An Efficient Middleware for Power-Aware Service Reconfiguration in Multi-DSP Smart Cameras

Andreas Doblander, Arnold Maier, Bernhard Rinner
Institute for Technical Informatics,
Graz University of Technology
Inffeldgasse 16/1, 8010 Graz, Austria
E-Mail: {doblander, maier, rinner}@iti.tugraz.at

Andreas Zoufal
Video and Safety Technologies,
ARC Seibersdorf research
2444 Seibersdorf, Austria
E-Mail: andreas.zoufal@arcs.ac.at

Abstract

Traffic video surveillance applications are increasingly maturing to intelligent infrastructures based on networks of embedded smart cameras. These smart cameras provide on-board real-time video analysis and streaming. In previous work a hardware platform for a smart camera was proposed. This work now presents a software framework for power-aware service reconfiguration in embedded multi-DSP smart cameras. An efficient publisher-subscriber communication scheme together with dynamic loading capabilities enable reconfiguration at runtime. Depending on the observed scenario an optimal camera configuration is chosen. A multi-criterion optimizer implemented as a genetic online algorithm continuously computes optimal algorithm configurations by minimizing power-consumption and maximizing quality-of-service. Based on a cost-function an appropriate configuration is chosen among the set of Pareto-optimal solutions.

1. Introduction

Intelligent video surveillance (IVS) based on smart cameras is getting more and more attention in the industry. Smart cameras [15] are equipped with high-performance on-board 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.

Typically, the cameras have to execute demanding video processing and compression algorithms [9]. 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 we have designed a smart camera—the *SmartCam* [5]—as a fully embedded system. 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).

Flexibility of algorithm configurations, i.e., how tasks are composed to build the application, as well as scalability concerning the number and the different types of employed surveillance tasks have to be concerned. Low power consumption and efficient resource utilization are prerequisites in an embedded setting. In a video surveillance application there are potentially many different video analysis algorithms intended to run. However, in a typical setting not all algorithms are required at all times. In dynamically reconfiguring the set of running algorithms, i.e., services, resources can be optimally utilized while minimizing power consumption and maximizing Quality-of-Service (QoS). A software framework for such dynamic power-aware service reconfiguration is devised in this work.

2. Related Work

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. Unfortunately, middleware technologies from general purpose computing, such as Microsoft DCOM [1], Java RMI [3] and OMG CORBA [2] are not suitable for very resource limited devices [11] as smart cameras are.

An interesting approach is the "Self-*" architecture [8]. It is a data-flow oriented and component-based middleware framework that is aimed for dependable pervasive computing systems.

The notion of *runtime configuration capable embedded systems* by Nitsch and Kebschull [13] 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.

A popular inter-process communication model for real-time systems is the realtime publisher/subscriber model (RT-PS) [14]. 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 our software framework.

Minimizing the power consumption and maximizing service-levels in IVS are—similar to a lot of other real-world problems—two conflicting objectives for optimization and therefore are referred to as *multi-criterion optimization (MCO)* problems [6]. Solving a MCO-problem does not result in a single scalar that represents an optimal value but in a set of several so called *non-dominated* solutions (also referred to as *Pareto-optimal* solutions). However, none of the Pareto-optimal solutions is "better" than another one in general but only in at least one criterion.

So called evolutionary approaches include genetic algorithms (GAs) that are used to solve MCO problems. GAs are heuristic search algorithms that are based on the evolutionary ideas of natural selection and genetics. They

0-7803-9521-2/06/$20.00 ©2006 IEEE.

are an applicable and robust approach especially for MCO problems with a large and complex search space. All genetic algorithms are based on the same generic concept. They start by a random initial population and generate better populations in each iteration by the principles of mutation and crossover. Each individual gets tested if it fits as possible solution due to a given cost function of each objective function [12].

Each camera configuration corresponds to a certain utilization of hardware components. Therefore, the approach presented in this paper takes use of dynamic power management (DPM) in order to minimize the power consumption [4]. DPM is based on the observation that a lot of power is wasted because of system components that are fully powered up even if they are not in use.

3.System Architecture

3.1.SmartCam Platform Overview

In previous work [5] we introduced our *SmartCam* as a prototype for an embedded smart camera for video surveillance. It has been designed as a low-power, high-performance embedded system comprising (i) the *sensing unit*, (ii) the *processing unit*, and (iii) the *communication unit*.

A CMOS image sensor is the heart of the sensing unit. It delivers images with VGA resolution and up to 30 frames per second. The processing unit can be equipped with up to ten TMS320C64x DSPs from Texas Instruments to adapt computing performance to the requirements of the real-time video analysis and compression tasks targeted for the smart camera. An aggregate performance of up to 80 GIPS can be delivered by the DSPs while keeping the power consumption low.

As the central part of the communication unit a network processor (Intel XScale) establishes the internal and external communication. Internal communication is performed via the PCI bus between either the DSPs or the DSPs and the network processor. IP-based external communication is provided via Ethernet or GSM/GPRS.

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 1.

The SC-FW that is illustrated in the left part of Figure 1 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. To further ease application development the SC-FW on the XScale is running on top of a standard LINUX kernel.

The DSP-FW, as indicated in the right part of Figure 1, runs on every DSP in the system. The main purposes of the DSP-FW 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-FW is the publisher-subscriber subsystem. The DSP-FW is built upon the DSP/BIOS real-time operating system from Texas Instruments.

All video analysis algorithms and several framework components of the DSP-FW can be loaded and unloaded at runtime by the *Dynamic Loader* module. Actually, only modules of the DSP-FW in dark shade in Figure 1 have to be available at startup. All other components can be dynamically loaded at runtime.

3.2.Publisher-Subscriber Subsystem

The publisher-subscriber architecture is an integral part of the DSP-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. To overcome these limitations a publisher-subscriber middleware layer (PS-MW) has been introduced [7].

Communication between algorithms on the same DSP an operating system mechanism called mailbox is employed. Mailboxes provide buffered communication and also allow for synchronization as tasks are 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 2 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 *publisher-subscriber manager* (PSM) is the authority where algorithms can register as data providers or data consumers. As algorithms can reside on different DSPs within a *SmartCam* it is also necessary that each PSM can discover services that have registered with a different

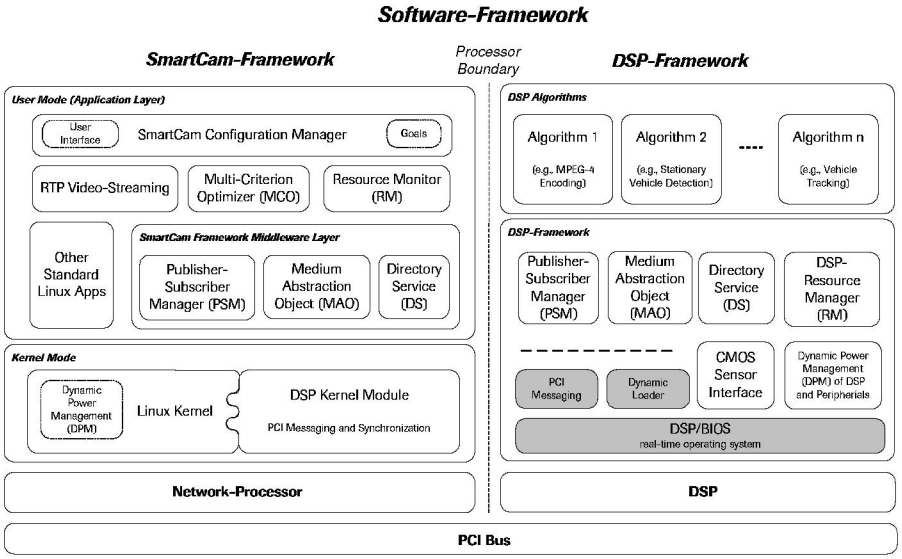

Figure 1. The overall software architecture of our smart camera. In the left part of the figure the so-called SmartCam-Framework is illustrated while the right part shows the so-called DSP-Framework.

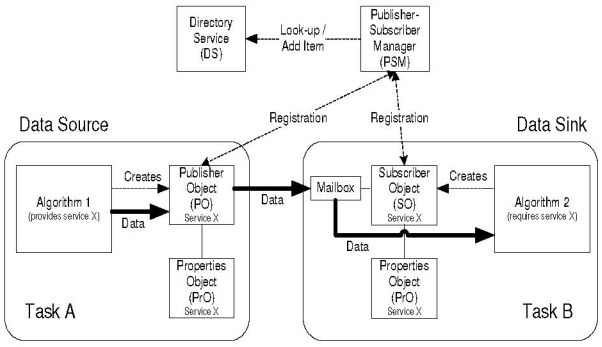

Figure 2. Principle relations between objects of the publisher-subscriber architecture. Only local connections within a single DSP are sketched.

PSM. Therefore, the network processor also hosts a PSM that relays service requests between PSMs on different DSPs. *Properties objects* (PrO) are used to describe published data and subscriptions in detail. Based on this information publishers and subscribers are able to discover each other and to connect at runtime. It is the responsibility of every algorithm to provide this information when it registers a service with the PSM. A *publisher object* (PO) is instantiated by every task that provides data services while tasks that require a data service of another algorithm instantiate a *subscriber object* (SO). These modules are communicating with the PSM to establish algorithm connections.

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, i.e., the PCI bus, 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.

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 the MAO instantiates a proxy SO and on the DSP with the data sink a proxy PO is created. These proxy objects behave like normal publishers and subscribers and enable transparent communication for the algorithm also between different DSPs. Remote subscriptions are limited by a resource manager to eliminate bus overloading so that real-time conditions are not violated.

For a convenient service discovery the PS-MW provides a directory service (DS) where all published services are listed together with their properties. To enable system-wide service discovery the DS is organized as a distributed service where all DSPs and the network processor each host a DS module. Entries of all DS modules are continually synchronized to ensure a consistent overall directory.

4. Dynamic Optimization of Service Configurations

The reconfigurability provided by the PS-MW is the basis for the power-aware reconfiguration in a cluster of smart cameras. A *configuration manager* (cf. Figure 1) on each camera is responsible for invoking an optimizer to find an optimal camera configuration which is then set by the configuration manager.

During operation, the system performs power-aware reconfiguration within a distributed cluster of embedded smart cameras by applying at least one of the following actions:

1. Adaptation of the IVS-services' QoS-levels
2. Reallocation of IVS-services, i.e., algorithms, within a smart camera ("intra-camera") and/or within multiple smart cameras ("intra-cluster")
3. Intra-camera DPM of hardware components (e.g., of processing units) and/or intra-cluster DPM of whole cameras (e.g., by temporary turning them off)

The problem of setting system configurations that are "optimal" in terms of QoS and power consumption can be seen as multi-criterion optimization problem. Therefore, a genetic online algorithm is employed that is suitable for solving the considered MCO problem on embedded smart

cameras under soft-realtime demands. It delivers a set of so called Pareto-optimal cameras configurations with respect to a given cost model that contains individual costs for both QoS and its corresponding power-consumption for each service executed on a smart camera. Figure 3 shows the a possible camera reconfiguration by switching to a configuration with less power costs but only a small QoS degradation.

The genetic algorithm can get triggered during operation by (1) the onboard configuration manager, or by (2) remote requests from other cameras. In both cases, the algorithm's input data needs to specify the optimization's search space and the cost functions for both optimization objectives. The search space is defined by the number of services s that are executed in up to q_s different QoS-levels on up to r hardware resources (i.e., processing units).

Equation 1 formulates the cost function for power consumption on a smart camera:

$$Costs_{Power} = \sum_{i=0}^{r} Costs_{Power}(i) \cdot \mu_i(s, q_s) \quad (1)$$

In this equation, $\mu_i(s, q_s)$ is an individual weighting function for the actual processor utilization that is caused by a service s in its QoS-level q_s on the considered hardware component i.

The weighting function $\mu_i(s, q_s)$ is likely to be defined as $'1'$ when the hardware component i is in its operational state that consumes the maximal amount of power, and it is defined as $'0'$ when the resource is turned off. Any resource utilization in between these boundaries depends on the relation of the components activity to idleness and is therefore a result of the amount and quality of the services that affect the device. Therefore, one objective function of the MCO aims at delivering configurations that tend to have costs toward zero ($Costs_{Power} = 0$).

Equation 2 formulates the cost function for service quality on a smart camera:

$$Costs_{Quality} = \sum_{i=0}^{s} Costs_{Quality}(i) \cdot \ _i(q_s, r) \quad (2)$$

In this equation $_i(q_s, r)$ is a weighting function for each QoS-level q_s that is currently executed and causes an utilization of a hardware resource r.

The weighting function $_i(q_s, r)$ corresponds to the q_s QoS-levels of each service i and is—in contrast to the costs for power consumption—defined as $'1'$ for the minimal QoS of the corresponding services and it is defined as $'0'$ if the best QoS is set.

The algorithm itself is based on an iterative genetic algorithm. It performs a randomized computation of feasible Pareto-optimal camera configurations and is described in more detail in [10].

Furthermore, the approach also covers the permanent sensing and analysis of application-and situation specific contextual information. The contextual information in an IVS-cluster is composed of numerous events that need to be sensed and analyzed. The events normally have an impact on the service-level and power consumption of a smart camera.

They may be generated within the camera itself, within the cluster (i.e., a camera causes an event that gets sensed by other cameras), or by direct user interaction. Since the information is brought into line with the current individual camera configurations, it allows making decisions about one of the following actions: (1) keep the current camera configuration, (2) (try to) relocate services within the cluster, (3) choose another (already pre-computed) configuration among the Pareto-set, or (4) (re-trigger) the optimizer due to a new search space.

An advantage of this method is a more autonomous system behavior instead of using pre-defined modes for reconfiguration [10]. As a consequence, reconfiguration is utilized on both cluster-and camera-level.

Table 1. Runtime memory requirements of middleware objects.

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

5. Experimental Results

5.1. Light-Weight PS-MW

Several experiments have been conducted to illustrate the performance of our framework for power-aware service reconfiguration. Experiments have been performed on our *SmartCam* prototype platform.

The basis of the platforms 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 Section 3.2.

The overall memory footprint is only 15.78 KB. It can be seen from Table 1 that the runtime memory consumption is also very low. Total memory consumption overhead, of course, depends on the number of publishings and subscriptions in the system as each of them requires a PrO and a PO or SO, respectively. In a typical setting with two algorithms per DSP and each algorithm providing one service and subscribes to one service this yields a total memory overhead of the middleware of 3.71 KB per DSP.

The PS-MW adds some management overhead to the

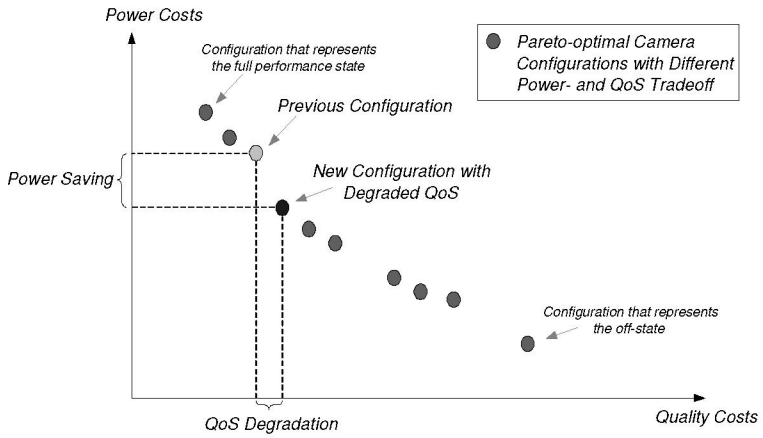

Figure 3. Changing a camera's configuration among the already pre-computed Pareto-set with different power and QoS tradeoffs.

system. So the initialization times of the PS-MW at system start-up and the PO and SO creation and registration times were examined. Initialization of the PSM and the DS is performed once at system startup. Creation and registration is performed whenever an according object is instantiated.Experiments showed that all of these times amount to a couple of microseconds.

To assess the overhead in message transfer time when employing our light-weight PS-MW we have performed some simple measurements. Communication time for a plain mailbox transfer between two tasks was measured to be 1.04 μs. The same tasks have been adapted to use the PS-MW which yielded a transfer time of 1.21 μs. Therefore, the overhead amounts to about 16.35 percent. 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.

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. Transfer time in this scenario increases almost linearly by approximately 1 μs for each subscriber. It is also interesting to note that transfer time is almost equal in the two cases where publisher and subscribers have the same priority or the publisher has the highest priority. If the subscribers have the highest priority then the transfer time increases by approximately 2 μs per subscriber. This is due to additional task switches by the DSP/BIOS scheduler when subscribers block on mailboxes.

The overhead for communication of algorithms residing on different DSPs stems from the indirection in the involved MAOs and the proxy PO as well as the proxy SOs. It can be seen from Table 2 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.

Table 2. Message transfer overhead time for publisher and subscribers residing on different DSPs. Overhead is given compared to direct PCI transfers without the PS-MW.

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

5.2. Genetic Optimization Algorithm

In order to evaluate the quality of PoSeGA's output, an exhaustive search algorithm that takes the whole search space into account has also been implemented. It therefore considers all possible camera configurations during the optimization and delivers all Pareto-optimal configurations. The exhaustive search algorithm has been executed on a standard x86-based platform but even there resulted in long execution times anyway.

The performance of the multi-criterion optimization online algorithm has been evaluated for several IVS-typical parameter setups as depicted in Table 3.

Table 3. Performance evaluation of PoSeGA for $r = 2$ DSP-units, $s = 6$ services in up to $q_s = 5$ QoS-levels.

Optima Found	Pareto-elements	Time (ms)
10%	19	51.02
25%	23	105.67
50%	29	260.83
70%	32	394.64
80%	33	514.86
90%	34	617.94
95%	35	866.08
100%	37	1849.00

It lists the obtained experimental results for a setup as given with the *SmartCam* with $r = 2$ DSPs and $s = 6$

IVS-services in up to $q_s = 5$ different QoS-levels per service. As also computed by the exhausting search algorithm, a total of 37 Pareto-optimal camera configurations exist for this setup.

The PoSeGA algorithm already finds all Pareto-optimal individuals after only testing 3,5% of the given theoretical search space of 153600 possible camera configurations. In this case, about 1.8 seconds have been measured for execution on the *SmartCam*. However, it can be seen that finding 90% of all existing optimal values only about 30% of the total execution time. This is an interesting observation which can particulary get used for designing different stop criteria for PoSeGA.

The obtained results demonstrate the feasibility of using an online optimizer by matching given soft-realtime constraints in IVS.

6. Conclusion

Smart cameras are an increasing trend in embedded video surveillance. They 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.

As smart cameras are embedded appliances that may also be deployed in solar- and battery-powered settings it is important to optimize power consumption. Furthermore, the Quality-of-Service of surveillance services has to be kept as high as possible to ensure proper detection rates. Therefore, a specialized power-aware reconfiguration framework for a cluster of distributed smart cameras is devised based on the *SmartCam* prototype platform developed in previous work [5].

Services, i.e., video analysis algorithms, are loaded / unloaded to cameras and service parameters are adjusted dynamically to minimize power consumption while retaining a maximum level of service quality. These optimal camera configurations are computed by a multi-criterion optimizer implemented as a genetic algorithm. A configuration manager decides on the best configuration based on a special cost function. The required flexibility in combining and exchanging algorithms at runtime is provided by a light-weight publisher-subscriber middleware framework.

In further work it is intended to advance the implementation of the cost model of PoSeGA so that continuously measured power and performance metrics are used instead of a priori estimates.

Furthermore, we are currently extending our framework for providing fault tolerance mechanisms. The dynamic reconfiguration capabilities are used to gracefully degrade surveillance services in case of failures.

7. References

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

[2] *The Corba Reference Guide: Understanding the Common Oject Request Broker Architecture*. Addison Wesley, Jan. 1998.

[3] *Java.rmi: The Remote Method Invocation Guide*. Addison Wesley, June 2001.

[4] A. Bogliolo, L. Benini, and G. D. Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3), June 2000.

[5] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach. Distributed Embedded Smart Cameras for Surveillance Applications. *IEEE Computer*, 39(2):68–75, Feb. 2006.

[6] C. C. Coello. A Short Tutorial on Evolutionary Multiobjective Optimization. In *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization, Zuerich, Switzerland*, pages 21 – 40, 2001.

[7] A. Doblander, B. Rinner, N. Trenkwalder, and A. Zoufal. A middleware framework for dynamic reconfiguration and component composition in embedded smart cameras. *WSEAS Transactions on Computers*, 5(3):574–581, Mar. 2006.

[8] C. Fetzer and K. 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 (WORDS'03)*, pages 66–73. IEEE, Jan. 2003.

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

[10] A. Maier, B. Rinner, H. Schwabach, and W. Schriebl. Online Multi-Criterion Optimization for Power-Aware Camera Configuration in Distributed Embedded Surveillance Clusters. In *Proceedings of the 20th IEEE International Conference on Advanced Information Networking and Applications (to appear), Vienna, Austria*, 2006.

[11] C. Mascolo, L. Capra, and W. Emmerich. Mobile computing middleware. In E. Gregori, G. Anastasi, and S. Basagni, editors, *Advanced Lectures on Networking: NETWORKING 2002 Tutorials*, volume 2497 of *Lecture Notes in Computer Science*, pages 20–52. Springer, 2002.

[12] K. Miettinen, editor. *Evolutionary Algorithms in Engineering and Computer Science*. John Wiley and Sons, 1999.

[13] C. Nitsch and U. 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 (DATE'02)*, page 1093. IEEE Computer Society, Mar. 2002.

[14] R. Rajkumar, M. Gagliardi, and L. 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, May 1995.

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