifications [The01, Obj02] have been introduced.

Schmidt et al. [Sch02] invented "TAO" as an implementation of the RT-CORBA specification. It is an object request broker especially developed for distributed real-time and embedded systems. Their *CIAO* framework [BWGS03] extends TAO to also include a component model for distributed real-time and embedded systems that enables easy component composition. All these approaches are quite large and, therefore, not suitable for our multi-DSP platform. They are further not available on the operating system of our DSPs and cannot easily be ported to it.

In general all these approaches share the idea of providing transparent communication among object or components residing in different address spaces. The problem is that they also aim at supporting a wide range of programming languages and mostly general purpose computer architectures. That is the reason why

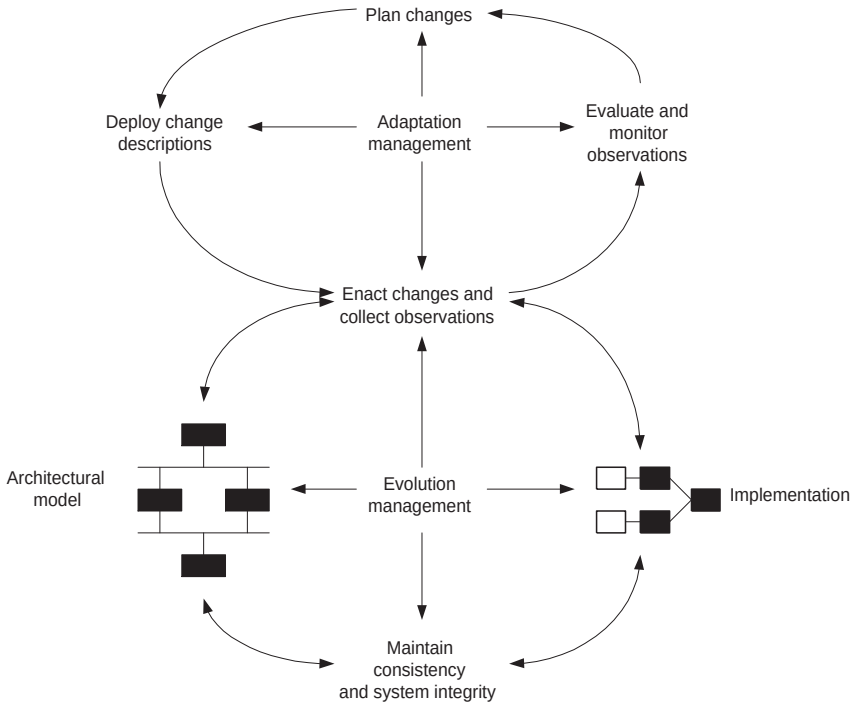

Figure 2.3: Architecture-based approach to self-adaptation adapted from [OGT+99].

these middleware systems impose substantial overhead. What they provide rather well is software reuse and platform independence. But as these advantages cannot be exploited in the highly specialized ***Smart****Cam* platform a very light-weight approach was chosen and is presented in this thesis.

Supporting Adaptable Distributed Systems with FORMAware A Java-based framework for adaptable systems is FORMAware [MBC04]. It is targeted for mobile platforms and ubiquitous applications. But FORMAware is also strongly dependent on general purpose operating systems. Therefore, it is not suitable for deployment on the DSPs of the ***Smart****Cam* hosting the proprietary DSP/BIOS operating system.

FORMAware is also an approach that focusses on the development phase of embedded software and facilitates reuse by introducing a component-based framework. Providing reflection mechanisms on the architectural and structural level the FORMAware middleware is an adaptive framework. That is, applications can be rather easily adapted to new application areas and extended with new functionality. Basically, the middleware keeps a meta-model of its current composition. By using indirection through adapters, delegates and factories components can be changed without halting dependent services. This is a functionality that is also key in the approach presented in this work. However, performance analysis sug-

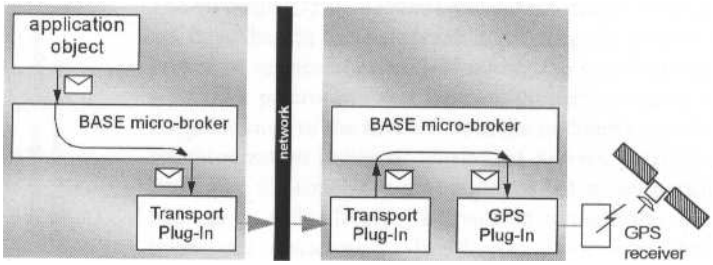

Figure 2.4: Accessing remote services or device capabilities in BASE adapted from [BSGR03].

gest that the Java implementation and the additional indirection impose substantial performance overhead.

A Micro-broker-based Middleware for Pervasive Computing In [BSGR03] the authors present their *BASE* middleware for pervasive computing. This work aims at a scalable and efficient middleware that serves all possible computing architectures for pervasive computing.

BASE is based on a micro-broker that only implements very basic functionality. All other features can be added as plug-ins as needed. Especially, transport protocols are added as plug-ins. By this technology it is easy to adapt the middleware to new protocols and communication devices.

In this thesis a very similar approach is presented where a medium abstraction entity takes care of transparent communication. This abstraction object is also easily extended to handle new communication media. The micro-broker is quite analogous to micro kernels known from operating system technology.

Although the BASE middleware was implemented in Java it is very memory efficient due to the micro-broker approach that focusses only on the most important middleware features. However, transport of remote invocations is realized through the Java RMI interface. Therefore, it suffers from substantial performance overhead for remote service invocations. However, local invocations can even be faster than standard RMI calls because they use a proprietary proxy solution. It exploits locality so that unlike RMI it does not need to pass through the RMI and TCP protocol stacks. Again, this is similar to the approach in this work where proxy objects are only used when communicating with remote services. Figure 2.4 illustrates the basic principle of service invocation in BASE.

A Data-Flow Oriented Component Framework for Pervasive Dependability Another interesting approach is the *Self-** (pronounce self-star) architecture [FH03]. It is a data-flow oriented and component-based middleware framework that is aimed at dependable pervasive computing systems. The architecture defines *components* and *pins* where pins are directed ports that connect components. That is, data flows through pins from one component to the other. The data-flow oriented approach has the advantage of clean separation of components. This is especially important because Self-* was designed to be a demonstration platform for illus-

trating dependability mechanisms based on automated fault injection. Mainly interface functions of components are tested for standardized hypotheses by online fault injection.

For example, a hypothesis may state that a function crashes if its first argument is negative. The automated fault injection subsystem tests the function and if the hypothesis cannot be rejected by the test the middleware can provide countermeasures such as a wrapper for the function that ensures a non-negative first argument. By this mechanism the authors want to increase overall system dependability. Of course, this approach involves a lot of overhead in storing hypotheses, conducting rejection tests by fault injection, storing all data of component communication to derive hypotheses and so on.

Although this approach is very promising for devising future dependability mechanisms it is not very suitable for use in real video surveillance applications. The approach presented in this thesis also builds on a data-flow oriented view of component communication. In contrast to explicit components for splitting and queueing data streams in our approach the middleware objects take care of multiple data receivers and buffered communication.

Concerning the dependability measures based on automatic fault injection we have also taken a different approach to increasing fault tolerance. This is mostly due to the fact that the runtime overhead of online tests and the storage needs for all relevant data is not feasible for the embedded video surveillance application of the ***Smart****Cam*.

The Publisher-Subscriber Paradigm A popular inter process communication model for embedded systems is the real-time publisher/subscriber model (RT-PS) [RGS95]. It supports loose coupling of tasks by message-oriented communication. As the registration of data sources and sinks can be done at runtime the RT-PS approach was chosen as the basis for the software framework presented in this thesis.

One promising commercial implementation of the RT-PS model for data centric communication, i.e., NDDS [PCSH99], is available from Real-Time Innovations (RTI). Unfortunately, this implementation supports only special operating systems and it is not available for the DSPs we use in our smart cameras. Furthermore, it cannot be easily applied to different computing architectures. Therefore, it is not well suited for the heterogeneous processor environment of the ***Smart****Cam*. Nevertheless, the basic principles of the NDDS approach by RTI are the same as in the middleware part of the framework presented in this thesis.

MicroQoSCORBA (MQC) One of the key features of MQC [Dor03] is a set of fault-tolerant mechanisms that allow for supporting applications which demand for a higher level of reliability. It is a middleware platform that focusses on embedded applications. A fine level of configurability allows developers to generate a customized middleware. MQC is, therefore, a framework that for instantiating a middleware customized to specialized hardware. It addresses advanced issues like fault tolerance, security, energy consumption and system performance. Fault-tolerance mechanisms incorporated into MQC include temporal redundancy (i.e.

automatic retransmissions), spatial redundancy (e.g. multiple transmission paths), value redundancy (e.g. checksums), as well as group communication and failure detection.

Generic Object Platform Infrastructure An early middleware architecture specially tailored for multimedia applications was supposed by Coulson [Cou99]. It is called *Generic Object Platform Infrastructure* (GOPI) and it strongly focusses on scheduling and threading mechanisms supporting multimedia streams. Based on conventional operating systems it does not rely on hard determinism as is the case with the framework presented in this work. Being a multimedia middleware solution one of its key tasks is to provide communication between multimedia applications on different network nodes. Unlike the approach presented herein GOPI relies on standard CORBA ORBs and direct socket communication. Although GOPI features QoS descriptions it does not provide dynamic reconfiguration capabilities.

Distributed SW Architecture for ubiquitous sensor systems Wolf et al. [LWD+06] present a software framework for ubiquitous smart cameras. It is a joint effort of the Princeton's smart camera group and the Vanderbilt University's *Model-Integrated Computing* (MIC) group. Their focus is on the modeling and design of real-time embedded camera systems.

Based on their gesture recognition system prototype they investigate a fully distributed communication pattern to support intelligent and ubiquitous applications using several cameras. As the heart of the system a multi-layer software framework provides a service-oriented platform for different algorithms.

Similar to the work presented in this thesis they have developed a middleware layer that orchestrates the algorithms comprising the actual application. The goal is to run at any time only services that are actually needed. Again this is very similar to our approach were the algorithm composition and the algorithm's QoS levels are adjusted according to actual application needs.

They also use service registration and binding to dynamically adjust the composition of different services. Decisions concerning which service configuration to choose are made using finite state machines that are application-specific. For the design and the modeling of the software they take advantage of the model-integrated computing paradigm. Domain-specific languages are used as a framework for developing the software for their embedded cameras.

In this thesis also a model-based development approach is suggested. Unfortunately, up to now they did not disclose enough details to do an in-depth comparison of their work with the ideas presented in this thesis. Currently, they are mainly focussing on the wireless communication performance and different network topologies for their camera networks.

Texas Instruments DaVinci Technology The *DaVinci* technology by Texas Instruments (TI) [Mod06] is an innovative framework for multi-core embedded DSP solutions. DaVinci is a combined hardware and software platform that builds upon a dual-core System-on-Chip (SoC). An ARM core and a C64x core are integrated in a

single chip. The intended applications are multimedia appliances that rely strongly on complex signal processing algorithms. Extending previous architectures TI provides a complete software bundle to ease application development. On the one hand there are the two operating systems, i.e., Linux for the ARM and DSP/BIOS for the C64x, along with different support libraries. On the other hand there is an abstraction to aid developers in using third-party components easily.

Similar to our approach presented in previous work [BDM+06] signal processing algorithms are treated as components. The application developer can plug and unplug them using standardized interfaces. But in contrast to our approach they currently support only encoder and decoder algorithms. Based on their XDAIS [Ins02] component standard they extended it to XDAIS-DM or XDM to also support algorithm descriptions that are needed for proper composition of multimedia algorithms. Mainly this information is dedicated to different QoS settings as resolution, frame rate and the like.

In contrast to that the algorithm description interface presented in this thesis is more flexible and is not limited to encoder and decoder tasks. Another difference to the presented approach is that the XDAIS-DM framework focusses on a single SoC. Indeed it handles two different cores but it does not address distributed nodes and communication among different algorithms residing in different address spaces as does the framework presented in this work.

2.5.3 Component Models and Technology

As a key building block most middleware specifications define a component model that specifies interactions of components within a system. Typical features of component models are listed in Definition 4. There are is a number of different component models presented in the literature. In the following only the most relevant for this thesis are briefly described and compared to the approach proposed in this work.

Large-scale server component models One of the most well known component models is the *CORBA Component Model* (CCM) [Gro05a] that is specified for the CORBA specification in its third version. Other well known commercial component-based approaches include Sun *Enterprise Java Beans* (EJB) [DeM02] and Microsoft *.NET* [Mic05a]. These are full-featured component systems that are mostly used in the development of large-scale business applications. Typically, all these large-scale component systems implement all features defining a component model (see Definition 4).

Because of the rich feature set they are also very large software systems that also impose substantial performance overhead. In embedded systems the focus is on light-weight solutions and, therefore, these major component systems along with their corresponding component models are not suitable in a typical embedded setting. To overcome the problems of excessive memory and computing power requirements *Light Weight CCM* (LwCCM) [ST03] was submitted to the Object Management Group (OMG) for specification.

LwCCM aims at providing only core features. Advanced functionality of the CCM is not included in LwCCM. Thus, it can be implemented for resource critical embedded systems. Embedded CORBA-based applications can, therefore, be realized using LwCCM. Persistence, transactions and security are not addressed in the LwCCM specification. Nevertheless, compatibility with the full-flagged CCM specification is retained so that LwCCM components can also be deployed on CCM-based systems.

The PECOS approach PECOS [WGC+02] is a collaborative project between industrial and research partners that seeks to enable component-based technology for a certain class of embedded systems known as "field devices". A component model for field device software is devised and a special component composition language called *CoCo* is introduced to ease software design. They also defined a mapping of CoCo to Java and C++ which not only allows to specify interfaces but also provides some help for specification of the actual behavior of components. This code generation raises the level of abstraction in embedded software development.

Unlike the approach suggested in this thesis the PECOS approach uses only textual representations for code generation. It is the aim of the PECOS project to support all necessary steps for embedded component development. Different tools have been developed and integrated into the *Eclipse* development environment. One goal is to transform this collection of experimental tools to a full suite of a commercial development environment for embedded software development.

As PECOS focusses on small devices, especially in the control domain it does not address dynamic reconfiguration of components. The central idea is to foster software development and software reuse by employing component-based principles to the embedded world. An idea similar to the approach presented in this work is to provide checking of component composition. However, in PECOS the checks are performed only at design time, whereas we are employing runtime checks to ensure proper dynamic component composition.

SaveCCM—A component model for safety-critical real-time systems SaveCCM [HÅCT04] is a specialized component model aimed at safety-critical control applications in vehicular systems. It is only of limited flexibility but, on the other hand, facilitates analysis of real-time and dependability issues in embedded control systems. As part of an overall effort to improve dependability in vehicular system SaveCCM is also accompanied by a dedicated component framework to also improve development processes.

Note that the term SaveCCM has nothing to do with the CORBA component model (CCM). It is merely a composition of the project name *SAVE*, the framework *SaveComp* and the general term component model and might be stated as *SaveComp Component Model*. Based on a pipes and filters paradigm the execution model of SaveCCM is rather restrictive. Components as the basic unit of encapsulation can be in either state, executing or waiting to be triggered, respectively.

The component model defines three other entities besides a component. First, there are *switches* that are used to dynamically change component interconnections. Second, *assemblies* are a means for forming aggregate components. As the third part the *runtime framework* provides services like component communication, component execution and control of sensors and actuators.

An interesting facet of SaveCCM is that a dedicated specification and composition language is used to represent a system in terms of the basic elements components, assemblies and switches. A graphical notation based on UML is also devised to have a symbolic representation of systems built with SaveCCM. As a demonstration example an adaptive cruise control application is used. It is shown that sacrificing flexibility in favor of facilitating real-time analysis helps in building reliable vehicular applications.

An Efficient Component Model for the Construction of Adaptive Middleware In [CBCP01] the authors present *OpenCOM* which is a light-weight component model based on the standardized COM component model [DeM95]. To be efficient it only supports a subset of the overall COM specification. That is, only a single address space is supported.

This is in contrast with the approach described in this work where the benefit is that component interaction is supported beyond address space boundaries in a transparent way. Furthermore, OpenCOM does not implement standard component model features such as distribution, persistence, security, and transactions. But the OpenCOM model is designed for dynamic reconfiguration of components which is in contrast to most standard component models that do not very well support the deployment phase of components in a dynamic application environment.

The interesting thing with OpenCOM is that it is designed as a component model for the design of middleware platforms itself. That is, it is not used to provide a structure for component interaction on top of a framework to form applications but to develop the framework. As a demonstration of its capabilities it was used to design *OpenORB v2* which is an adaptive middleware platform. Using OpenCOM the authors where able to introduce reflection to the middleware layer and, therefore, make the middleware itself adaptive.

At its heart the OpenORB builds also on GOPI [Cou99] which is a CORBA-based multimedia middleware platform. However, OpenCOM and its derivative technologies are designed for general purpose computing platforms rather than embedded architectures.

AFT-CCM—Adaptive Fault-Tolerance on the CORBA Component Model AFT-CCM [FSF03] is a component model based on CCM. It is aimed at applications with fault-tolerance requirements. Like most CORBA-based technologies it is also designed for large-scale distributed computing systems mostly applied in Web applications. The application programmer can specify QoS requirements for services and the desired levels of dependability can also be defined.

To achieve a special dependability level different forms of component, i.e., service, replication are employed. Several dedicated system components are respon-

sible for the transparent replication of application components. Furthermore, key system components are also replicated on different hosts in the system to guarantee correct replication also in case of failures in the runtime environment supporting the component model. Of course, it is also possible to integrate components into the system that are not critical and, therefore, do not need to be replicated. Persistence of component state information is achieved by constantly saving it to local non-volatile storage. Hence, on failure of a component its state is restored to a replica to continue normal operation after minimum downtime.

Given the significant overhead of the overall management framework and the full redundancy of replication it is understandable that each host in such a system has to provide substantial hardware resources. Therefore, AFT-CCM is not suitable for cost-sensitive embedded applications.

A Novel Component Platform for Logistics Software Product Lines An interesting approach for a component model in the area of logistics software is presented by Teiniker [Tei05]. The author introduces this reference component model in a formal notation based on graph theory. In extension to current server component technologies, e.g., COM+, EJB, or CCM, this approach introduces nested component composition and contract aware interfaces. That is, component interfaces are not only described in their syntactical properties but also in formally specified semantic behavior.

These semantic descriptions as defined in interface contracts consist of preconditions, postconditions and invariants. It is interesting to note that this novel component model decouples components from the communication middleware. Therefore, the decision of which middleware technology to choose can be separated from component development. This is in contrast to the approach presented in this thesis where a special middleware layer is obligatory to deploy algorithm components.

But similar to our approach the reference model of Teiniker et al. also supports local component communication [TMK+02] to boost performance. That is, components are connected directly if they reside in the same address space. In other component middleware it is usual that communication is always carried out using the whole communication stack of the used middleware technology.

Teiniker et al. also emphasized the aspect of reusing legacy code in the proposed reference model. Therefore, they provide a whole framework for generating compositional code and interfaces corresponding to the reference component model. Based on the suggested reference component model the author also derived a new hybrid development process especially suited for the intended application area of logistics software. Figure 2.5 presents an overview of the resulting hybrid development process.

Note that the system development step is also a model-based approach that raises the level of abstraction so that the developer's understanding of the system and the degree of reuse can be increased. In this aspect it is similar to the ideas for software development for embedded smart cameras presented in this thesis.

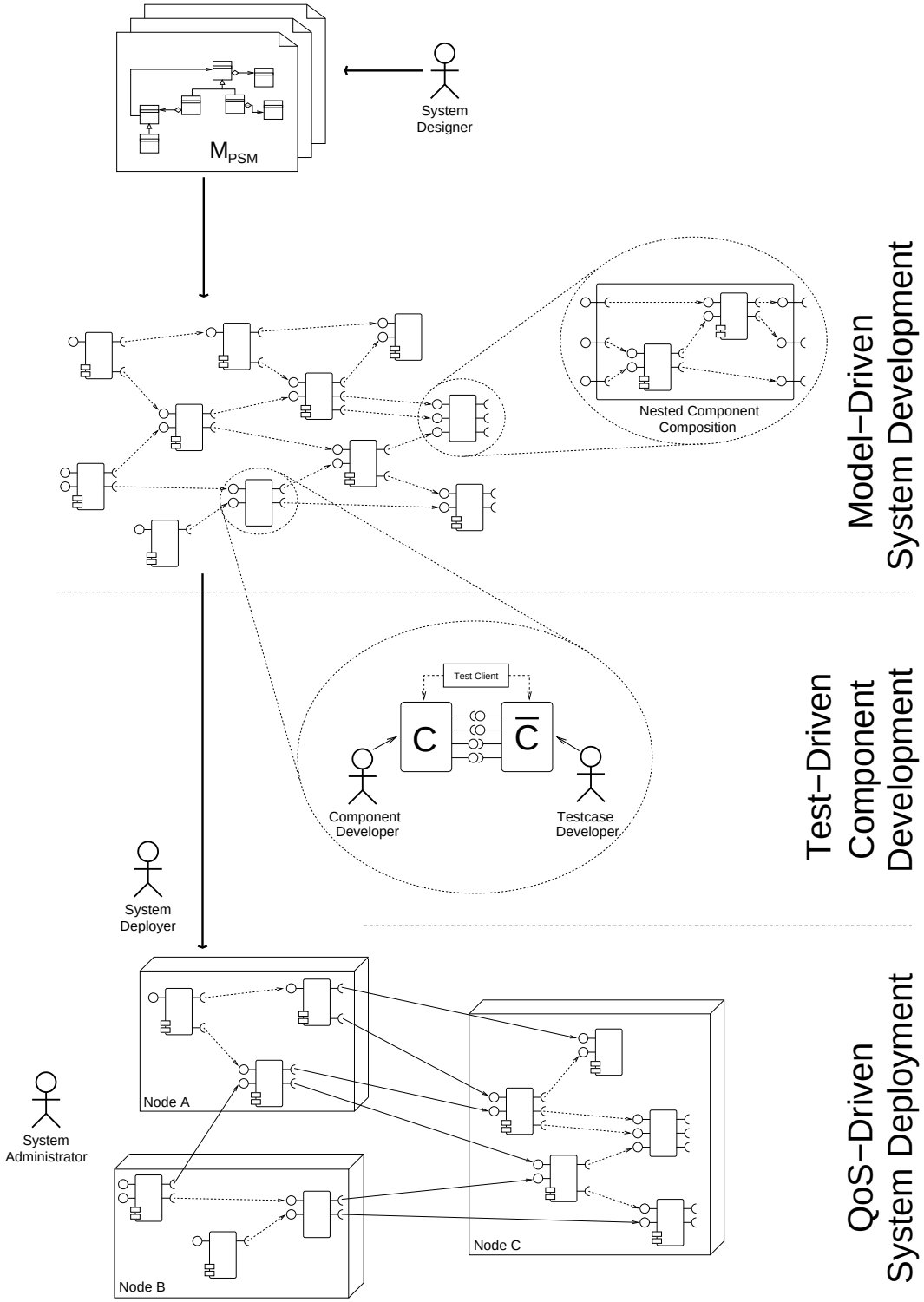

Figure 2.5: The hybrid development process according to [Tei05].

2.5.4 High-Level Software Development for Embedded Systems

There are a number of approaches to high-level or model-based embedded software development. For example platform-based development [SV02] suggests a layered design with well defined interfaces between different layers. It does not only cover software parts of a system but also the system hardware. As a very basic concept platform-based development is mostly used in real-time systems in critical applications. Principles of platform-based development are incorporated in most development processes to simplify system understanding. Therefore, also the approach presented in this thesis can be seen in terms of a system formed by different platforms where lower levels provide services for higher levels. Especially, the software framework of the ***Smart****Cam* is an example for a platform-based design (cf. Figure 3.2).

Other popular methodologies are model-driven architecture (MDA) [Gro05b] and UML-based methods [SM04]. All of these approaches raise the level of abstraction and employ code synthesis which is also a key benefit of modeling in Simulink as suggested in this thesis. Nevertheless, they are mostly focussing on automatic synthesis of skeleton code for the connection of distributed objects and components. It eases the development by relieving the programmer from tedious network programming and the like. In the approach presented in this thesis the focus is on algorithmic code. That is, besides skeleton code for component composition the application logic is generated from the model. In that way algorithm design is decoupled from framework design.

In Wybo and Putti [WP99] the synthesis capabilities of Simulink are evaluated for automotive powertrain control. Although most automotive applications have to meet hard real-time deadlines their overall performance requirements are significantly smaller than in video analysis. Similarly, an integrated modeling approach for audio signal processing using Simulink and Texas Instruments DSPs is discussed by Hong et al. [HGC+00].

Whalen and Heimdahl [WH99] discuss requirements and problems of automatic code generators for safety-critical systems. Major problems are model integrity, unambiguous syntax, and syntactic expressiveness. For automatic synthesis it is important that a model's expressions are unique so that code generation is a unique mapping. To be useful in the overall development process models have to be transformed to different levels of abstraction and to different domains representing different development phases. An initial model that is created in the requirements phase has to be passed to design people that are likely to transform it to a more detailed model that they pass to the next phase, respectively. All transformations have also to be such that there is a one-to-one mapping from the first model up to the code generated from the final model. Another issue is that a model has to be expressive enough to represent all entities and behaviors of the desired application domain.

A special code generation environment for safety-critical systems is described in Kim and Lee [KL03].

The model-integrated computing initiative is also closely related to the model-based development efforts undertaken in academia. It is described in more detail in the section on autonomous systems (cf. Section 2.5.1).

2.5.5 Software Fault Tolerance in Distributed Embedded Systems

There is a lot of work around concerning fault tolerance in distributed systems. Especially, middleware fault tolerance mechanisms for general purpose computing are well researched [SNT04]. But for highly resource constrained distributed embedded systems there is less work available in the research community.

Since fault tolerance can be integrated into different layers, i.e., into the hardware, the middleware or the application itself, a vast field of research is established. Fault tolerance is inherent to the massive redundancy of sensor networks. Unfortunately, we cannot afford hardware redundancy due to the tight cost constraints in the embedded market. The ambition of our work is, however, to integrate fault tolerance into the existing middleware and to increase service availability by benefiting from graceful degradation.

In [SS03] the improvement of resource utilization in fault-tolerant multi-resolution video servers is addressed. Similar to our work, for the achievement of graceful degradation this approach proposes a QoS degradation method by addressing the multi-resolution property of video streams.

Fault-tolerant CORBA (FT-CORBA), a popular fault tolerance extension of the *Common Object Request Broker Architecture* (CORBA), a standard for software componentry, is the foundation of many projects (see, e.g., [NGYS00], [GHN03], [SNT02]). Unlike in our work, the foundation of fault tolerance in FT-CORBA lies in redundancy, achieved by replication of objects in combination with logging and recovery.

The Aroma system presented in [NMMS00] enhances the Java Distributed Object Model with support for object replication. Aroma is suitable for existing and new Java RMI [PM01] applications as it can be deployed at runtime with only little modification to the Java RMI infrastructure. Phoenix [KKL04] is a fault-tolerant middleware layer for data intensive grid applications. Transient failures are handled according to user specified policy, a concept that we also consider to embed.

Although CORBA and also Java RMI are middleware technologies suited for distributed applications, they still target general purpose computing and are therefore too weighty for our objectives. Hence, we rely on the development of a self-contained solution to meet the special demands of the already existing system.

A Framework for Dynamic Software Architecture-based Self-healing In [QXcMw05] the authors present a reconfiguration scheme based on architectural reflection. That is, a model of the software architecture is maintained to reflect the current state of its structure. In case of monitored anomalies a dedicated architecture manager analyzes the changes and chooses a repair strategy.

A verification step ensures that only architecture reconfigurations following the system constraints are considered. After that a reconfiguration script is generated and executed to take the reconfiguration in place. The interesting part is that only

structural changes are taken into account. No change of behavior can be considered in repair strategies. But this approach is able to do the reconfiguration at runtime.

Fault Adaptive Embedded Software for Large-Scale Real-Time Systems In [MJO+05] the authors present a large-scale fault adaptive embedded software prototype for the Fermilab BTeV high energy physics experiment. Given the vast amount of data that has to be handled a multi-layer architecture is implemented comprising about 2500 DSPs for data pre-filtering and about 2500 Linux workstations for further data analysis.

Resulting from the large number of hardware elements a centralized expert system providing fault mitigation schemes for all possible system states is not feasible. Therefore, a distributed and multi-layered fault mitigation scheme is proposed. Fault handling is done at three different layers.

First, on each DSP there is a self-protecting and self-optimizing very light-weight agent monitoring the physics application on that DSP. Second, multiple DSPs are collected in so-called *Farmlets* which also have a dedicated fault handling authority. Third, several Farmlets are combined to *Regions* that themselves can impose fault handling strategies on their wider scope. In general only very rudimentary fault handling is provided. That is mainly due to the extreme data rates that have to be handled. The most important strategies are:

- Resetting the physics application on a single DSP when it fails to meet a time-out deadline.
- Setting a Farmlet-wide rate for dropping events without analysis on all DSPs in that Farmlet to prevent queue overflow.
- A global rate for dropping events for all Farmlets.
- Authorization of a Region to declare a Farmlet as defect and redirect future work to a hot spare Farmlet.

Real-Time Object-Oriented Adaptive Fault Tolerance Support (ROAFTS) Middleware In [Kim01] the author presents a discussion of the ROAFTS middleware for real-time fault tolerance for object-oriented distributed computing systems. The paper is not focussed on embedded systems but treats conventional distributed systems like web servers. It merely concentrates on real-time requirements of fault-tolerance actions itself. There are two claims the presented approach has to fulfill:

- Real-time computations have to be completed on time even when faults occur.
- Fault detection techniques have to yield a bounded detection latency and recovery techniques have to yield bounded recovery time.

In the approach presented in this thesis these two metrics are treated in a somewhat relaxed form. Given the soft real-time requirements and relatively low availability demands in video surveillance algorithms it is sufficient to concentrate on best-effort techniques based on graceful degradation.

ROAFTS builds on extensive system monitoring to detect resource failures and transient overloads. Similar to our approach its aim is to maintain the minimum required QoS of critical components as long as possible. Unlike the approach presented in this thesis ROAFTS relies on component replication following the distributed recovery block scheme (DRB).

To save the limited resources of a ***Smart****Cam* we do not use replication but only application-specific state redundancy.

A Framework for Scalable Analysis and Design of System-wide Graceful Degradation in Distributed Embedded Systems Shelton et al. [SKN03] take an interesting approach to defining graceful degradation of a system by introducing a so-called *utility metric*.

The basic assumption of the principle of graceful degradation is that a system can be defined as working with less than all functionality available. That is, there have to be some less important functionality that can be shut down or reduced to lower QoS in favor of more important behavior.

The utility is a measure for the benefit that is gained from a system. It may be related to utility attributes such as functionality, performance or reliability and safety. For each component of a system the designer has to assign a value for each of the chosen utility attributes. All utility attributes of a component together form the utility vector of this component. Overall system utility serves then as a metric to compare all possible system configurations. A decision on which configuration to take is deferred by comparing resulting system utilities.

Middleware for Embedded Adaptive Dependability (MEAD) The authors in [BN03] assume that software faults cannot be entirely eliminated and suggest to deal with them at runtime. Real-time systems for avionics and other mission-critical applications typically face constraints that limit hardware redundancy.

Nevertheless, dependable software is a must in such systems. In their opinion CORBA will be mostly used for next generation combat systems. However, they state that the lack of fault-tolerant real-time CORBA solutions forces developers to implement ad-hoc solutions in the application layer. MEAD is designed to transfer these solutions to the middleware layer and relieve applications from repeatedly included fault-tolerance code. Only policies governing the middleware mechanisms are provided by the application developer.

At its heart MEAD relies on component replication that is done transparently and using all available processors in the system. Based on a specified fault-tolerance properties of a component it is replicated by the framework at runtime. A monitoring entity generates fault reports that trigger a replication manager to switch between the results of replicas and re-replicate affected components.

The monitoring mechanism detects crashed components and processes. Similar to the approach presented in this thesis a hierarchical resource management continually collects current resource usage of each component. Resource overloading can thus be avoided.

An interesting feature of the approach is that they try to predict faults to apply anticipatory and preventive recovery. Basically, they use statistical and heuristic methods to compute the time span to future faults. Then they can reconfigure the system within that time span to make it less vulnerable to that fault.

For example, if it is likely that a processor will fail within five minutes the dependability framework can move the most important replicas from that processor to another. This minimizes the influence of the fault that may occur, of course, only to some level of confidence.

Fault-Tolerant Distributed Vision for Object Tracking In [KZSR01] a fault tolerant distributed vision system for smart room applications is described. It is based on the assumption that the aggregation of information from multiple viewpoints reduces uncertainty about a scene.

A consequence of this assumption is that there is redundancy in the sensory information and, therefore, there is no single point of failure regarding the sensors. This redundancy of aggregate information from multiple cameras is exploited to increase detection quality. The authors focus on a person tracking application and base their approach on distribution of time-stamped object features among cameras to improve the tracking performance.

Note that they assume a global time base for all cameras achieved by the network time protocol (NTP). Given a frame rate of 25 frames per second the authors prove that the resolution of NTP is sufficient.

Basically, every sensor node extracts object features from the observed scene and publishes it along with its time stamp to a resource manager. On a higher level of the architecture user agents can then take feature sets from all available resource managers to perform the actual vision application.

Resource managers are, therefore, entities to decouple the sensors from the user agents so that every agent can take advantage of information from all sensors. They demonstrated their approach in a smart room with four cameras of different view angle. Computation is done on standard workstations. Inaccurate localization of objects stemming from using only one camera could be corrected by using aggregated data from all cameras.

Chapter 3

A Software Framework for Autonomous Embedded Smart Cameras

3.1 SmartCam Hardware Platform Overview

The basis for the work presented in this thesis is the ***Smart****Cam* hardware platform [Bra05, BDM+06]. It is a highly flexible platform comprising a *sensing unit*, a *processing unit*, and a *communication unit*. A hardware architecture overview is depicted in Figure 3.1.

In the sensing unit the images are acquired by a 30 frames per second (fps) VGA CMOS sensor. The sensor is connected to one DSP via a FIFO memory to it's external memory interface to optimize image data throughput. Nevertheless, this direct connection to a single DSP poses several difficulties concerning image data distribution to different DSPs and, of course, fault tolerance considerations. Given the sensor's logarithmic characteristics it is able to eliminate substantial problems that CCD sensors are inherently suffering from. However, this logarithmic characteristic introduces substantial complexity of sensor control that have to be addressed by the software framework.

Actual image and video analysis is taking place in the processing unit. This central unit of the ***Smart****Cam* is designed to comprise up to ten DSPs for maximum computing performance. The number of DSPs can flexibly be adapted to the actual analysis performance requirements. Due to the characteristic processing steps in video analysis and the high computing power requirements high-end fixed-point DSPs are employed.

The communication unit's main part is the network processor that is at the same time the responsible unit for camera management tasks. Furthermore, the network processor also serves as the communication host for internal communication via the local PCI bus, as well as for external IP-based communication over wired and wireless connections.

In the software the inherent flexibility of the hardware design as well as the internal bus and external IP communication have to be reflected to fully exploit the

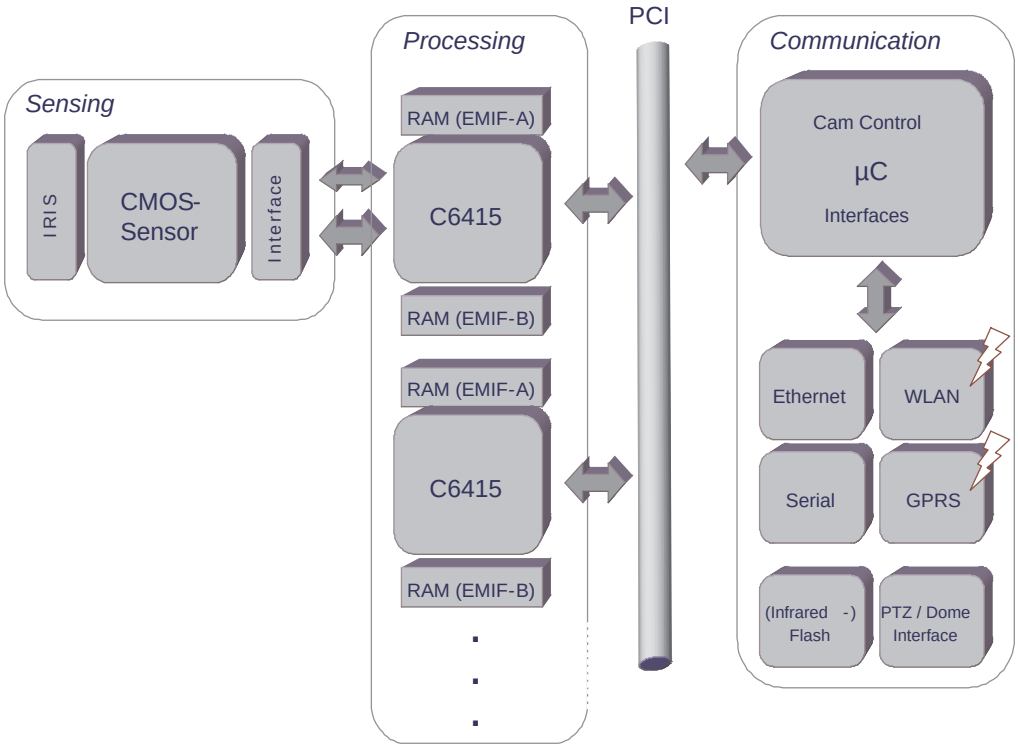

Figure 3.1: The ***Smart****Cam* hardware architecture. It comprises a sensing unit, a processing unit, and a communication unit. Up to ten DSPs provide the necessary computing power for video analysis algorithms.

architecture. Furthermore, the potentially many processors in the system have to be efficiently utilized. A trade-off between the number of running services, Quality-of-Service (QoS) and power consumption, as well as redundancy for fault-tolerance has to be found dynamically by the software framework.

3.2 Software Framework Requirements

Given the flexible hardware architecture described in Section 3.1 and the intended application domain of video surveillance several requirements are imposed on the software framework. The software framework has to meet them to fully exploit the flexible hardware architecture in order to provide the basis for innovative surveillance applications.

- Flexibility of algorithm configurations, i.e., how tasks are composed to build the application.
- Scalability concerning the number and the different types of employed surveillance tasks.
- Low resource consumption of the framework so that resources are spared for surveillance tasks and image buffers, i.e., for the application.
- Low performance overhead to allow real-time operation of surveillance tasks. At least frame rate requirements of all tasks have to be met.
- Provide means for detecting and mitigating faults in the system to reach some degree of autonomy and, therefore, aid operators and minimize maintenance effort.
- Provide appropriate interfaces and mechanisms that algorithms can be developed using a high-level model-based development process.

As the heart of the whole system the software framework is responsible to make smart cameras capable of innovative and increasingly complex video surveillance applications. By meeting the above requirements the presented architecture takes the ***Smart****Cam* another step towards intelligent video surveillance.

3.3 Software Framework Architecture

In the current software architecture the flexible allocation of algorithms onto different cameras in a surveillance network is realized by a mobile agent system (MAS). The MAS is hosted on the network processor and each video analysis algorithm is represented by a software agent that is able to migrate in the whole network of smart cameras. Actual image processing and video analysis, however, is performed on the DSPs. Note that the presented framework approach is not restricted to using a MAS. But the transparent agent migration and communication mechanisms make it an interesting solution for our prototype implementation. Another argument for

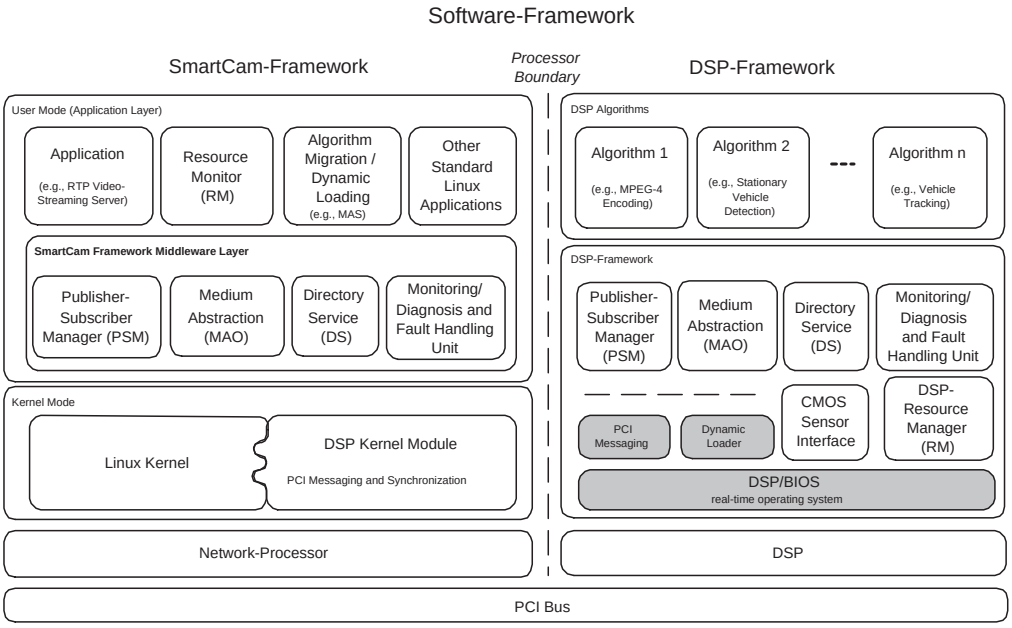

Figure 3.2: The overall software architecture of our smart camera. In the left part of the figure the so-called SmartCam-Framework is illustrated while the right part shows the so-called DSP-Framework.

a MAS is the platform-independence achieved by its Java implementation. The resulting performance penalty may be of interest in further research but is of little relevance to the work presented in this thesis.

To ease application development for this platform of heterogeneous processors an abstract programming model is used. The network processor as the central unit hosts the application logic. That is, the interconnections of agents representing the algorithms form the actual surveillance application. All DSPs on the camera are viewed only as computing power providers. Every algorithm executed on a DSP is represented by a special entity on the network processor. In other words every entity representing a video analysis algorithm carries a DSP binary that can be downloaded to a DSP so that it can perform the video processing there.

The software architecture of our smart camera is designed for flexibility and reconfigurability. It consists of several layers which can be grouped into (i) the *DSP-Framework* (DSP-FW), running on the DSPs, and (ii) the *SmartCam-Framework* (SC-FW), running on the network processor. This architecture is based on the abstraction that the application logic is running on the network processor and loads and unloads the actual analysis algorithms onto the DSPs as needed. An overview of the software architecture of our smart camera is depicted in Figure 3.2.

SmartCam Framework The SC-FW that is illustrated in the left part of Figure 3.2 serves two main purposes. First, it provides an abstraction of the DSPs to ensure platform independence of the application layer. Second, the application layer uses the provided communication methods, i.e., internal messaging to the DSPs and

external IP-based communication, to exchange information or offer data relay services for the DSP-FW. Modules of this part of the software architecture support application development in that they provide high-level interfaces to DSP algorithms and functions of the DSP-FW. Especially, the MAS system makes extensive use of these services to access the DSPs. To further ease application development the XScale processor is operated by LINUX. Thus, the SmartCam-Framework is running on top of a standard LINUX kernel.

DSP Framework This part of the software architecture, as indicated in the right part of Figure 3.2, runs on every DSP in the system. The main purposes of the DSP-Framework are (i) the abstraction of the hardware and communication channels, (ii) the support for dynamic loading and unloading of application tasks, and (iii) the management of on-chip and off-chip resources of the DSP. Of course, the sensor interface module is only needed on the DSP to which the image sensor is connected. The key functionality in the DSP-Framework is the publisher-subscriber middleware that is described in Section 3.4. These service management facilities are needed to allow algorithms on different DSPs to establish connections to each other dynamically. The DSP-Framework is built upon the DSP/BIOS operating system from Texas Instruments.

Dynamic Loading All video analysis algorithms and also some framework components can be loaded and unloaded at runtime by the *Dynamic Loader* module. Actually, only modules of the DSP-FW in dark shade in Figure 3.2 have to be available at startup. All other components can be dynamically loaded at runtime. Therefore, the framework and the application can easily be extended or adapted to dynamic changes in the system's environment if desired.

The dynamic loading facilities are also the basis for more sophisticated services like load distribution [Bra05], dynamic power management [Mai06], and graceful degradation to cope with faults.

3.4 The Publisher-Subscriber Middleware

The publisher-subscriber architecture is an integral part of the DSP-FW and the SC-FW. It aims at providing seamless and flexible connections between the algorithms running on the DSPs. Furthermore, it has to provide the basic means for supporting application reconfigurations aimed at reducing power consumption or realizing graceful degradation in case of failures.

From the framework's point of view every video analysis algorithm is a separate entity that is executed in its own thread. Interconnections of the algorithms are defined by the application. In previous work we used statically defined relations among different data services, i.e., algorithms, to simplify inter-task communication. This resulted in a very efficient message exchange over the PCI bus. However, the static bindings of data producers and consumers substantially restricted flexibility in dynamically combining algorithms. Furthermore, algorithms had to directly invoke PCI communication primitives which reduces portability.

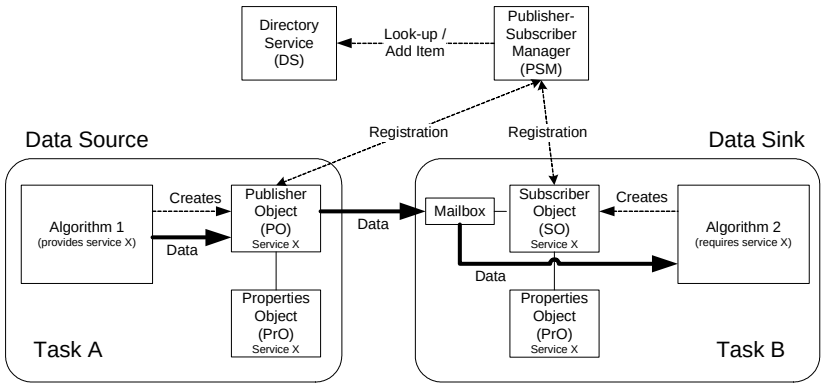

Figure 3.3: Fundamental relations between objects of the publisher-subscriber architecture. Only local connections within a single DSP are sketched.

To overcome these limitations a publisher-subscriber middleware layer (PS-MW) has been introduced. It provides the algorithms on the DSPs with basic message-oriented communication facilities that are transparent concerning the underlying transport medium. Additionally, a directory service was added to enable dynamic service discovery. It is important to mention that the major goal of our efforts was to provide these services with minimum overhead to save resources on the DSPs. Additional information on several details of the publisher-subscriber architecture described in the following can be found in [Tre06].

3.4.1 Architecture Description

As previously described applications for the ***Smart****Cam* are organized as different algorithms. These algorithms are interconnected depending on the data flow required by the surveillance application. Each algorithm is running in its own task. For communication between algorithms an operating system mechanism called mailbox is employed. Mailboxes provide buffered communication and also allow for synchronization as tasks can be blocked when they are waiting for data delivered by the mailbox.

In video applications a large amount of data has to be handled. To use the limited memory of the DSPs efficiently image data is not copied when sent between algorithms on the same DSP. Only references to actual data are exchanged. Small messages like system commands or monitored performance information are directly posted to mailboxes.

Figure 3.3 depicts the situation for two algorithms residing on the same DSP. The first algorithm provides a data service X that the second uses for further processing.

The core of our publisher-subscriber architecture is realized as an efficient object-oriented implementation. In the following the different objects of the PS-MW are briefly described.

Publisher-Subscriber Manager Object The *publisher-subscriber manager* (PSM) is the authority where algorithms can register as data providers or data consumers. That is, they register a publication or a subscription, respectively. There is one PSM running on each DSP and on the XScale. Registration is available through a simple interface. When an algorithm wants to register a service it first instantiates a publisher or subscriber object depending on whether a publication or subscription is needed. This object then registers itself with the PSM. In this process it is also assigned a unique number so that other algorithms can reference the service. The newly registered service is also added to the directory service where it can then be looked up based on its unique identification number or its properties. As algorithms can reside on different DSPs within a ***Smart****Cam* it is also necessary that each PSM can discover services that have registered with a different PSM. Therefore, the network processor also hosts a PSM that relays service requests between PSMs on different DSPs.

Properties Object *Properties objects* (PrO) are used to describe published data and subscriptions as well. Each publisher and subscriber object owns a PrO that identifies the details of provided and subscribed data services, respectively. Therefore, a PrO represents the Quality-of-Service (QoS) configuration of a data service. Algorithms derive their own PrO from an abstract class and add additional features as needed. Examples for basic properties include image resolution and frame rate. The MPEG-4 algorithm extends these by adding quantization level, bit rate and other algorithm-specific properties. In the service discovery process the PrOs are used to match subscribers to appropriate publishers by comparing their properties. By using a description in terms of properties it is possible to let an application object (algorithm) decide whether an available service meets its requirements or not. If there are several similar services available algorithms make their decision based on the information offered through PrOs. It is the responsibility of every algorithm to provide the necessary information for offered (data) services when the service is registered with the PSM. Typically, this is done during the initialization phase of an algorithm.

Publisher Object Every task that provides data services instantiates one *publisher object* (PO) for each message type it wants to publish to other tasks. On instantiation the PO then handles the registration with the PSM. Every publisher keeps a PrO that contains a description of the provided service. When data is ready for transmission from the algorithm the PO posts a reference to this data as a message to the mailboxes of all subscribers registered for this service. If there are subscribers residing on different DSPs an intermediate subscriber is used. This procedure is described in more detail in Section 3.4.2.

Subscriber Object A task that requires a data service of another algorithm instantiates a *subscriber object* (SO). The SO in turn registers with the PSM. In order to receive data a mailbox is created using operating system services. A mailbox is a buffered communication mechanism that blocks writing and reading tasks if the

buffer is full or no data is available, respectively. To define the required data quality each SO owns a PrO. In the registration process the PSM looks up the appropriate service using the directory service DS (cf. Section 3.4.3). If a fitting service, i.e., a PO with a matching PrO, is discovered then the discovered publisher stores a reference to the mailbox of the requesting SO. Messages are then transferred through this mailbox.

3.4.2 Medium Abstraction and Remote Subscription

In case of algorithms residing on different DSPs, i.e., a so-called *remote subscription*, an extension to the plain architecture described above is needed. A special object for abstracting from the communication medium is used to establish the connection. This *medium abstraction object* (MAO) is part of the middleware layer and is present on every processor of the platform. That is, a MAO is available on each DSP and the network processor (XScale). In general it is possible to use it for different communication media. But currently it is only used for providing abstract communication over the local PCI bus of the ***Smart****Cam*. Figure 3.4 illustrates the case of two algorithms residing on two different DSPs in more detail.

A remote subscription scenario is very similar to the single DSP case. It can be seen from Figure 3.4 that the situation on the involved DSPs is the same as it is in the single DSP case (cf. Figure 3.3). But now the MAO takes the role of the local SO and PO on the involved DSPs, respectively. That is, on the DSP with the data source (task A on DSP 1) the MAO instantiates a proxy SO and on the DSP with the data sink (task B on DSP 2) a proxy PO is created. These proxy objects behave like normal publishers and subscribers, respectively. They exchange data by means of posting messages to the SO mailboxes. As previously described, in case of large data, i.e., video frames, only references to local buffers are transferred. In contrast to that the MAO objects transfer the actual data through the medium they are bound to. That is the local PCI bus in this case.

As the PCI bus has limited transfer capacity it is necessary to limit the number of remote subscriptions so that real-time operation is possible. This is realized by the use of a dedicated resource manager (RM) that keeps record of available resources.

3.4.3 Directory Service and Service Discovery

For a convenient service discovery the DSP middleware, i.e., the DSP-FW, provides a *directory service* (DS) where all published services are listed together with their properties. Currently, the search algorithm of the DS uses only a simple description to find appropriate publishers for registering subscribers. That is, only a message type and important QoS parameters are used to choose the best matching data service. To support applications that need more control over the selection of publishers and subscribers, respectively, it is also possible that a list of similar services is returned. It is then the application's responsibility to choose one.

The DS is organized as a collection of simple lists because of the relatively small number of entries. Each entry has an identification number that is a system-wide

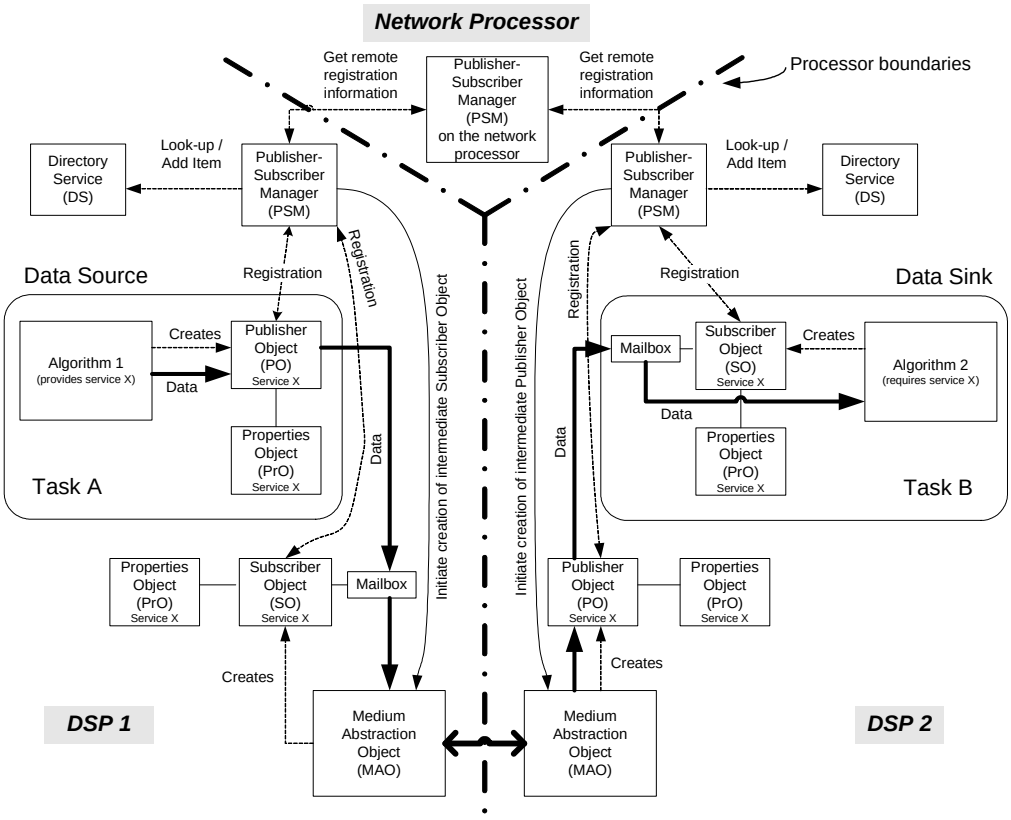

Figure 3.4: Extended publisher-subscriber architecture to connect algorithms running on different DSPs. These remote connections beyond DSP boundaries are established via intermediate publishers and subscribers. The special medium abstraction object (MAO) is used to abstract from PCI communication.

unique key identifying publishers and subscribers. These keys are created on instantiation of a publisher or subscriber.

If there is no matching PO or SO for a registering SO or PO, respectively, then a remote service discovery process is initiated by the local PSM. In a remote lookup the local PSM queries the PSM residing on the XScale that in turn keeps records of PSMs of all other DSPs. The PSMs use their associated directory services to look up the requested service. Therefore, all available services in the system are taken into account in this search.

In the future it will be possible to extend the DS to build on more abstract service descriptions to even better support QoS management. That is, to allow algorithms to choose an appropriate service by providing a textual description most likely provided in XML format. However, to use XML information a parser is needed. As we do not want to load the DSPs unnecessarily it can be imagined that the parsing is done on the network processor and the data is then passed back to the corresponding DSP. Since the network processor is operated by LINUX standard XML parsers could be used which significantly reduces development effort.

3.5 Dynamic Component Composition

3.5.1 Dynamic Loading and Reconfiguration

As described earlier a central aspect of our smart cameras is the dynamic loading and unloading of video analysis algorithms at runtime. The *Dynamic Loader* module from Texas Instruments is able to dynamically link and load DSP binaries and has been integrated into the DSP-FW.

Furthermore, each algorithm has to support different QoS levels that can be changed at runtime. A required change in the QoS configuration is signaled by the DSP-FW using a special command message type. Commands are not time-critical and are, therefore, not treated as important as normal data services with tight timing requirements.

In general there are two different types of trigger sources for reconfiguration actions. One source of triggers for these reconfigurations are alarms generated by the analysis algorithms. Another possibility for triggering a reconfiguration are events raised by internal system-level services like the load distribution service [BRS05], the power management facility [MRS05], or a failure management service.

3.5.2 DSP Algorithm Component Model

To support the dynamic reconfiguration of algorithms, i.e, their composition and change of attributes, in our surveillance applications it is necessary for each algorithm to comply with a special component model—the *DSP Algorithm Component Model* (DACM)—as indicated by Figure 3.5. The DACM is based on the XDAIS algorithm component model from Texas Instruments [Ins02]. It extends the XDAIS model to support dynamic loading and the publisher-subscriber communication scheme, as well as by adding crucial entries in the algorithm's resource descriptions to address all critical system resources. In the XDAIS model the focus is on

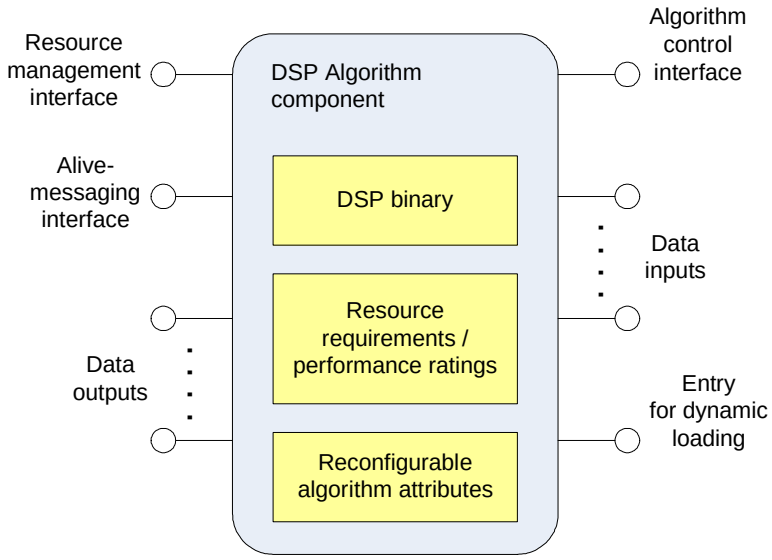

Figure 3.5: Principle structure of a DACM component.

design time integration and, therefore, resource ratings are only provided in the component documentation.

In the DACM all components have to provide all their resource information at runtime to allow for dynamic component composition. It further defines the necessary interfaces and algorithm descriptions that are required by the framework to load an algorithm, i.e., a component, at runtime. Only algorithms following the DACM can be dynamically composed at runtime.

The DACM is the basis for a safe composition of video analysis algorithms at runtime. As mentioned earlier each algorithm in our system is a component following the DACM. That is, each algorithm provides well defined interfaces and descriptions of its resource requirements and average performance ratings for each of its QoS levels. Algorithm characteristics that have to be exhibited by each algorithm component are collected in Table 3.1.

In the framework the resource manager module keeps track of already allocated resources and available resources. Based on this information and the algorithm characteristics the framework can decide whether a component can be (dynamically) integrated into the system. Note that the *enhanced direct memory access controller* (EDMA) of the DSPs is a critical resource as image analysis is very memory intensive and data is mostly copied by EDMA to keep CPU load as low as possible.

3.5.3 Ressource Monitoring and Component Composition

The PS-MW has to ensure proper component composition when new algorithms are loaded at runtime. As a basis the framework uses the component resource descriptions provided by each algorithm following the DACM to determine the component's resource requirements. Now to decide upon the feasibility of a composition the available resources in the system have to be calculated and compared

Required Services from other components
QoS levels
Resource requirements
EDMA channels and their priorities
EDMA tables
EDMA interrupts
Performance Ratings
CPU utilization for each QoS level
Transfer frequency of each EDMA channel
Transfer length of each EDMA channel

Table 3.1: Example algorithm information as provided by the DACM.

to the resource requirements. The resource monitoring module in the framework constantly computes resource loading.

Resource Monitoring

Countable resource metrics like the number of used EDMA channels, EDMA tables, and EDMA transfer complete interrupts are quite easy to determine for each algorithm. In the software framework this is achieved by a *EDMA manager* that is the only authority to request EDMA related resources. Therefore, it is also easy to check whether a component's resource requirements can be met by a simple comparison of available and demanded resources. Only if enough resources are available the component is loaded and started. The actual composition is then simply realized by the PS-MW. All required data services are looked up and connected adequately as described in Section 3.4.

On the other hand, it is quite hard to provide exact characteristics of more complicated resource metrics like CPU utilization, PCI bus utilization, and EDMA controller utilization—they are also subject to constant fluctuations which makes accurate a priori characterization impossible. However, these metrics are typically critical in terms of real-time operation of the system. As they are dynamically changing it is necessary for the framework to observe them constantly. If limits are going to be violated the framework initiates a graceful degradation in QoS of less important algorithms. That is, the QoS levels of low priority algorithms are reduced. Prioritization of algorithms is defined by the application.

An implicit assumption for this procedure is that a lower QoS level results in reduced resource utilization. In case that QoS reduction does not yield enough resources for the most important algorithms to run then the least important algorithms are removed from the system until the remaining more important algorithms can be run. This procedure ensures that as many algorithms as possible remain functional. However, if high priority tasks have to be degraded in their QoS too much or they have to be removed the application's requirements cannot be met any more and a system failure notice is generated. Application requirements are

provided to the framework by the means of a degradation policy and the above mentioned prioritization.

Information about PCI bus utilization is not part of an algorithm description. As algorithms are composed at runtime it cannot be determined a priori by the algorithm designer whether local mailbox communication or remote PCI communication will be used at algorithm deployment. However, for system stability it is important not to overload the PCI bus. Therefore, PCI utilization is monitored by the resource manager on the network processor. To do so it collects measurements of the traffic through the MAOs of all DSPs and the network processor. This is possible because the MAO is the unit on each processor where all traffic to other processors is routed through. Therefore, overall PCI bus load in a single ***Smart****Cam* i, i.e., $Load_{PCI,i}$, can be computed as

$$Load_{PCI,i} = Load_{PCI,XScale} + \sum_{n=1}^{N} Load_{PCI,DSP_n}, \tag{3.1}$$

where N is the number of DSPs and $Load_{PCI,XScale}$ and $Load_{PCI,DSP_n}$ denote the load in bytes per second measured at the MAO of the XScale and DSP n, respectively.

Utilization of the EDMA resources on the DSPs is a critical metric for overall system performance because image data is mostly transferred by EDMA. If the EDMA subsystem is overloaded the timely operation of all algorithms is at risk. To improve the reliability of the system especially with respect to timeliness it is necessary to avoid resource overloading. EDMA controller load generated from an algorithm is estimated from the algorithm's characteristics provided by the DACM. It can be noted as $Load_{EDMA} = \sum Load_{EDMA,l}$, where $l = 1, \ldots, L$ are the L hardware priority queues of the EDMA controller and

$$Load_{EDMA,l} = \sum_{c=1}^{K} length(c,l)\ freq(c,l) \tag{3.2}$$

denotes the transfer bandwidth of priority queue l taking into account all of the K channels c. The function $length(c,l)$ yields the number of bytes transferred on channel c iff channel c is assigned priority l. It returns zero for all other values of l. Similarly, $freq(c,l)$ yields the number of transfers issued per second on channel c iff c is assigned priority l.

The third critical system resource is memory. As the PS-MW provides a dynamic environment it is key to estimate dynamic memory usage of algorithms and to monitor dynamic memory availability. Fortunately, it is relatively straightforward to profile memory consumption of algorithms at design time. Therefore, algorithm resource descriptions can be made quite accurate. Monitoring of free dynamic memory resources is done by querying operating system memory management facilities. It is, therefore, easy to decide whether an algorithm component's memory requirements can be met.

However, due to the high dynamic execution environment on a ***Smart****Cam* memory fragmentation can be a problem. Without garbage collection or other sophisticated memory management policies the only way to resolve this problem is

to monitor the fragmentation. If a critical level of fragmentation is reached it is necessary to remove all algorithms and reload them again. Fortunately, a high level of fragmentation is very unlikely. The coarse-grained partitioning into whole algorithms as components that are dynamically loaded and unloaded make sure that mostly large memory blocks are allocated and freed.

Component Composition

Given the resource requirements information in the algorithm description of the DACM and the continuous monitoring of actual resource occupancy as described in Section 3.5.3 the basic step of the composition process is a comparison of required to available resources.

If feasibility with respect to resource requirements of the algorithm is confirmed the algorithm is loaded by the dynamic loader facility. On load of the algorithm it registers with the PSM. That is, it queries for services it requires and publishes services it provides. In this respect our approach is somewhat different to other component-based middleware because the algorithm is loaded even if required services are currently not available in the system. However, then the algorithm is put to sleep because it cannot do its work. But if at a later time another component is inserted that provides the missing service then the sleeping algorithm is brought back to work by the PSM. With this simple mechanism we can load algorithms without bothering about the sequence of algorithms defined by data-flow dependencies.

Another relaxation in our component composition approach compared to standard middleware technology is that there is some degree of freedom concerning service querying. In general it is necessary that service interfaces, i.e., output of one component and input of another component, completely match in order to be connected. This is in principle also true for this approach but with the introduction of different QoS levels it is also possible for a component to accept services that do not match up to a certain extent. Of course, it is required that key attributes have to match. But it is up to the algorithm to decide which ones it is able to accept even if diverting.

In that respect it is possible that there are several services available in the system that potentially match a new components requirements. Then this component has to choose one of these. Generally, the one with the highest QoS level would be the best choice. But especially in abnormal situations like failure conditions and the like the situation might be different. Then it could be the case that using a lower quality service can allow the algorithm to at least provide rudimentary functionality. This is a basic feature that is exploited when graceful degradation is used to cope with faults that lead to resource failures.

3.5.4 Component Performance Monitoring

It is important for several reasons to continually monitor all components in the system for their performance. First, it allows the framework to reason about likely deadline misses that compromise real-time operation. Second, performance mea-

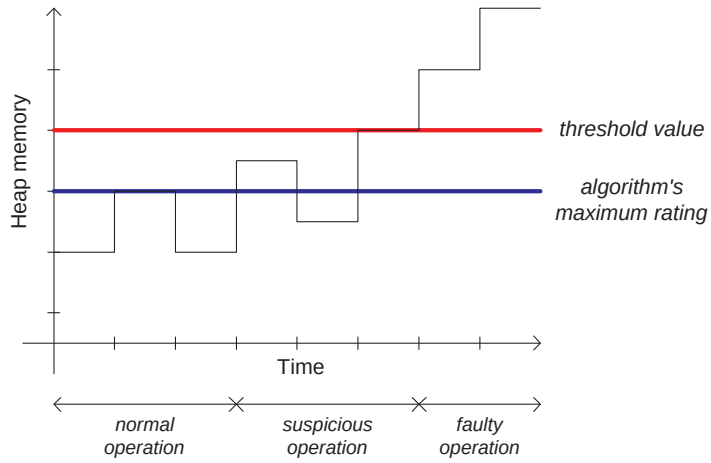

Figure 3.6: Simple example for heap allocation problems. The algorithm allocates more dynamic memory than it specified to the framework.

surements can be used to reason about the fitness of components which is important for fault-tolerance mechanisms.

Dynamic Memory Usage

Especially, memory consumption of a component observed over time can exhibit buffer management problems in algorithms or other memory leaks. Of course, only dynamic memory allocation in heap memory is observed. Operating system primitives are used to determine current memory usage for each task in the system. This is sufficient since every algorithm runs in its own execution task.

Figure 3.6 illustrates a simple example where an algorithm uses up more heap than it specified to the framework. The maximum rating it provided at registration time is marked as well as a threshold value that can be set to deal with measurement errors. If the threshold value is surpassed a problem with the algorithm's memory management is very likely. This simple mechanism is one possibility for the framework to monitor algorithms for correct behavior. Note that the threshold value is arbitrarily set by the system integrator based on experience and experimental examination.

Execution Time

Execution times are constantly measured by hooks in the PS-MW at the inputs and the outputs of all algorithms (cf. Figure 3.7. That is, a system counter is captured each time a hook function is called in a Subscriber or a Publisher, respectively. By this mechanism current computation time in CPU cycles is determined as the difference $T_{A_i,exec} = |T_{A_i,out} - T_{A_i,in}|$, where $T_{A_i,in}$ represents the counter value at the time when all inputs of algorithm A_i were ready. $T_{A_i,out}$ stands for the counter value when all outputs of algorithm A_i were ready. Algorithms are typically such that they perform a loop in that they take some input data, transform it somehow,

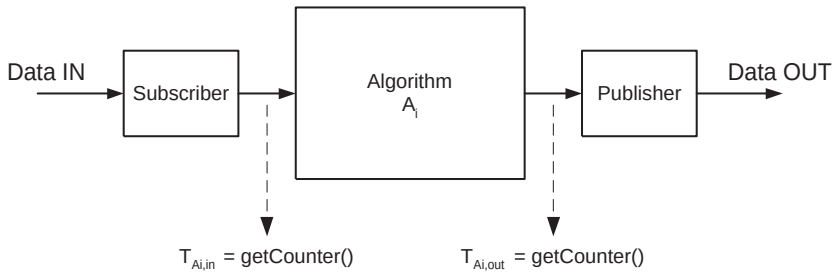

Figure 3.7: Basic mechanism to get an estimate for algorithm execution times.

and produce some output data. Therefore, measuring the CPU cycles from the moment an algorithm receives data to the moment it post the output is a good estimate for its execution time.

As a side product the input frame rate of an algorithm can be checked by observing two subsequent input counter values $T_{A_i,in}[n]$ and $T_{A_i,in}[n+1]$. An estimate $E_{f_{frame}}[n+1]$ for the current input frame rate of algorithm A_i at sample time $n+1$ is then given by

$$E_{f_{frame}}[n+1] = \frac{f_{CPU}}{|T_{A_i,in}[n] - T_{A_i,in}[n+1]|} \tag{3.3}$$

where f_{CPU} denotes the clock frequency of the CPU. By continually observing these frame rate estimates problems can be detected early so that interventions are likely to prevent failures.

Communication Delay

Another performance rating that can be observed by the framework is communication delay. That is, the delay from a publisher to its associated subscribers is evaluated. As the execution time the communication delay is also an estimate based on capturing a counter at well defined interaction points in the publisher-subscriber subsystem.

In Figure 3.8 the principle is illustrated for two algorithms A_i and A_{i+1}, respectively. Note that the same measurement points are involved as used for the execution time estimation. But in this case the probe points of different algorithms are used.

The estimate $E_{A_i \to A_{i+1}}[n]$ for the communication delay at sample time n can be written as

$$E_{A_i \to A_{i+1}}[n] = \frac{|T_{A_i,out}[n] - T_{A_{i+1},in}[n]|}{f_{CPU}} \tag{3.4}$$

with f_{CPU} being the CPU clock frequency.

A trend of communication delay estimates over a certain time can reveal timing problems. Possible causes could be high loads on the CPU or the PCI bus. Single absolute values of communication delay can be used to uncover real-time problems. For example, the sum of all execution times and communication delays

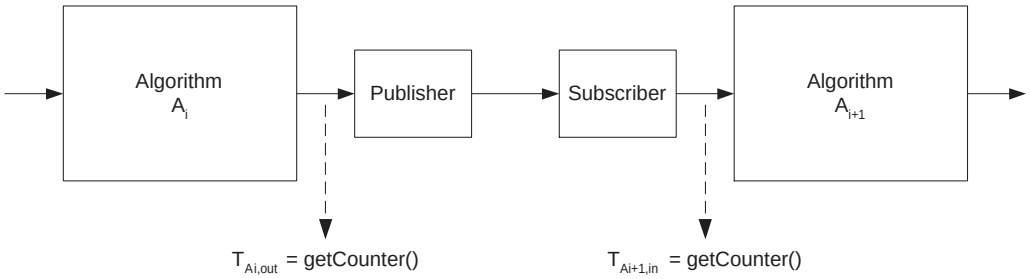

Figure 3.8: Basic mechanism to get an estimate for communication delay from algorithm A_i to algorithm A_{i+1}.

in a processing chain determine the maximum possible frame rate at the system perspective.

3.6 Fault Handling in a Network of Smart Cameras

To take a step towards autonomous operation of smart cameras middleware-based fault tolerance mechanisms have been introduced into the software framework. The main management parts of this *fault tolerance architecture* (FTA) are hosted on the network processor within the SC-FW, whereas mainly low-level monitoring is included in the DSP-FW of each DSP.

The principle idea of the FTA is to introduce some degree of fault-tolerance without imposing additional hardware costs. That is, even sophisticated software replication techniques are not considered as they also require additional hardware. Tight cost requirements in the embedded video surveillance market demand for trading system dependability for unit price.

But nevertheless it is possible to exploit domain-specific knowledge and the cooperation feature of smart cameras to provide some fault-tolerance. That is, simple metrics are used to detect and localize faults and then graceful degradation is employed to mitigate fault effects to prevent system failures.

This is especially interesting as there is an increasing trend towards integrating third-party algorithms. Generally, these external algorithms have to be considered less trustworthy than algorithms developed in-house. Unfortunately, also in-house software is unlikely to be completely correct. Therefore, it makes sense to use fault-tolerance mechanisms to increase system dependability in spite of potentially unreliable (software) components.

In case of hardware failures the dynamic reconfiguration features of the ***Smart****Cam* described earlier in this chapter are now used to provide the above mentioned fault-tolerance mechanisms. First, it is possible to migrate algorithms to different cameras. This is, of course, only feasible if the application is such that affected algorithms are not bound to dedicated cameras. By migrating algorithms resources are freed locally so that failed hardware could be tolerated. Second, algorithm's QoS levels can be reduced so that less resources are occupied. This graceful degradation is only feasible to a certain extent as the application might require

minimum QoS for dedicated algorithms. But with these actions the chances are better that most services can be provided at least at some degraded level instead of resulting in a system failure.

In order to improve efficiency fault-tolerance is provided by the software framework rather than by each application, i.e., algorithm. Such a middleware-based approach reduces code size because fault-tolerance code has to be included only once in the whole system. Furthermore, there are two levels of fault-tolerance mechanisms to be considered. First, at the node level the view is restricted to a single smart camera. This is the level of self-diagnosis where different means for detecting faults within the multi-processor architecture of a *SmartCam* are employed. Second, the system level or network level involves mutual monitoring of neighboring cameras, i.e., nodes. On this level communication problems and inconsistencies in scene observations can be detected.

In the following the fault classes and fault handling procedures provided by the middleware framework are introduced. It is outside of the scope of this thesis to take into account every possible fault scenario. Rather it is the aim of this work to increase service availability by simple techniques that impose minimal overhead in the system.

3.6.1 Considered fault classes

As far as this work is concerned only a subset of all possible faults within a single *SmartCam* or in a network of collaborating *SmartCams* is considered.

Algorithm faults The main focus of the FTA is to provide some higher level fault detection for algorithms based on application-specific knowledge. That is, analysis results of different algorithms are often related to each other. For example, in case of a traffic jam two stationary vehicle detection algorithms on two adjacent cameras have to come to the same decision—at least after some limited time interval. If one algorithm detects the traffic jam and the other fails to do so it can be deduced that one of them exhibits incorrect behavior. There are other cases where inconsistent observations of two or more algorithms suggest a failure of one of them.

It is assumed that all smart cameras are geographically collocated and that all nodes know their neighbors, i.e., the network topology. These assumptions result from the main application area of traffic video surveillance along highways or in tunnels. There the collocation is very regular and linear with relatively constant overlapping of observed scenes.

Furthermore, in future work it should be considered to make algorithms more robust by incorporating self-checks. These could be plausibility tests of input data and its own analysis results. In the framework there is a simple interface that allows an algorithm to rate its input or outputs as credible or not. The FTA then decides how to use this information for further diagnosis.

Communication faults It is essential for a network of cooperating smart cameras to have mutual communication paths readily available to exchange informa-

tion about the observed scene. Furthermore, a smart camera can only provide its functionality if it is able to communicate its analysis results and video streams to interested entities in the system. The middleware framework employs a simple messaging protocol between neighboring nodes to check mutual reachability. These so called *alive-* or *heartbeat-messages* are exchanged regularly. Missing messages over a preset time span results in the corresponding node to be considered as down.

To save bandwidth these alive messages are sent in intervals of a few seconds. If there is other interaction between two neighboring nodes then no additional alive messages are sent. By this overall messaging overhead can be reduced to a minimum.

Hardware faults As the main focus of the FTA is handling software problems it makes sense, however, also to treat several hardware problems. This class is considered mainly because it is relatively easy to handle them in the given framework for dynamic reconfiguration and coping with algorithm problems. In detail the following hardware faults are considered for fault handling: DSP outages and outage of a whole ***Smart****Cam*.

3.6.2 Fault handling procedures

To cope with the above-mentioned fault-classes different counter measures are used by the FTA.

Algorithm reload If an algorithm shows unexpected behavior it has to be restarted. That is, the algorithm is reloaded on the DSP. This procedure takes only a couple of milliseconds.

As the reboot of a DSP takes significantly longer than the reload of an algorithm, it is advisable to first try if reloading the algorithm in question solves the problem. But if restarting or even repeated restarting does not lead to a successful recovery, rebooting the DSP can be of assistance. Incorrect behavior of all algorithms running on one DSP indicates a malfunction caused by the DSP and a reboot is necessary.

Graceful degradation A key mechanism for increasing service availability is to degrade some functionality in spite of risking overall system failure. There are basically two basic means for degrading a service. First, its QoS level can be reduced. A simple but realistic assumption we make here is that higher QoS results in increased resource usage. Second, an algorithm can be shut down completely.

A special *importance* measure is assigned to each active algorithm to decide about the order in which algorithms are considered for degradation. The lower this importance of an algorithm the earlier it is considered for degradation actions. That is, algorithms with lowest importance are first reduced in their QoS levels or shut down completely, respectively. The FTA also has to be able to modify each algorithm's importance as needed to gradually adapt to the needs of the environment, i.e., application requirements.

DSP reboot The reboot of a DSP is necessary if a DSP crashed or is under the strong suspicion to have at least partially crashed (e.g., temporary malfunction of the RAM).

Node reboot In case that a node recognizes that it is isolated from the rest of the network it can decide to undergo a reboot procedure to eliminate possible transient network (stack) problems. Rebooting in this case is unproblematic because if network communication failed it does not contribute to system goals any more. But chances are that a transient problem is eliminated after reboot.

Operator notification An operator, i.e., some global monitoring authority, has to be informed if any unexpected behavior is noticed. If a node repeatedly shows abnormal behavior despite automatic recovery actions human inspection and maintenance actions are inevitable. Therefore, all detected fault events subsequent counter measures are logged so that an operator can retrieve the information on demand.

Furthermore, a node must always be informed about the operational reliability of its neighbors. That is, if a node diagnoses itself as (partially) faulty its direct neighbors have to be informed about these fault condition. This is to simplify credibility checks in neighboring nodes. Because if a node testifies itself als faulty the others can skip the voting process and exclude the faulty camera's results from further consideration.

3.7 Middleware-Based Fault-Tolerance Architecture for Smart Cameras

As introduced in Section 3.6 the ***Smart****Cam* software framework incorporates the fault tolerance architecture (FTA) that provides fault-tolerance as middleware services. The FTA comprises several units on the network processor and the DSPs. Figure 3.9 illustrates the principle relationships of the different components.

Every algorithm on the DSP that is subject to the FTA monitoring is registered with the PSM so its input and output connections are known. Furthermore, the algorithm descriptions as described in Section 3.5.2 provide the basis for decisions on the algorithm's resource usage. Most interesting in this respect are the algorithm's name for identification, its current QoS level and what other QoS levels are offered, the resources requirements for the current QoS level, the current importance measure assigned by the application developer, and information about the algorithm's typical execution time. As said before this information is extracted from the resource description the DACM requires for each algorithm and the registration information at the PSM.

The principle functionality of the involved framework components is described in the following. For a more elaborate treatment of the details of the monitoring and diagnosis architecture for the ***Smart****Cam* refer to [Kla06].

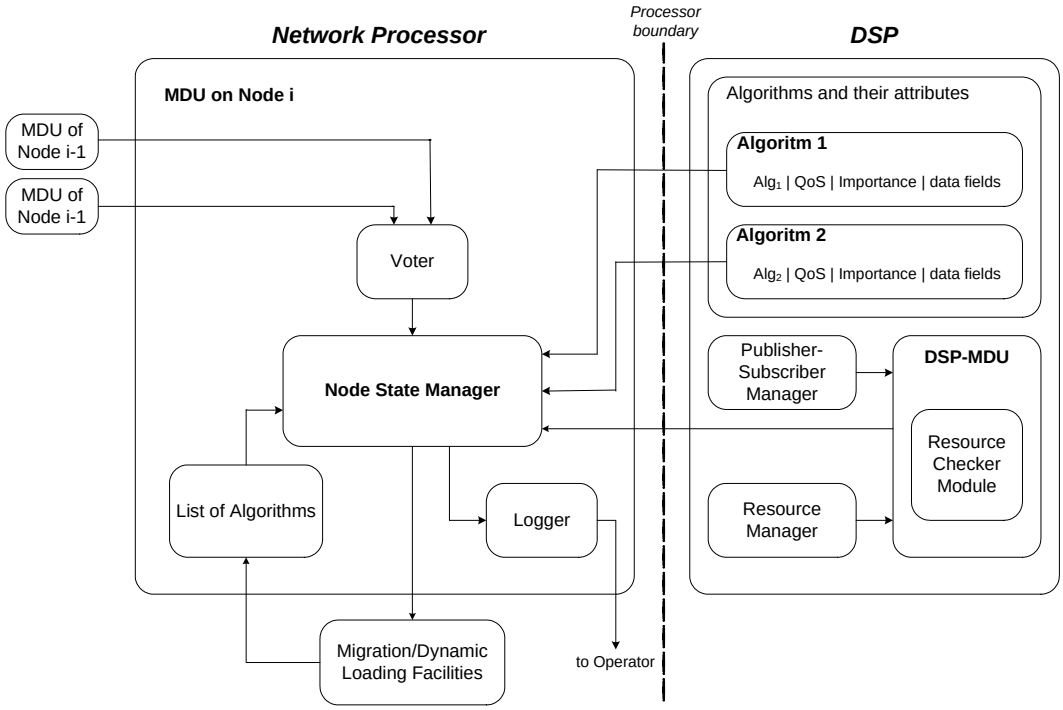

Figure 3.9: Overview of the fault tolerance architecture as it is included in the ***Smart****Cam* software framework.

Resource Checker Module The main part of the FTA resides in the so-called *monitoring and diagnosis unit* (MDU) on the network processor. On the DSP the only module in addition to already described framework components of the DSP-FW is the *resource checker module* (RCM). Relevant data concerning projected and actual resource usage of each active algorithm are conducted into the RCM. Resources demanded by the algorithms are compared to the resources available in the system. Necessary resource information is queried from the RM residing on the DSP. The RCM determines whether sufficient resources are available and also communicates its data to the node state manager (NSM) on the network processor.

Node State Manager The *node state manager* (NSM) is the central entity of the FTA. It determines a node's state by evaluating data from the RCM, the voter and the analysis results of the currently active algorithms.

The most simple decisions are these that are based only on resource usage information. Because they are comparisons of metrics that are readily available. A more sophisticated monitoring approach is achieved by judging analysis results in view of results of other algorithms within the same node or from neighboring nodes. In this respect the application developer programs conditions for analysis results into the NSM. Typically these conditions are also quite simple but to come to a monitoring and diagnosis decision it is necessary to consider several variables possibly from different nodes. For example, in video traffic surveillance a condition can require the traffic statistics of adjacent nodes to match within a given tolerance

reflecting measurement deviations—provided that there are no crossings along the scene observed by the involved cameras. The *voter* unit compares the node's analysis results with the results of the two closest neighboring nodes. Deviations are then feed to the NSM that changes the node's state if applicable. A reasonable frequency for voting analysis results from algorithms is once in a few seconds depending on the application requirements. It makes sense to compare results not too often to allow for a reasonable change of the environment, i.e., traffic flow in the above traffic statistics example. Furthermore, as the voting process involves data from neighboring nodes bandwidth considerations in the network also suggest a reasonable frequency.

Note that this regular communication of analysis results also serves as alive messaging between the nodes. Therefore, no polling of dedicated alive messages is required as long as algorithms are exchanging analysis results.

Different none-exclusive states or modes characterize the system on node level as well as on system level. They are described in Section 3.7.1 and Section 3.7.2, respectively.

Logger The node's state is recorded by the *logger* framework unit. This data can be used to detect abnormal behavior in the long term behavior of a node like periodical failures of the hardware or software due to, e.g., environmental conditions. Depending on the node's state appropriate actions are induced by the NSM and recorded by the logger.

Furthermore, logged data can be retrieved by remote clients, i.e., operator workstations. Especially, for system maintenance this information is valuable. Additionally, the logger collects fault and failure histories of monitored entities so that each components reliability can be scrutinized from this data. One interesting use case for logged faults, failures, and reconfiguration actions is post-mortem system analysis for tracking errors.

Reloading In case of the necessity of a reload or unload, the NSM instructs the *migration and dynamic loading facility* (MDL) to reload or unload the algorithm in question. The MDL induces the reload or in case of an unload performs the unload and updates the list of current algorithms residing on the DSP. This list holds information including which algorithm runs on which DSP on this node as well as on the two closest neighboring nodes. In that way, the status quo can be restored after rebooting from a DSP crash. Should the network processor have to reboot too, the list of former active algorithms can be retrieved from up to two neighbors. This so called *distributed information* introduces a form of redundancy that contributes to the systems recovery capabilities and reduces initialization times.

3.7.1 Node Level Modes

As mentioned earlier the NSM distinguishes two different mode categories. The first, called *node level modes* comprises five distinct modes that a single node can be determined to be in. These five modes are briefly described in the following.

- In *normal mode* all algorithms work correctly and sufficient resources are provided.
- In *low resources mode* insufficient resources are available, due to, e.g., a memory leak.
- In *DSP crash mode* one or more DSPs have crashed. This event is detected by a watchdog timer. If there is no active communication within a reasonable time interval the DSP is considered to have crashed.
- In *algorithm crash mode* one or more algorithms have crashed. Detection is achieved similar to the recognition of a DSP crash if there are no alive messages from the algorithm under test any more.
- In *malfunctioning communication device mode* communication to other nodes is not possible. No network communication can be established. Therefore, the node's contribution to overall system functionality is non-existent. However, it makes sense to keep the algorithms working so that they do not have to re-initialize when the network becomes available again.

In normal mode everything is fine and no further actions are needed. In case of low resources one or more algorithms have to be switched to a lower QoS level. Alternatively, one or more algorithms have to be unloaded in order to ensure a continual functional system. These algorithms are determined via their current importance. Resource problems also include lack in bandwidth for sending video streams.

In case of a partial node crash, i.e., some but not all DSPs are non-functional, the neighboring nodes have to be informed. Additionally, the load distribution service described in [Bra05] can be triggered to compute a new algorithm allocation. If the resulting reconfiguration leads to a situation in that a node N_i cannot deliver necessary data for it's neighbors' voting processes node N_{i-1} becomes the new neighbor of node N_{i+1}. If node N_i has sufficiently recovered from the crash the original configuration is re-established again. Note that the list of formerly active algorithms on node N_i is retained on nodes N_{i-1} and N_{i+1}. Without this change of neighbors it would not be possible to further perform a majority voting.

3.7.2 System Level Modes

By considering system level observations and events the NSM distinguishes *system level modes*. Based on results from multiple nodes sharing monitoring information and exploiting application specific knowledge such system level modes can be defined. Note that especially system level modes are not only domain-specific but also very application-specific. That is, they have to be defined by the application developer. For the video surveillance scenario in our project we decided on the following three system level modes.

- *traffic jam mode*, referring also to stop-and-go traffic applies in case of

  - the detection of many stationary vehicles in a small area.
  - a very low average vehicle speed.
  - a significant thinning of the traffic flow behind the perceived cause of the traffic jam (i.e., an accident or construction site).

- In *obstacles mode* the road is blocked by, e.g., lost cargo.
- In *inconsistent observations mode*, one or both neighboring nodes observed different events.

3.8 High-Level Software Development for DSPs

As stated in previous sections future video surveillance systems integrate image acquisition and analysis with compression and network communication functionality into a single embedded device [WOL02]. High performance DSPs are often used to provide the required processing power. Such complex configurations impose significant challenges on the software development. Tight resource constraints have to be met while facing increasing application complexity and pressing time-to-market demands. Although modern DSPs offer substantial computational resources code optimization is mostly crucial for media applications [KMGK03, BBRS04].

It is a major challenge in embedded software development to increase the level of abstraction while meeting tight resource constraints [SVM01]. Recently added support for DSP targets in the synthesis tools for Simulink [The06] simplifies high-level development for such platforms. Block-oriented modeling also supports hierarchical designs that promote reuse and address algorithmic complexity. Validation by simulation is already available in early development stages. Synthesis tools in combination with target specific components are used to generate production code for the embedded platform.

For this work model-based development and synthesis using the Simulink environment have been explored. The intended scenario is to directly integrate synthesized algorithmic code into an intelligent embedded multi-DSP camera for video surveillance. An overview of the model-based development process together with an evaluation of the quality of DSP code synthesized using the Real-Time Workshop Embedded Coder (RTW-EC) for Simulink is presented in the following. For a more elaborate description of the model-based development process for video analysis algorithms using Simulink refer to [Gös05].

3.8.1 Model-Based Development of Embedded Video Surveillance Applications

Model-based design is a generic development paradigm that addresses system specification, validation by simulation, model analysis, synthesis, and test. The key idea is to build a model that satisfies the requirements and to use this model (or automatically transformed models) for all further development steps including code generation.

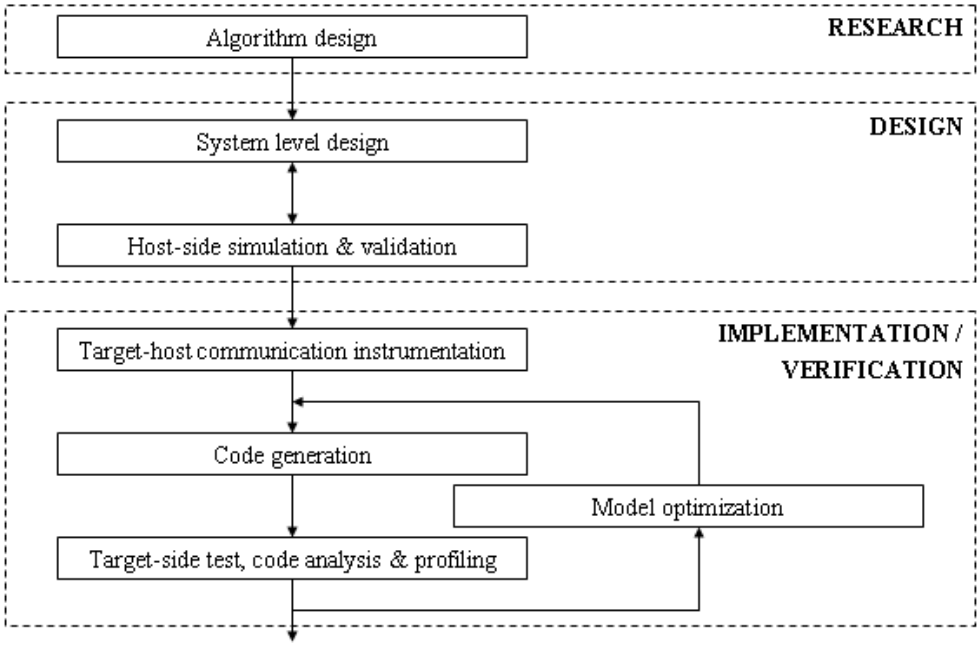

Figure 3.10: Generic model-based development process [The06, KSLB03].

In Figure 3.10 the basic steps of a model-based development process for video analysis applications are illustrated. Basically, all development phases depicted in Figure 3.10 are supported by Matlab/Simulink together with the RTW-EC and the Embedded Target for Texas Instruments C64x DSPs (ET). Algorithmic design (research phase) is usually performed using a high-level development environment such as Matlab or C/C++ libraries. System level design translates the well-defined algorithms into the domain of the modeling-language. Simulation is employed to maintain a validated reference. Optionally, the model can be instrumented for target-side testing. Finally, code is synthesized and can be executed on the target. The overall build process of the Simulink environment is illustrated in Figure 3.11.

Taking into account timing constraints and resource requirements of algorithms in the video surveillance domain, highly efficient code is needed. Unfortunately, current modeling systems do not provide special video analysis function blocks that yield efficient DSP code. Algorithms have to be modeled by intricate compositions of simple blocks that are often transformed to suboptimal code. Therefore, optimizations are needed to generate efficient code. Modifications on synthesized code are not an option. They would break up the mapping between model and generated code such that the model-based design process would be corrupted. The only choice is to optimize the model.

For that purpose tools like Simulink provide mechanisms to extend their built-in functionality. Such blocks can be written in a traditional programming language (e.g. C/C++). When implementing custom modules to improve efficiency of synthesized code one has to consider two important issues.

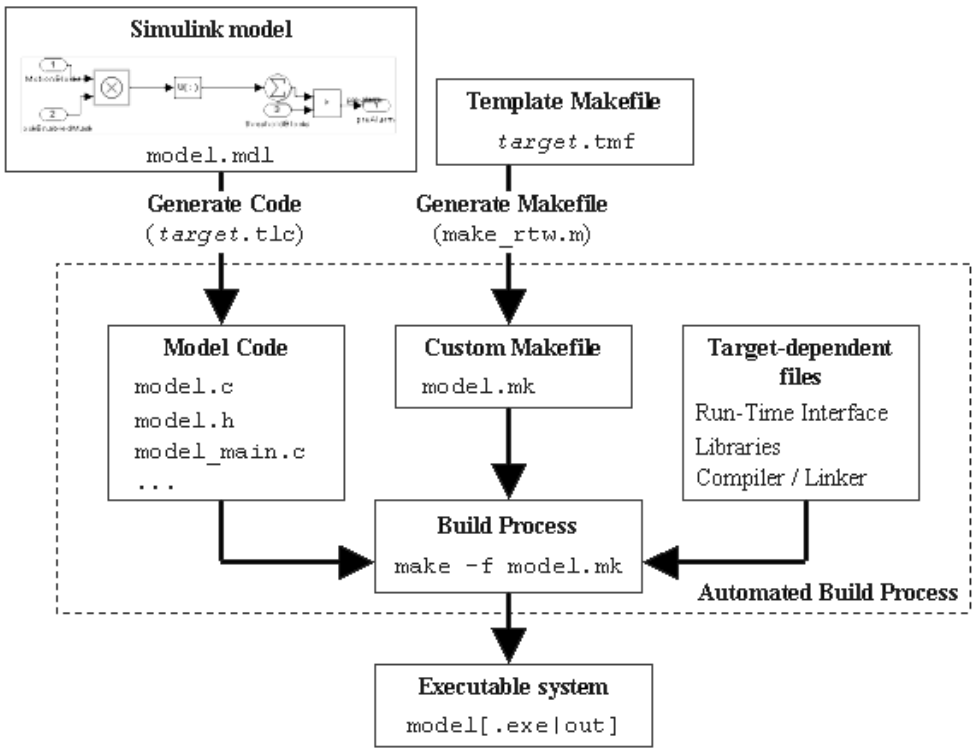

Figure 3.11: The complete build process that generates code from a Simulink model using the Real-Time Workshop program contained in the development suite [The06].

1. Function granularity. Fine-grained modules with very basic functionality ensure high flexibility and reusability. Coarser-grained modules, however, offer better opportunities for optimizations by the compiler [BBRS04].

2. Optimization level. Generic implementations in ANSI C ensures platform portability. Target specific C implementation, i.e., C plus intrinsic instructions and compiler directives, on the other hand, make use of proprietary hardware features, e.g., direct memory access (DMA) and, therefore, yield performance gains.

In Section 4.4 an experimental evaluation of the model-based development approach for automatic code generation of video analysis algorithms is presented. A discussion of the major findings and conclusions are also summarized there.

3.8.2 Integration of Automatically Generated Components

An interesting extension to the work presented in [Gös05] is the possibility of integrating automatically generated modules directly into the software framework of the ***Smart****Cam*. This is possible if the presented component model described in Section 3.5.2 is reflected in the modeling environment. That is, code generation templates have to be adjusted to use dedicated interface calls for resource management and inter-component communication. In the presented framework all components have to communication via the PS-MW described in Section 3.4. Furthermore, a special block has to be provided that lets the algorithm component developer specify the algorithm's performance figures, resource requirements and other information that is required for compliance with the DACM. It has to be ensured that the resulting binary generated from the code generator is executable on top of the software framework running on the DSPs. In Simulink the custom block feature and custom code generation templates are the means to adjust the environment for a special domain, a custom hardware platform, and a specialized software framework.

Figure 3.12 illustrates the process of integrating automatically generated algorithm components with the software framework into the final application. The three different developer roles contribute to different parts of the development process. It is interesting to note that the algorithm developers are quite loosely coupled to the rest of the process. This is achieved by the high-level development approach provided by Matlab/Simulink and automatic code generation. As it is often the case that algorithm developers are not familiar with the hardware platform it makes sense to shield them from device details so that they can concentrate on efficient and robust algorithms. On the other hand, platform experts and application domain experts implement the code generation templates and the software framework that make algorithm development transparent to the hardware details. By adhering to standard interfaces required by the framework, i.e., the component model, it is possible to generate algorithms from high-level models so that the binary components can be integrated into the final application.

Unfortunately, as the overall model-based development process leads to a lot of effort that has to be invested upfront it makes sense only to switch completely to

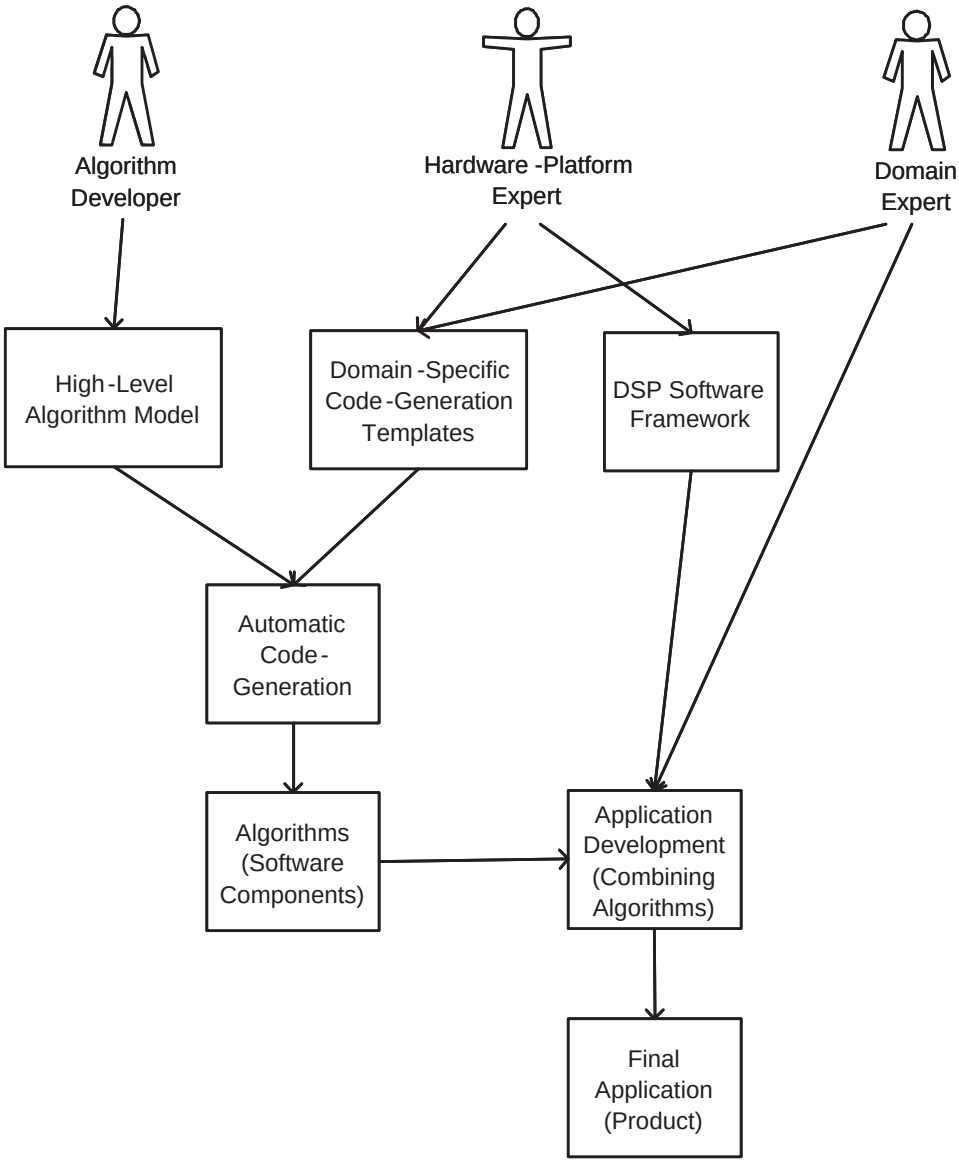

Figure 3.12: Development process that uses automatic code generation and integration of generated components into the framework.

the model-based paradigm. That is, block libraries have to be created that provide domain-specific functionality. More than that, the hardware platform and the basic software framework have to be reflected in the models. General support for different processors, e.g., the Texas Instruments C64x DSPs, is already integrated into Matlab/Simulink. But for custom boards the board-specific features have to be integrated manually. For the incorporation of framework behavior into the modeling environment it can be said that this is only necessary if simulation for component validation is intended. However, simulation is a key benefit from component development in Simulink.

It is also interesting to note that there are other commercial model-based development environments on the market now. Of course, the ideas presented above are also suitable for environments other than Matlab/Simulink. One of the most promising developments is the integration of the model-based paradigm and code generation in National Instruments' LabVIEW suite [Nat06]. Like Matlab/Simulink the LabVIEW integrated development environment also directly supports Texas Instruments hardware. It is, therefore, a matter of availability and a company's preferences which system fits best. Unfortunately, the different development environments are not compatible. Maybe in the future a standard for model-based development will arise that would ease exchange of models between different development suites.

Chapter 4

Implementation and Experimental Evaluation

In this chapter several experimental results are presented to better illustrate the performance of the software framework developed in this work. A short introduction to the details of the evaluation environment used for testing the PS-MW and the FTA can be found in Section 4.1. A performance analysis of the light-weight publisher-subscriber middleware (PS-MW) is given in Section 4.2. All experiments have been performed on our ***Smart****Cam* prototype platform. Key figures for memory consumption and timing measurements are presented. Section 4.3 summarizes performance figures for fault detection times based on fault injection experiments on the prototype. At the end of the chapter Section 4.4 presents an experimental evaluation of the high-level software development approach described in Section 3.8. For these measurements a different evaluation platform was used for simplicity reasons. A short description of the used platform is included therein.

4.1 Prototype Platform as Evaluation Environment

As previously mentioned the ***Smart****Cam* prototype was used for the experimental evaluation. The basis of the platform is an Intel IXDP425 development board comprising an Intel IXP425 XScale network processor running at 533 MHz. It is equipped with 16 MB of flash memory and 256 MB of SDRAM. Two to four ATEME NVDK PCI boards each comprising a Texas Instruments TMS320C6415 DSP running at 600 MHz are plugged into the base board. Each NVDK is equipped with 264 MB of SDRAM.

The XScale is operated by a LINUX kernel version 2.6.10 and the DSPs run the Texas Instruments DSP/BIOS real-time operating system kernel as provided with the Code Composer Studio 3.0 development environment. Framework components have been used as they are described in Chapter 3.

A picture of the ***Smart****Cam* prototype is presented in Figure 4.1. All major parts as described above are marked in the image. Further details of the ***Smart****Cam* are presented in Section 3.1. A more elaborate treatment of the hardware platform and its use for traffic surveillance can be found in [BDM+06] and [Bra05], respectively.

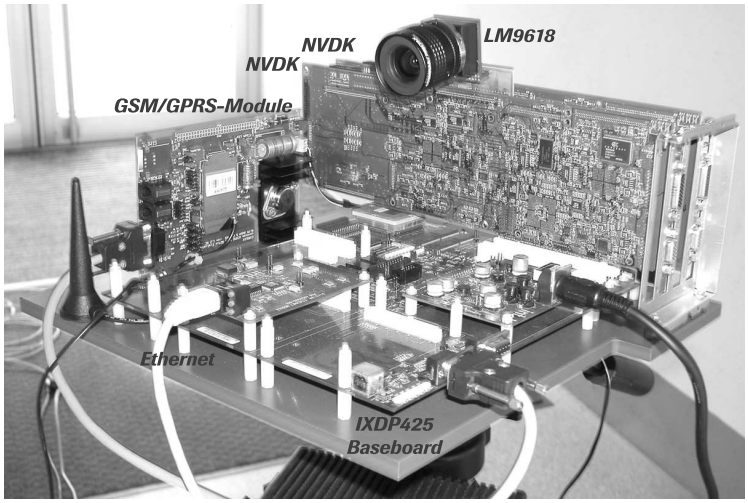

Figure 4.1: The ***Smart****Cam* prototype comprising an Intel IXDP425 baseboard, two ATEME NVDK DSP boards, a GSM/GPRS module, and a CMOS image sensor. A Ethernet connection serves as the main communication medium.

For the evaluation of the code synthesis capabilities of the Matlab/Simulink development environment a simpler platform was used as described in Section 4.4. A single DSP development board was chosen to better concentrate on actual performance evaluation of the generated code. As it comprises the same type of DSP that is also used in the ***Smart****Cam* the presented results are also significant for the ***Smart****Cam* prototype.

4.1.1 Demonstration Application

As described in Section 3.4 there are several units on the DSPs and the network processor that comprise the publisher-subscriber middleware (PS-MW). Basically, there are corresponding entities on each processor that provide the same functionality. But due to the different operating systems on the DSPs and the network processor there are slight differences in the implementation of the inter-process communication. In the following the basic implementation principles are described. A more elaborate treatment of the implementation details can be found in [Tre06].

To better illustrate the interaction of involved framework units a special demonstration application was developed. It is a simple video surveillance application hosted on the ***Smart****Cam* prototype platform. For simplicity only the robust MPEG-4 encoder algorithm was chosen for this demonstrator. This algorithm was adapted to the PS-MW. That is, the dedicated interfaces were implemented and it was prepared to be capable for being dynamically loaded. As stated earlier in this thesis it was one of the design goals to have multiple instances of the same algorithm running on one smart camera. Therefore, the demonstrator hosts two instances with potentially different QoS levels on a single ***Smart****Cam*.

Although the demonstration application is intentionally kept as simple as possible it is also relevant for current surveillance tasks. It is often required to have one

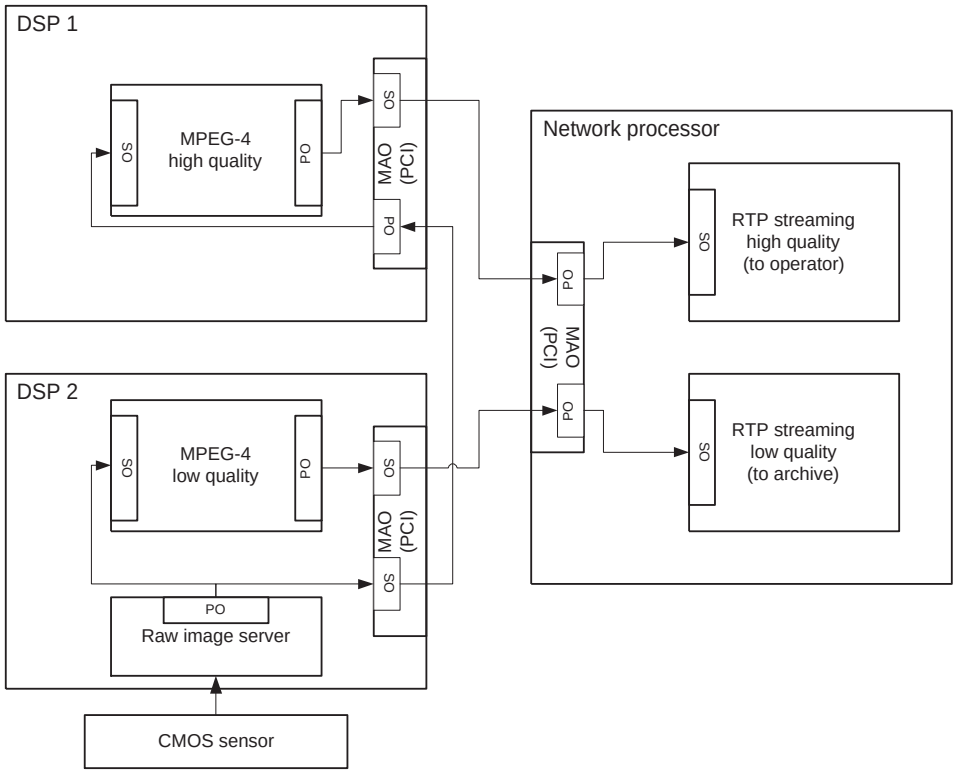

Figure 4.2: Demonstration application resembling a simple surveillance scenario where a high quality stream is sent to an operator terminal and a lower quality stream is archived to a network storage.

camera streaming to two different destinations. A high quality stream is sent to an operator for observation and a lower quality stream is archived to a permanent storage device. This scenario is depicted in Figure 4.2.

4.1.2 Dynamic loading

A prerequisite for dynamic reconfiguration of algorithm compositions is a facility for dynamic loading and linking. In our software framework the *Dynamic Loader* (DL) from Texas Instruments is used to dynamically load and link DSP algorithm binaries at runtime. Of course, the DL can also be used to unload algorithms, i.e., components, when they are not needed any more.

On load a uniform entry point is called from the DL. In this entry function the algorithm creates its own task and allocates needed resources by using a dedicated interface to the software framework. Furthermore, POs and SOs are created in order to register published services and subscriptions with the PS-MW (cf. Section 3.4).

Note that each algorithm also registers a SO for receiving algorithm control commands from the framework or other algorithms. Public algorithm attributes are configured through this command interface. After this initialization phase the

actual DSP algorithm contained in the newly loaded component starts its computations.

4.1.3 Extension to the Fault Tolerance Architecture

Test Algorithms For the evaluation of the FTA simplified algorithms are used to simulate real algorithms. These *test algorithms* (TA) are used because they ease fault injection experiments. Furthermore, there are also not all intended algorithms available for the ***Smart****Cam* platform up to now. Several algorithms are available as prototypes written in high-level languages in a form not appropriate for the embedded smart camera platform. Nevertheless, these algorithm prototypes are still to be ported to the target platform by platform experts. In the future this problem could be circumvented by using the model-based development approach described earlier in this work in Section 3.8.

Long Term Statistics In the implementation the special situation of traffic surveillance results in the need for improving decision based on unreliable analysis results. Therefore, a dedicated *long term statistics* unit (LTS) is added. A timer for periodic comparisons with relatively long intervals, e.g., 24 hours, triggers an additional voting process using the node's and its neighbors' long term statistics data. Thus, small deviations in the frequently taken measurements, that culminate in a major deviation can be detected. Note that the video-based algorithms for computing traffic statistics like average vehicle speed show an accuracy of quite less than 100%. Given current statistics algorithms, the use of traffic statistics to prove faulty behavior of an algorithm is not an exact procedure. Only significantly different statistics can be used for diagnosing faults. The resulting extension of the FTA is illustrated in Figure 4.3.

4.2 Performance Analysis of the Publisher-Subscriber Middleware

4.2.1 Memory Requirements

An important requirement for the task communication framework on the DSPs of the ***Smart****Cam* is to use only little memory to save it for the analysis algorithms. Although our middleware was implemented in C++ the memory footprint is only 15.78 KB. It can be seen from Table 4.1 that the runtime memory consumption is also low.

Total memory consumption overhead, of course, depends on the number of published services and subscriptions in the system as each of them requires a PrO and a PO or SO, respectively—and of course each object has a corresponding entry in the DS. In a typical setting there are two algorithms per DSP and each algorithm provides one service and subscribes to one service. Together with the management objects this yields a typical total memory overhead of the middleware of 3.71 KB per DSP.

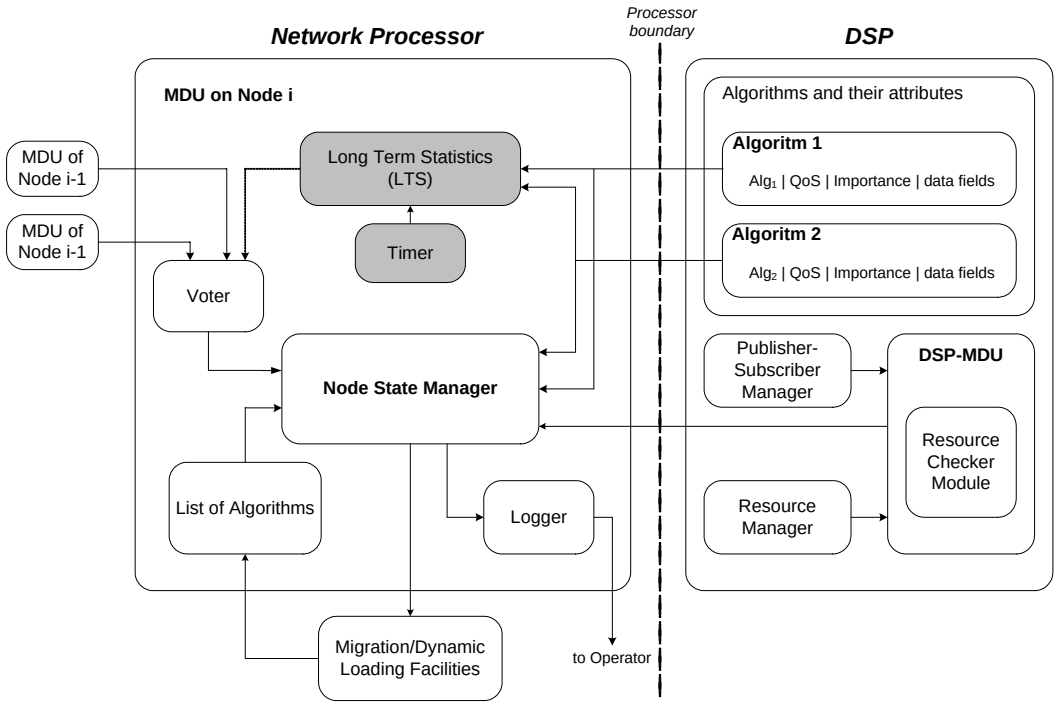

Figure 4.3: The extended fault tolerance architecture to cope with deviations of statistical analysis results on different cameras.

Middleware Component	**Memory Usage** (in bytes)
Publisher-Subscriber Manager (PSM)	472
Directory Service (DS)	256
Publisher Object (PO)	192
Subscriber Object (SO)	96
Properties Object (PrO)	34-72

Table 4.1: Memory requirements of middleware objects.

Component	Initialization time [μs]
Publisher-Subscriber Manager (PSM)	4.68
Directory Service (DS)	9.90
Creation/Registration Publisher Object (PO)	10.17
Creation/Registration Subscriber Object (SO)	11.01

Table 4.2: Initialization times of PS-MW components.

Transfer Mode	Value [μs]
Mailbox only	1.04
With PS-MW	1.21

Table 4.3: Message transfer times for plain mailbox communication and for a transfer using our publisher-subscriber middleware.

4.2.2 Initialization and Communication Overhead

As the PS-MW adds some management overhead to the system we measured the times spent in the initialization phase of the PS-MW at system start-up, i.e., initialization of the PSM and the DS. Additionally, PO and SO creation and registration times were examined. The results for the different PS-MW objects are collected in Table 4.2. Initialization of the PSM and the DS is performed once at system startup. Creation and registration is performed whenever an according object is instantiated.

To assess the overhead in message transfer time when employing our lightweight PS-MW we have performed some simple experiments. Several different scenarios have been examined. First, the time spent for a plain mailbox communication between two tasks was measured. After that the same tasks have been adapted to use the PS-MW. That is, they communicated via a PO at the sender and a SO (including a mailbox) at the receiving task. In this experiment the time spent from sending the message at the publisher until it was received at the subscriber was measured. Note that in this scenario one publisher with exactly one connected subscriber was examined, i.e., a unicast communication scheme. All these took only tasks on the same DSP into account. The results are summarized in Table 4.3. The overhead in this simple configuration amounts to 16.35% compared to simple mailbox transfers.

In another scenario we examined the multicast communication scheme, i.e., one publisher with several subscribers connected to it. The significant time measure in this case is the overall time needed to transfer the published message to all subscribed tasks. Again, only tasks on the same DSP were considered. It can be seen

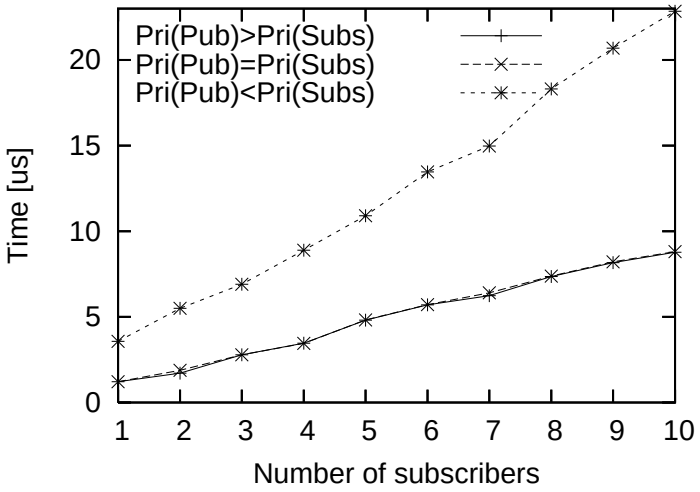

Figure 4.4: Transfer time in a multicast scenario increases depending on the number of subscribers and the priorities of publisher and subscriber tasks (denoted "Pri(Pub)" and "Pri(Subs)").

Number of SOs	**Transfer time per SO** [μs]		
	$P_{PO} > P_{SO}$	$P_{PO} = P_{SO}$	$P_{PO} < P_{SO}$
1	4.72	5.72	4.39
2	3.87	6.30	5.11
3	4.00	6.54	4.51
6	5.59	6.98	4.13

Table 4.4: Message transfer time from a single publisher to multiple subscribers depending on the number of subscribers and the task priorities. P_{PO} and P_{SO} denote task priorities of the publisher and the subscribers, respectively.

from Figure 4.4 that transfer time increases almost linearly with the number of subscribers.

The influence of the scheduler of the DSP/BIOS real-time operating system was examined by measuring message transfer times depending on the task priorities of publisher and subscriber tasks. Table 4.4 summarizes the results of these measurements and Figure 4.4 illustrates that transfer time is almost equal when the publisher and the subscriber have the same priority or the publisher has the highest priority. When the subscribers have the highest priority the transfer time increases significantly. This is due to additional task switches when subscribers block on mailboxes.

There are slight fluctuations depending on the number of overall tasks running on the DSP and their according priorities. That is due to internal management structures of the DSP/BIOS scheduler. Fortunately, the message transfer time per subscriber is relatively constant with respect to the number of subscribers and the task priorities.

Number of SOs	Transfer overhead [μs] 2 DSPs	3 DSPs	4 DSPs
1	3.49	- -	- -
2	4.69	5.24	- -
3	5.91	6.44	7.49

Table 4.5: Message transfer overhead time for publisher and subscribers residing on different DSPs. Overhead is given compared to direct PCI transfers without the PS-MW.

In another experiment the transfer times between tasks on different DSPs have been analyzed. The results are summarized in Table 4.5. The overhead in this case stems from the indirection in the involved MAOs and the proxy PO as well as the proxy SOs. It can be seen from the table that multiple subscribers on the same remote DSP yield less overhead than if they all reside on different DSPs. This is due to less management overhead in the target MAO. Also note that data is transferred only once to each DSP even if there are multiple subscribers for that data on the DSP.

4.3 Evaluation of the Fault Tolerance Architecture

Besides the PS-MW as the core of the software framework also the fault-tolerance architecture (FTA) has been evaluated. For simplicity the performance of the FTA was evaluated by experiments with dedicated test algorithms instead of real video analysis algorithms. A test algorithm (TA) is a piece of code suited for the framework that mimics the behavior of a surveillance algorithm. Its output is faked by a parameterizable data generator. For example, a test algorithm for traffic statistics outputs values for average vehicle speeds, the number of vehicle passed by and the like without any real video analysis. To test the FTA the output can be influenced to yield correct or erroneous data.

Therefore, a TA is well suited for fault injection experiments where faulty behavior of video analysis algorithms can be simulated. Fault injection is well suited for the evaluation of the FTA. No real faults have to be provoked and adequate test patterns can be applied. Thus, different fault scenarios can be examined easily.

Every TA is launched as a single task. It implements the standard interfaces needed to be deployed within the software framework. This includes also an appropriate algorithm description as it is required by the framework. The simulated analysis results are generally the same as from the real algorithm but can be influenced by the experimenter. Outputs of the TAs are, therefore, as meaningful as those of the actual algorithms with respect for their use with the FTA.

As with normal algorithms TAs also communicate using the publisher-subscriber middleware. In the prototype implementation, communication within a node is conducted via the local PCI bus whereas inter-node communication is IP-based via a standard Ethernet connection.

Algorithm	QoS Levels	Importance	Minimum QoS Level
MPEG-4 Encoder (MPEG)	Q_1, Q_2, Q_3	5	Q_2
Stationary Vehicle Detection (SVD)	Q_1, Q_2	2	Q_2
Traffic Statistics (STAT)	Q_1, Q_2	1	—

Table 4.6: Algorithms and their attributes for the example traffic surveillance application.

To evaluate the fault-tolerance architecture two key metrics are used as the evaluation criteria:

- the time elapsed to fault recognition and
- the time required for the execution of counter measures.

In order to demonstrate the FTA's ability to detect faults and to illustrate its reactions two example fault scenarios are presented in the following:

1. Scenario 1: Inconsistent observations of algorithms on different nodes, and
2. Scenario 2: A crashed DSP.

The surveillance setting assumed for the two scenarios comprises three smart camera nodes N_{i-1}, N_i, N_{i+1} along a highway where the cameras are equipped as the prototype described in Section 3.1. It is the assumed application design that three algorithms run on each camera to observe the scene. The algorithms and their attributes of this example application are listed in Table 4.6.

The importance measure of an algorithm denotes its value for the application. That is, an algorithm with a higher importance number provides more value to the application as one with a lower importance. Therefore, algorithms with lower importance figures are degraded first. Note that in general only the relative value of two algorithm's importance is important. But it is advisable that there is one maximum value which implies that algorithms of this maximum importance are absolutely required to fulfill application requirements. That is, if an algorithm of maximum importance has to be shut down due to fault conditions the application does not fulfill its requirements any more. This case then is not a graceful degradation any more. At best it can be seen as best effort operation that has to be treated by external intervention. Currently, having a MPEG-4 stream is considered as the minimum requirement for a video surveillance system as it allows at least human analysis if all other algorithms fail. It is also important that a minimum QoS level is specified for maximum importance algorithms so that it is clear which quality is needed for analysis by the operator.

Note also that importance values are fixed in this example. Nevertheless, the FTA allows for dynamic changes of each algorithm's importance to accommodate to different states observed in the environment. For example, in a traffic jam all algorithms might be reduced in their importance as it is not very likely that the

situation changes a lot. On the other hand side, after an accident the video stream, i.e., the MPEG, and the SVD are rather important to detect consecutive events.

The following considerations are based on performance numbers presented in earlier work [BRS04]. Over the PCI bus a transfer rate of 15 MB/s for communication between a DSP and the XScale is assumed as the lower bound for small messages. The size of the transferred messages is always 128 bits consisting of a 32-bit message ID and 96 bits of data. These include the algorithm's reference number and analysis results, e.g., the considered time interval and the number of vehicles counted during this interval. Therefore, every message sent over the PCI bus via the PS-MW was measured to result in an average transfer time of $t_{msg,PCI} = 0.031ms$.

4.3.1 Scenario 1: Inconsistent Observations

Given are three camera nodes N_{i-1}, N_i, N_{i+1} along a highway. Node N_i is observing an area characterized by stop-and-go traffic and it is in normal mode. It hosts an MPEG-4 encoder (MPEG) on one DSP and a stationary vehicle detection (SVD) on the second DSP. The SVD on node N_i is faulty and, therefore, does not detect any stationary vehicles during time interval t. The two neighboring nodes N_{i-1} and N_{i+1}, however, register a number of x and y stationary vehicles during time interval t, respectively.

System Response to Scenario 1

As node N_i's neighbor's observations are not consistent with those of node N_i this indicates a malfunction as it is not very likely to have stop-and-go traffic in two medium sized regions but neither passing nor stationary vehicles in between. Evidently, the voter's output does not match the SVD's output and the NSM indicates a malfunction of the SVD. The NSM instructs the migration and dynamic loading facility (MDL) to reload the SVD. It sets the node to inconsistent observation mode and sends this information to the logger. The SVD is reloaded and initialized. As most problems with algorithm's detection results are due to transient buffer problems it is likely that the re-initialization solves the problem and the algorithm works properly again. If the problem persists the algorithm has to be removed and the operator has to be notified. From tests with real algorithms it was found that detection problems arise almost periodically after some time. Note that the reload strategy allows normal operation at least in between re-initialization intervals. Even if this interval is long enough so that the algorithm is not completely removed the log allows to identify a recurring problem with the algorithm.

Time to Fault Detection The SVD sends its output every two seconds to the ACM and the voter, respectively. The voter's output is sent to the NSM, thus, two messages have to be sent and the first message is sent τ seconds after the occurrence of the fault. The voter's output reaches the NSM within a time interval of

$$t_{detection} = \tau + 2 \cdot t_{msg,PCI} = \tau + 0.062ms \tag{4.1}$$

Algorithm	QoS Level	QoS Description	CPU [MIPS]	RAM	
				internal	external
MPEG-4	Q_1	PAL 20 fps	2840	400 kB	0
MPEG-4	Q_2	PAL 10 fps	1920	400 kB	0
SVD	Q_1	CIF 12 fps	3600	500 kB	17 MB
SVD	Q_2	QCIF 12 fps	900	330 kB	4 MB

Table 4.7: Resource requirements for surveillance tasks according to [BBRS04].

where $\tau \leq 2s$ is the interval of alive messages specified in the framework.

Time Required for Counter Measures To handle the problem the system's reaction results in the sending of one more message to the MDL to reload the SVD. The time $t_{reload,SVD}$ required for reloading the SVD was measured to be 46 ms. Additional time $t_{reg,SVD} = 1\ ms$ for registering and starting the algorithm has also to be considered. Initialization of the SVD takes approximately 10 frames which corresponds to $t_{init,SVD} = 500\ ms$ in case of QoS level Q_2 with 10 fps. The total time spent on counter measures adds up to

$$t_{counter} = t_{msg,PCI} + t_{reload,SVD} + t_{reg,SVD} + t_{init,SVD} \quad (4.2)$$

$$t_{counter} = 0.031ms + 46ms + 17ms + 500ms \quad (4.3)$$

$$t_{counter} = 563.031ms \quad (4.4)$$

It can be seen from the above result that the overhead of the FTA is negligible compared to the algorithm-specific re-initialization times. Of course, algorithms with less initialization time result in less out time of the service in case of necessary reconfiguration.

4.3.2 Scenario 2: DSP Crash

In this scenario the MPEG encoder operates at QoS level Q_1 and an importance of 5 on DSP 1. Additionally, the SVD runs with QoS level Q_1 and an importance value of 2 on DSP 2. Then the DSP 2 crashes and cannot be rebooted so that the system has to proceed with only one remaining DSP.

System Response to Scenario 2

As the network processor maintains a list of currently active algorithms along with their importance values and DSP assignments the node state manager (NSM) can determine that the SVD algorithm is missing on the node.

This is because DSP 2 has not reacted to polling from the MDU for one second. A message is sent to the NSM informing that DSP 2 crashed. As resources demanded by the MPEG encoder and the SVD on the highest QoS level exceed the remaining DSP's computational power of 4800 MIPS (cf. Table 4.7) the NSM has to reconfigure the system. Note that image scaling and the shutter control for the image sensor takes approximately 1900 MIPS which is also considered by the NSM.

Since the SVD has the lower importance the NSM calculates whether it is possible to have the MPEG encoder run on QoS level Q_1 and the SVD on QoS level Q_2. Hence this is not feasible due to the above-mentioned overhead of 1900 MIPS the NSM subsequently determines that it is possible to run the MPEG encoder on Q_2 in combination with the SVD on Q_2. In that way the node chooses to gracefully degrade the QoS as opposed to a degradation of the service availability. The NSM instructs the MDL to load the SVD onto DSP 1, the MPEG encoder's QoS level is adjusted, and relevant information is sent to the logger. When loading of the SVD onto DSP 1 is finished the procedure is complete.

Time to Fault Recognition The time elapsed until the fault is recognized is primarily determined by the polling interval $t_{polling}$. By adding the transfer time of the notification message $t_{msg,PCI}$ from the polling interface results in a detection time $t_{detection}$ of

$$t_{detection} = t_{polling} + t_{msg,PCI} \quad (4.5)$$
$$t_{detection} = 1000ms + 0.031ms \quad (4.6)$$
$$t_{detection} = 1000.031ms. \quad (4.7)$$

Time Required for Counter Measures Again the measures for $t_{reload,SVD}$, $t_{reg,SVD}$, and $t_{init,SVD}$ of the SVD introduced in Section 4.3.1 can be used to compute the time for handling the problem. Additionally, the times for readjusting the MPEG-4 encoder's QoS level and the time $t_{adapt,MPEG}$ for the encoder adapting to the new QoS level have to be considered. The MPEG needs only one frame for adaptation which corresponds to $t_{adapt,MPEG} = 100ms$, respectively. Furthermore, two message sending times are involved in the handling of this scenario. First, the MDL has to be notified to reload the SVD. Second, the MPEG encoder has to be commanded to switch to QoS level Q_2. Therefore, the time for necessary reconfigurations to handle the detected problem computes to

$$t_{counter} = 2 \cdot t_{msg,PCI} + t_{reload,SVD} + t_{reg,SVD} + t_{init,SVD} + t_{adapt,MPEG} \quad (4.8)$$
$$t_{counter} = 0.062ms + 46ms + 17ms + 500ms + 100ms \quad (4.9)$$
$$t_{counter} = 663.062ms. \quad (4.10)$$

4.3.3 Summary

Both of the above scenarios show that the detection and reconfiguration overhead is dominated by algorithm-specific initialization times. Current algorithm implementations often rely on building some kind of models of the scene. The quality of the analysis depends strictly on the quality of the models. Therefore, many frames are used to build-up the models before actual analysis is performed. The encoder algorithms are better in this respect as they do not rely on sophisticated scene models.

The presented brief results are based on quite restrictive figures for communication times. That is, typically communication is much faster over the PCI bus. But

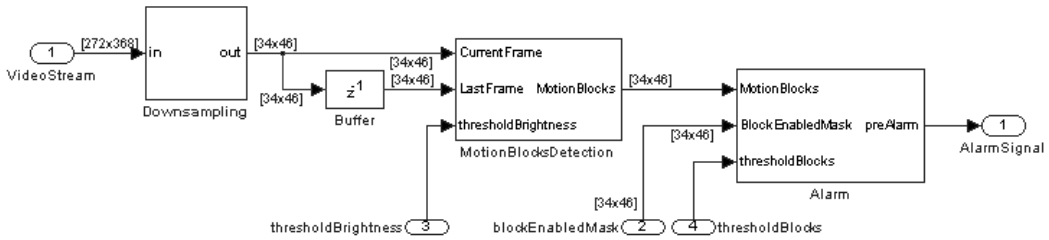

Figure 4.5: The Simulink model of the motion detection algorithm as it was used for the evaluation experiments.

to have some upper limit of the detection times the minimum measured PCI speed was considered.

4.4 Evaluation and Discussion of Model-Based Development of Video Analysis Algorithms using Simulink

Experiments were conducted using the Matlab/Simulink R14 modeling environment (changed from R13 to R14). A Texas Instruments C6416 DSP Starter Kit together with the Code Composer Studio v2.21 IDE (CCS) was used as the development environment. The following four different implementations of a motion detection algorithm (MD) with different levels of optimization were profiled and compared:

i Reference. A manually coded, hand-optimized C implementation exploiting the hardware capabilities of the TI C64x DSP.

ii Unoptimized model. The implementation synthesized from an ad hoc model without special optimization.

iii Generic optimizations. An implementation synthesized from a model where generic optimizations were applied. Target independent ANSI-C constructs were integrated. Substantial improvements are achieved by extracting iterative parts of the model and put them into custom blocks implemented in C. Nesting loops to support compiler optimizations is a very promising approach here.

iv Target-specific optimizations. An implementation synthesized from a model where target specific custom blocks were used. Adapted code segments from the manually coded reference (i) written in C were integrated.

The model of this simple motion detection algorithm used for the evaluation is presented in Figure 4.5

For the tests a video format with a resolution of 368 x 272 pixels was used. A model was created for each case listed above. From each model code was generated and imported to CCS for profiling. Profiling is also supported in the Simulink

Module	Reference implementation (i)	Non-optimized model (ii)	Generically optimized model (iii)	Target-specific optimized model (iv)
Downsampling	628752	27665088	5030016	639014
Buffering/Unbuffering	1408	37876	1592	1592
Sum of absolute differences	1520	85024	26832	1536
Generate pre-alarm	432	44116	3296	448
Overall	632112	27832104	5061736	642590

Table 4.8: Profiling results in CPU cycles split among different parts of the algorithm. Ratings for all implementation variants are collected.

	Reference implementation (i)	Non-optimized model (ii)	Generically optimized model (iii)	Target-specific optimized model (iv)
Code size	148	345	346	346
Data memory	111	115	116	116

Table 4.9: Memory consumption in KB. The code size and the memory required for data storage and buffers are summarized for each implementation variant.

environment but CCS offers more control over the profiling process. Results from CPU load profiling are summarized in Table 4.8.

Data memory consumption of the different examined cases differs only slightly (cf. Table 4.9). Input video frame buffers make up about 85% of the total data memory consumption. The rest is used for past frames for frame differencing. The synthesized executables require more than twice the program memory of the reference implementation.

Code generated from the initial unoptimized model (ii) was 44 times slower than the reference (i). With generic optimizations (iii) the execution time could be reduced to 18% compared to the unoptimized model (ii). But this code was still eight times slower than the reference (i). The second level of optimization utilized embedded C segments from the reference as a simple target specific optimization. Now the obtained performance was similar to that of the reference (i). Only an overhead of about two percent is still imposed by the modeling environment. With generic optimization (iii) the code was about 7.88 times slower than the code with target specific optimizations (iv).

An important reason for the poor performance of the code generated from the unoptimized model (ii) is that generated code often contains multiple sequential loops. In the manually optimized version operations are compacted to a single loop. Compilers are then able to better parallelize instructions resulting in a performance gain. Additionally, blocks that are only used to interface special function blocks in the model often result in redundant code in the synthesis process.

Another reason for performance losses is the inefficient use of the memory subsystem. Even simple functional blocks such as buffering and unbuffering can lead to degraded performance of synthesized code. Using DMA features of the target processor could substantially increase data transfer performance. Especially, video analysis algorithms benefit from DMA because a lot of data has to be transferred from and to memory. It is almost always possible to fully load the CPU while DMA transfers are performed in the background.

It is also shown that the use of specialized DSP instructions improves performance. In Simulink such specialized code is incorporated into the model via custom blocks that directly make use of target specific instructions or call into optimized (assembly) libraries. Of course, there is a one-time overhead for implementing such custom blocks. However, they can then be easily reused in similar projects without substantial effort.

These code profiling experiments of a motion detection algorithm indicate that model-based design is a promising approach for embedded video surveillance applications. Reusability and maintainability of the software are promoted by the high-level design. However, complex embedded video surveillance applications with their resource limitations require rather efficient algorithm implementations that often cannot be satisfied by code generation from simple models. Therefore, model optimization is necessary because optimizing the generated code would corrupt the model-based development flow.

In Simulink custom blocks can be used for model optimizations. Generally, there are two major optimization levels. First, generic optimization concentrates on hardware independent model modifications. Target specific modifications, on

the other hand, are not easily portable to different hardware any more. Nevertheless, they yield the most performance improvements in the synthesized code. To maximize the use of custom optimizations a domain specific library of custom blocks can be created and reused for similar projects.

Future work in this area may concentrate on the use of the code generation framework of Simulink to synthesize algorithmic code for an embedded smart traffic surveillance camera [BBRS04]. The code generation templates have to be adapted so that generated code can be used as dynamically loadable modules in the software framework that is currently developed.

Chapter 5

Conclusion

There is a strong trend towards intelligent infrastructures to ease everyday live. In traffic surveillance, e.g., networks of embedded intelligent cameras are introduced. These cameras provide on-site video analysis to detect dangerous traffic situations and compute traffic statistics that can be used for traffic management. High performance embedded computing platforms are required to provide enough computing power for the video analysis algorithms. In previous work [BBRS04] we developed the ***Smart****Cam* that is a heterogeneous multi-processor prototype of an embedded smart camera. It comprises a network processor and several DSPs.

Because of the limited resources of the embedded platform it is not possible to run all analysis algorithms simultaneously. Therefore, all algorithms are loaded and unloaded on demand at runtime. To support communication between these dynamically changing algorithms on the DSPs a middleware layer that supports loose coupling of tasks is required. Furthermore, a network of smart cameras has to be robust and tolerant with respect to algorithm failures and other unexpected behavior. As video surveillance applications are also getting increasingly complex it is necessary to introduce new development paradigms to keep the time-to-market low. The software framework for a smart camera has, therefore, to allow for high-level development.

In this work a real-time publisher-subscriber middleware (PS-MW) for the ***Smart****Cam* platform is described. It is a very light-weight architecture that supports loose coupling of tasks in the given dynamic application environment. By introducing only minimal indirection it also provides little transfer time overhead. Transparent communication within a single DSP and between different DSPs via the local PCI bus is supported. To abstract from the PCI bus a special proxy mechanism is used. An experimental evaluation on the ***Smart****Cam* prototype shows that our PS-MW has a memory footprint of as little as 15.78 KB. Transfer time overhead in case of communication between tasks on the same DSP is only 16.35%. In a multicast scenario the PS-MW scales well in that the transfer time per subscriber is almost constant with respect to the number of subscribers. Due to the efficient abstraction mechanism the message transfer time overhead compared to a direct PCI transfer is also in the order of several microseconds.

The software framework comprises also a fault-tolerance architecture that provides monitoring and diagnosis, as well as graceful degradation to cope with

unexpected behavior. We concentrate on software fault-tolerance because hardware redundancy results in prohibitive device cost. The dynamic reconfigurability achieved by the PS-MW serves as the basis for the graceful degradation scheme. For detection of problems application-specific knowledge is exploited. The redundancy of several cameras observing overlapping or adjacent scenes is exploited. Comparing corresponding algorithm results from neighboring cameras allows to detect problems with algorithms.

Additionally, the framework provides a component model for DSP algorithms. This set of dedicated interfaces and component descriptions is a prerequisite to use high-level development paradigms for algorithm development. An improved development process is presented that reduces overall development time. Traditionally, there are different roles for developing smart cameras. Especially, algorithm developers are experts for video analysis but are often not familiar with the target hardware platform. Thus, the development process is rather inefficient because platform experts have to port algorithms from high-level modeling environments to the target hardware. By using these high-level algorithm models directly for code synthesis overall development time is reduced. As synthesized code has to be efficient on the target hardware this work presents an evaluation of the performance of code synthesized from Matlab/Simulink as an example model-based development environment for embedded DSP systems.

5.1 Future Work

Advanced middleware services to improve QoS and power management as well as additional fault tolerance mechanisms should be further investigated. The goal is to achieve a system of autonomously operating smart cameras for pervasive intelligent infrastructure applications.

Therefore, also more algorithms are needed that take advantage of multiple camera systems. That is, algorithms should be designed to intensively exploit information from neighboring nodes. Robustness of algorithms, as well as overall system reliability could be well improved.

Multiple sensor integration is also an area of current research in the ***Smart****Cam* project. By using information from many different sensors it is likely also to improve detection accuracy.

For future applications in rural areas and also for easier installation wireless network communication for a network of smart cameras is currently explored. The aim is to provide robust and energy-aware wireless communication among a dense network of many ***Smart****Cams*. It is an important aspect that the software framework supports these new communication paradigm. Hence, challenges such as hidden station and other problems of multiple-node highly distributed systems have to be taken into account. Fortunately, the flexible design of the framework allows for relatively easy extensions for additional communication means.

In some application domains it might be of interest to have a high-reliability version of a smart camera where hardware cost unit price are not that important design considerations. Then it could be investigated to include automatic replica-

tion mechanisms into the framework. It can be imagined that crucial framework components are automatically replicated and synchronized to provide a highly reliable computing environment. Algorithms could also be replicated. With algorithms it would make sense to let the application decide on the replication scheme to account for different operational modes.

Bibliography

[ALRL04] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 2004.

[BBRS04] Michael Bramberger, Josef Brunner, Bernhard Rinner, and Helmut Schwabach. Real-Time Video Analysis on an Embedded Smart Camera for Traffic Suveillance. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 174–181, 2004.

[BDJ+06] Michael Bramberger, Andreas Doblander, Milan Jovanovic, Andreas Klausner, Arnold Maier, Bernhard Rinner, and Allan Tengg. Embedded Smart Cameras as Key Components in Reactive Sensor Systems. In *Proceedings of the International Conference on Cognitive Systems with Interactive Sensors*, Paris, France, March 2006.

[BDM+06] Michael Bramberger, Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Distributed smart cameras for surveillance applications. *Computer*, 39(2):68–75, February 2006.

[BDR+99] Douglas Bernard, Richard Doyle, Ed Riedel, Nicolas Rouquette, Jay Wyatt, Mike Lowry, and Pandurang Nayak. Autonomy and software technology on NASA's Deep Space One. *IEEE Intelligent Systems*, 14(3):10–15, May-June 1999.

[BGK+06] Krishnakumar Balasubramanian, Aniruddha Gokhale, Gabor Karsai, Janos Sztipanovits, and Sandeep Neema. Developing applications using model-driven design environments. *Computer*, 39(2):33–40, February 2006.

[BN03] Tom D. Bracewell and Priya Narasimhan. A middleware for dependable distributed real-time systems. In *Proceedings of the Joint Systems and Software Engineering Symposium*, April 2003.

[Bra05] Michael Bramberger. *Distributed Dynamic Task Allocation in Clusters of Embedded Smart Cameras*. PhD thesis, Institute for Technical Informatics, Graz University of Technology, Graz, Austria, June 2005.

[BRS04] Michael Bramberger, Bernhard Rinner, and Helmut Schwabach. An Embedded Smart Camera on a Scalable Heterogeneous Multi-DSP System. In *Proceedings of the European DSP Education and Research Symposium*, November 2004.

[BRS05] Michael Bramberger, Bernhard Rinner, and Helmut Schwabach. A Method for Dynamic Allocation of Tasks in Clusters of Embedded Smart Cameras. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, pages 2595–2600, Hawaii, U.S.A., October 2005.

[BSGR03] Christian Becker, Gregor Schiele, Holger Gubbles, and Kurt Rothermel. BASE—a micro-broker-based middleware for pervaisve computing. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pages 443–451. IEEE, March 23-26 2003.

[BWGS03] Krishnakumar Balasubramanian, Nanbor Wang, Chris Gill, and Douglas C. Schmidt. Towards composable distributed real-time and embedded software. In *Proceedings of the 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 226–233, Guadalajara, Mexico, January 2003.

[CBCP01] Michael Clarke, Gordon S. Blair, Geoff Coulson, and Nikos Parlavantzas. An Efficient Component Model for the Construction of Adaptive Middleware. In R. Guerraoui, editor, *Proceedings of the 2001 IFIP/ACM International Conference on Distributed Systems Platforms*, number 2218 in Lecture Notes in Computer Science, pages 160–178. Springer, 2001.

[Cou99] Geoff Coulson. A configurable multimedia middleware platform. *IEEE MultiMedia*, 6(1):62–76, January-March 1999.

[DeM95] Linda G. DeMichiel. The component object model specification. Technical report, Microsoft Corporation, November 1995.

[DeM02] Linda G. DeMichiel. Enterprise JavaBeans Specification Version 2.1. Technical report, SUN Microsystems, November 2002.

[DGRS05] Andreas Doblander, Dietmar Gösseringer, Bernhard Rinner, and Helmut Schwabach. An Evaluation of Model-Based Software Synthesis from Simulink Models for Embedded Video Applications. *International Journal of Software Engineering and Knowledge Engineering*, 15(2):343–348, April 2005.

[DMRS05a] Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Improving Fault-Tolerance in Intelligent Video Surveillance by Monitoring, Diagnosis and Dynamic Reconfiguration. In *Proceedings of the Third IEEE-Workshop on Intelligent Solutions in Embedded Systems, Hamburg, Germany*, pages 194–201, 2005. ISBN 3-902463-03-1.

[DMRS05b] Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Increasing Service Availability in Intelligent Video Surveillance Systems by Fault Detection and Dynamic Reconfiguration. In *Proceedings of the Telecommunications and Mobile Computing Workshop on Wearable and Pervasive Computing*, Graz, Austria, March 2005.

[DMRS06] Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. A Novel Software Framework for Power-Aware Reconfiguration in Distributed Embedded Smart Cameras. In *Proceedings of the 12th IEEE International Conference on Parallel and Distributed Systems*, volume 1, pages 281–288, Minneapolis, Minnesota, USA, July 2006. IEEE Computer Society.

[DMRZ06] Andreas Doblander, Arnold Maier, Bernhard Rinner, and Andreas Zoufal. An Efficient Middleware for Power-Aware Service Reconfiguration in Multi-DSP Smart Cameras. In *Proceedings of the 2nd IEEE International Conference on Information and Communication Technologies: From Theory to ApplicationsSoftware Engineering*, pages 1093–1094, Damascus, Syria, April 2006. IEEE.

[Dor03] Kevin Dorow. Flexible fault tolerance in configurable middleware for embedded systems. In *Proceedings of the 27th Annual International Computer Software and Applications Conference*, pages 563–569, 3-6 November 2003.

[DRTZ06a] Andreas Doblander, Bernhard Rinner, Norbert Trenkwalder, and Andreas Zoufal. A light-weight Publisher-Subscriber Middleware for Dynamic Reconfiguration in Networks of Embedded Smart Cameras. In *Proceedings of the 5th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems*, Madrid, Spain, February 2006. World Scientific and Engineering Academy and Society.

[DRTZ06b] Andreas Doblander, Bernhard Rinner, Norbert Trenkwalder, and Andreas Zoufal. A Middleware Framework for Dynamic Reconfiguration and Component Composition in Embedded Smart Cameras. *WSEAS Transactions on Computers*, 5(3):574–581, March 2006.

[FH03] Christof Fetzer and Karin Högstedt. Self-*: A data-flow oriented component framework for pervasive dependability. In *Proceedings of the Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 66–73. IEEE, 15-17 January 2003.

[FMR00] G. L. Foresti, C. Mähönen, and C. S. Regazzoni. *Multimedia video-based surveillance systems*. Kluwer Academic Publishers, 2000.

[FSF03] Joni Fraga, Frank Siqueira, and Fábio Favarim. An adaptive fault-tolerant component model. In *Proceedings of the Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 179–186, Capri Island, Italy, 2003.

[GHN03] Fabíola Greve, Michel Hurfin, and Jean-Pierre Le Narzul. OPEN EDEN: a Portable Fault Tolerant CORBA Architecture. In *Proceedings of the Second International Symposium on Parallel and Distributed Computing*, pages 88–95, 2003.

[Gös05] Dietmar Gösseringer. Evaluation of a Model-Based Design Approach for Embedded Image Processing Algorithms. Master's thesis, Institute for Technical Informatics, Graz University of Technology, 2005.

[Gro04] Object Management Group. *Common Object Request Broker Architecture: Core Specification (Version 3.0.3)*, chapter 23. Object Management Group, March 2004.

[Gro05a] Object Management Group. `http://www.omg.org/technology/documents/formal/components.htm`, November 2005.

[Gro05b] Object Management Group. `http://www.omg.org/`, 2005.

[HÅCT04] Hans Hansson, Mikael Åkerholm, Ivica Crnkovic, and Martin Törngren. SaveCCM—a component model for safety-critical real-time systems. In *Proceedings of the 30th EUROMICRO Conference*, pages 627–635, 2004.

[HC01] D. K. Hammer and M. R. V. Chaudron. Component-based software engineering for resource-constraint systems: What are the needs? In *Proceedings of the Sixth Internation Workshop on Object-Oriented Real-Time Dependable Systems*, pages 91–94, January 8-10 2001.

[HGC+00] K. H. Hong, W. S. Gan, Y. K. Chong, K. K. Chew, C. M. Lee, and T. Y. Koh. An integrated environment for rapid prototyping of DSP algorithms using Matlab and Texas Instruments TMS320C30. *Microprocessors and Microsystems*, 24:349–363, 2000.

[HLKK00] Joshua Haines, Vijay Lakamraju, Israel Koren, and C. Mani Krishna. Application-level fault tolerance as a complement to system-level fault tolerance. *The Journal on Supercomputing*, 16:53–68, 2000.

[Ins02] Texas Instruments. *TMS320 Algorithm Standard—Rules and Guidelines*. Texas Instruments, October 2002. Literature Number: SPRU352E.

[KBE99] Mieczyslaw M. Kokar, Kenneth Baclawski, and Yonet A. Eracar. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, 14(3):37–45, May-June 1999.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.

[Kim01] K. H. Kim. Middleware of real-time object based fault-tolerant distributed computing systems: Issues and some approaches. In *Proceedings of the Pacific Rim International Symposium on Dependable Computing*, pages 3–8, Seoul, Korea, December 2001.

[KKL04] George Kola, Tevfik Kosar, and Miron Livny. Phoenix: Making Data-intensive Grid Applications Fault-tolerant. In *Fifth IEEE/ACM International Workshop on Grid Computing*, pages 251–258, 2004.

[KL03] Jesung Kim and Insup Lee. Modular code generation from hybrid automata based on data dependency. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 160–168. IEEE, 27-30 May 2003.

[Kla06] Karima Klamminger. Design and implementation of a fault tolerance concept in a network of smart cameras. Master's thesis, Institute for Technical Informatics, Graz University of Technology, 2006. (to appear).

[KMGK03] Kerem Karaday, Vishal Markandey, Robert J. Gove, and Yongmin Kim. Strategies for mapping algorithms to mediaprocessors for high performance. *Micro*, 23(4):58–70, July–August 2003.

[KS99] Gabor Karsai and Janos Sztipanovits. A model-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):46–53, May-June 1999.

[KSLB03] Gabor Karsai, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, January 2003.

[KZSR01] Deepak R. Karuppiah, Zhigang Zhu, Prashant Shenoy, and Edward M. Riseman. A fault-tolerant distributed vision system architecture for object tracking in a smart room. In B. Schiele and G. Sagerer, editors, *Proceedings of the Second International Workshop on Computer Vision Systems*, volume 2095 of *Lecture Notes in Computer Science*, pages 201–219. Springer, 2001.

[Lad99] Robert Laddaga. Creating robust software through self-adaptation. *IEEE Intelligent Systems*, 14(3):26–29, May-June 1999.

[LLWO04] Chang Hong Lin, Tiehan Lv, Wayne Wolf, and I. Burak Ozer. A Peer-to-Peer Architecture for Distributed Real-Time Gesture Recognition. In *Proceedings of the 2004 IEEE International Conference on Multimedia and Expo*, pages 57–60, 2004.

[LWD+06] Chang Hong Lin, Wayne Wolf, Andrew Dixon, Xenofon Koutsoukos, and Janos Sztipanovits. Design and Implementation of Ubiquitous Smart Cameras. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, volume 1, pages 32–39. IEEE, June 2006.

[Lyu95] Michael R. Lyu, editor. *Software Fault Tolerance*. Number 3 in Trends in Software. John Wiley & Sons, January 1995.

[Mai06] Arnold Maier. *Dynamic Power-Aware Camera Configuration in Distributed Embedded Surveillance Clusters*. PhD thesis, Institute for Technical Informatics, Graz University of Technology, Graz, Austria, February 2006.

[MBC04] Rui S. Moreira, Gordon S. Blair, and Eurico Carrapatoso. Supporting Adaptable Distributed Systems with FORMAware. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*, pages 320–325, 23-24 March 2004.

[MCE02] Cecilia Mascolo, Licia Capra, and Wolfgang Emmerich. Mobile computing middleware. In Enrico Gregori, Giuseppe Anastasi, and Stefano Basagni, editors, *Advanced Lectures on Networking: NETWORKING 2002 Tutorials*, volume 2497 of *Lecture Notes in Computer Science*, pages 20–52. Springer, 2002.

[Mic05a] Microsoft. .Net Home Page. `http://www.microsoft.com/net`, November 2005.

[Mic05b] Sun Microsystems. `http://www.sun.com`, July 2005.

[MJO+05] Derek Messie, Mina Jung, Jae C. Oh, Shweta Shetty, Steven Nordstrom, and Michael Haney. Prototype of fault adaptive embedded software for large-scale real-time systems. In *Proceedings of th 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 498–505, April 2005.

[Mod06] Mihir Mody. XDAIS-DM (XDM): A step towards the "plug and play" architecture for multimedia codecs. TI Developer Conference, February 2006. `http://www-s.ti.com/sc/techlit/sprp496.pdf`.

[MPT04] Annukka Mäntyniemi, Minna Pikkarainen, and Anne Taulavuori. A framework for off-the-shelf software component development and maintenance. VTT Publications VTT-PUBS-525, VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FIN-02044 VTT, Finland, April 2004. ISBN 951-38-6368-7.

[MRS05] Arnold Maier, Bernhard Rinner, and Helmut Schwabach. A Hierarchical Approach for Energy-Aware Distributed Embedded Intelligent Video Surveillance. In *Proceedings of the IEEE/IFIP International Workshop on Parallel and Distributed Embedded Systems*, pages 12–16, Fukuoka, Japan, 2005.

[MRSS06] Arnold Maier, Bernhard Rinner, Wolfgang Schriebl, and Helmut Schwabach. Online Multi-Criterion Optimization for Dynamic Power-Aware Camera Configuration in Distributed Embedded Surveillance Clusters. In *Proceedings of the 20th IEEE International Conference on Advanced Information Networking and Applications*, pages 307–312, Vienna, Austria, April 2006.

[Nat06] National Instruments, Inc. National instruments website, September 2006. `http://www.ni.com/`.

[NGYS00] Balachandran Natarajan, Aniruddha Gokhale, Shalini Yajnik, and Douglas C. Schmidt. DOORS: Towards High-performance Fault Tolerant CORBA. In *International Symposium on Distributed Objects and Applications*, pages 39–48, 2000.

[NK02] Carsten Nitsch and Udo Kebschull. The use of runtime configuration capabilities for network embedded systems. In *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition*, page 1093. IEEE Computer Society, March 4-8 2002.

[NMMS00] N. Narasimhan, L.E. Moser, and P.M. Melliar-Smith. Transparent consistent replication of Java RMI objects. In *International Symposium on Distributed Objects and Applications*, pages 17–26, 2000.

[Obj02] Object Management Group. Minimum CORBA 1.0. `http://www.omg.org`, 2002.

[OGT+99] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, May-June 1999.

[PCSH99] Gerardo Pardo-Castellote, Stan Schneider, and Mark Hamilton. NDDS: The Real-Time Publish-Subscribe Middleware. `http://www.rti.com`, 1999. Real-Time Innovations, White paper.

[PM01] Esmond Pitt and Kathy McNiff. *Java.rmi: The Remote Method Invocation Guide*. Addison Wesley, 2001.

[Pop98] Alan Pope. *The CORBA Reference Guide: Understanding the Common Oject Request Broker Architecture*. Addison Wesley, 1998.

[Pul01] Laura L. Pullum. *Software Fault Tolerance Techniques and Implementation*. Artech House Publishers, 2001.

[QXcMw05] Yang Qun, Yang Xian-chun, and Xu Man-wu. A framework for dynamic software architecture-based self-healing. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–4, July 2005.

[RGS95] Ragunathan Rajkumar, Mike Gagliardi, and Lui Sha. The real-time publisher/subscriber inter-process communication model for distributed real-time systems: Design and implementation. In *Proceedings of the Real-Time Technology and Applications Symposium*, pages 66–75. IEEE, 15-17 May 1995.

[Sch02] Douglas C. Schmidt. Middleware for real-time and embedded systems. *Communications of the ACM*, 45(6):43–48, June 2002.

[Sch06] Douglas C. Schmidt. Model-driven engineering. *Computer*, 39(2):25–31, February 2006.

[Ses97] Roger Sessions. *COM and DCOM: Microsoft's Vision for Distributed Objects*. John Wiley & Sons, 1997.

[SKN03] Charles P. Shelton, Philip Koopman, and William Nace. A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems. In *Proceedings of the Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 156–163. IEEE, 2003.

[SLS+04] Suman Srinivasan, Haniph Latchman, John Shea, Tan Wong, and Janice McNair. Airborne traffic surveillance systems - video surveillance of highway traffic. In *Proceedings of the ACM 2nd international workshop on Video Surveillance and Sensor Networks*, pages 131–135, 2004.

[SM04] Tim Schattkowsky and Wolfgang Mueller. Model-based specification and execution of embedded real-time systems. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 1392–1393, 16-20 February 2004.

[SNT02] Diana Szentiványi and Simin Nadjm-Tehrani. Building and Evaluating a Fault-Tolerant CORBA Infrastructure. In *Proceedings of the Workshop on Dependable Middleware-Based Systems*, pages –, 2002.

[SNT04] Diana Szentiványi and Simin Nadjm-Tehrani. Middleware support for fault tolerance. In Qusay H. Mahmoud, editor, *Middleware for Communications*, pages 439–464. John Wiley & Sons Ldt., 2004.

[SS03] Minseok Song and Heonshik Shin. A QoS degradation policy for revenue maximization in fault-tolerant multi-resolution video servers. *IEEE Transactions on Consumer Electronics*, 49(2):392–402, May 2003.

[SSBG03] S. Sastry, Janos Sztipanovits, R. Bajcsy, and H. Gill. Scanning the issue—special issue on modeling and design of embedded software. *Proceedings of the IEEE*, 91(1):3–10, January 2003.

[ST03] Mercury Computer Systems and Thales. Light Weight CORBA Component Model. Technical report, Object Management Group, 2003.

[SV02] Alberto Sangiovanni-Vincentelli. Defining platform-based design, February 2002. EEDesign Magazine of EETimes (EEDesign.com).

[SVM01] Alberto Sangiovanni-Vincentelli and Grant Martin. A vision for embedded software. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 1–7, Atlanta, Georgia, USA, November 2001. ACM.

[Szy97] Clemens Szyperski. *Component Software—Beyond Object-Oriented Programming*. Addison Wesley Longman Limited, 1997.

[Tei05] Egon Teiniker. *A Novel Component Platform for Logistics Software Product Lines*. PhD thesis, Institute for Technical Informatics, Graz University of Technology, 2005.

[Tel06] Telelogic. Telelogic website, September 2006. `http://www.telelogic.com/`.

[The01] The Object Management Group. Real-Time CORBA 2.0. `http://www.omg.org`, September 2001.

[The06] The MathWorks, Inc. MathWorks Website, September 2006. `http://www.mathworks.com/`.

[TMK+02] Egon Teiniker, Stefan Mitterdorfer, Christian Kreiner, Zsolt Kovács, and Reinhold Weiss. Local components and reuse of legacy code in the CORBA component model. In *Proceedings of the 28th Euromicro Conference*. Institute for Technical Informatics, Graz University of Technology, IEEE, 2002.

[Tre06] Norbert Trenkwalder. Implementierung und Evaluierung eines Frameworks für DSP basierte Smart Cameras. Master's thesis, Institute for Technical Informatics, Graz University of Technology, 2006. (to appear).

[WGC+02] Michael Winter, Thomas Genßler, Alexander Christoph, Oscar Nierstrasz, Stéphane Ducasse, Roel Wuyts, Gabriela Arévalo, Peter Müller, Chris Stich, and Bastiaan Schönhage. Components for Embedded Software—The PECOS Approach. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 19–26. ACM, 2002.

[WH99] Michael W. Whalen and Mats P.E. Heimdahl. On the requirements of high-integrity code generation. In *Proceedings of the 4th IEEE International Symposium on High-Assurance Systems Engineering*, pages 217–224, 17-19 November 1999.

[Wha05] Whatis.com. `http://searchvb.techtarget.com/sDefinition/0,290660,sid8_gci211826,00.html`, July 2005.

[WOL02] W. Wolf, B. Ozer, and T. Lv. Smart cameras as embedded systems. *Computer*, 35(9):48–53, September 2002.

[WP99] David Wybo and David Putti. A qualitative analysis of automatic code generation tools for automotive powertrain applications. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, pages 225–230, Kohala Coast-Island of Hawaii, Hawaii, USA, August 1999. IEEE.