A Middleware Framework for Dynamic Reconfiguration and Component Composition in Embedded Smart Cameras

ANDREAS DOBLANDER, BERNHARD RINNER, NORBERT TRENKWALDER
Graz University of Technology
Institute for Technical Informatics
Inffeldgasse 16/1, 8010 Graz
AUSTRIA
{doblander, rinner, trenkwalder}@iti.tugraz.at

ANDREAS ZOUFAL
Austrian Research Centers Seibersdorf
Video and Safety Technology
Forschungszentrum, 2444 Seibersdorf
AUSTRIA
andreas.zoufal@arcs.ac.at

Abstract: Distributed embedded multi-processor smart cameras are central components in future intelligent video surveillance systems. Due to the complexity of video surveillance applications and the limited resources of the embedded smart cameras the set of employed analysis tasks has to be reconfigurable at runtime. In order to support dynamic reconfiguration in the resource-limited cameras a light-weight middleware for flexible algorithm communication and dynamic component composition is presented in this work. Based on a publisher-subscriber model the middleware aims at imposing only minimum communication overhead while providing adequate abstractions from algorithm interconnections. To further ease dynamic reconfiguration algorithms are provided as binary components complying with a special component model. A surveillance application can, therefore, be built by plumbing together several algorithm components. Permanent monitoring of all algorithm's resource and performance metrics allows the framework to detect possible resource overloading. By gracefully degrading the Quality-of-Service overloading can be prevented and system dependability is increased.

Key–Words: distributed multi-DSP, video surveillance, publisher-subscriber, dynamic reconfiguration, component composition

1 Introduction

Networks of distributed smart cameras are an emerging technology for a broad range of important applications, including smart rooms, surveillance, tracking and motion analysis. Smart cameras [1] are equipped with high-performance on-board computing and communication devices. They combine video sensing, processing and communication within a single embedded device.

We have designed a smart camera—we call it the *SmartCam*—as a fully embedded system. The *SmartCam* 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) [2].

Several requirements have to be met by the system software to employ this flexible high-performance platform in real distributed (surveillance) applications: (i) Flexibility in algorithm configurations, i.e., how tasks are composed to build the application, (ii) scalability concerning the number and the different types of employed surveillance tasks, (iii) low resource consumption so that resources are spared for surveillance tasks and image buffers, (iv) low performance overhead to allow real-time operation of surveillance tasks, and (v) real-time operation to meet requirements of surveillance tasks.

To meet the above requirements we have implemented a multi-layer heterogeneous software framework for our smart cameras. Since our smart cameras comprise a network processor and several DSPs the framework is divided into two parts. First, the *SmartCam-Framework* (SC-FW) running on the network processor. Second, the *DSP-Framework* (DSP-FW) is based on a publisher-subscriber middleware approach and is running on the DSPs.

This middleware allows to dynamically change the camera's functionality, i.e., various tasks can be loaded and unloaded at runtime or their Quality-of-Service (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. Actual analysis algorithms are binary components complying to a dedicated DSP algorithm component model that can be dynamically composed to build and adapt a surveillance applications, respectively. The framework constantly monitors all component's resource and performance data to identify resource bottlenecks. In case of expected resource overloading a graceful degradation of QoS is undertaken to retain system integrity. Therefore, overall system dependability is increased.

The remainder of this paper is organized as follows. In the next section important related work is presented. A brief overview of the system architecture of our smart camera is provided in Section 3. Then Section 4 discusses our software framework based on the publish-subscribe inter-process communication model for local inter-DSP services. Dynamic component composition and the DSP algorithm component model together with the component monitoring mechanisms are presented in Section 5. An experimental evaluation of the described approach is presented in Section 6. Finally, Section 7 concludes the paper.

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. Component-based middleware technologies from general purpose computing, such as, Microsoft DCOM [3], Java RMI [4] and OMG CORBA [5] are not suitable for very resource limited devices [6]. To adapt the CORBA technology to resource constrained real-time systems the Real-Time CORBA (RT-CORBA) and Minimum CORBA specifications [7, 8] have been introduced. Schmidt et al. [9] 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 [10] 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.

There are also other interesting component-based architectures for embedded platforms like, e.g., *SaveCCM* [11] and *PECOS* [12]. However, they are tailored for special operating systems or require an object-request broker (ORB). In our DSP-based platform we need only little of the functionality that an ORB provides and our focus is on a light-weight communication system with minimum overhead.

In [13] 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. The "BASE" middleware was implemented in Java which is not appropriate for our DSPs.

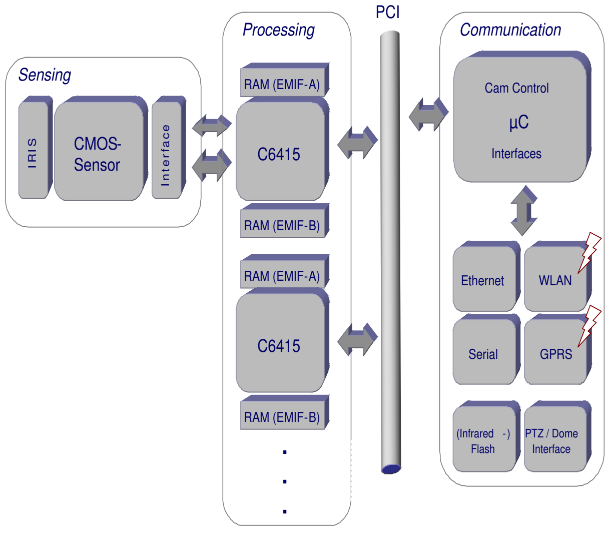

Figure 1: The scalable hardware architecture of the smart camera.

A popular inter process communication model for embedded systems is the real-time 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.

3 SmartCam Platform Overview

Our smart camera has been designed as a low-power, high-performance embedded system.

It comprises of a CMOS image sensor that delivers images with VGA resolution, a processing unit that can be equipped with up to ten TMS320C64x DSPs from Texas Instruments, and an Intel IXP425 network processor. The computing performance of this scalable architecture can be adapted to the requirements of the real-time video analysis and compression tasks intended for the application.

The DSPs are coupled via a local PCI bus which also serves as the connection to the network processor. The network processor also provides IP-based external communication via Ethernet and GSM/GPRS. A block diagram of our smart camera is shown in Fig. 1.

To ease application development for this platform of heterogeneous processors an abstract programming model is used. The DSPs are viewed as computing power providers and the network processor hosts the actual application logic where each algorithm is represented as an object. These algorithm objects carry a DSP binary that can be downloaded (on demand) to a DSP and performs the actual video processing.

4 Real-Time Publisher-Subscriber Architecture for DSP Algorithms

Applications for the *SmartCam* 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 buffered messaging via mailboxes is employed.

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.

4.1 Algorithms on a single DSP

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

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. A PSM is 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 depending on whether a publication or subscription is needed. This object then registers itself with the PSM. The newly registered service is added to the directory service where it can be looked up based on its unique identification number or its properties. 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 PSM. Therefore, the network processor also hosts a PSM that relays service requests between PSMs on different DSPs.

Properties (PrO) are used to describe published data and subscriptions as well. Each publisher and subscriber owns a PrO that identifies the details of provided and subscribed data services, respectively. Therefore, a PrO represents the QoS configuration of a data service. Examples for typical properties include image resolution and frame rate. 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 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.

Every task that provides data services instantiates a *publisher* (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.

A task that requires a data service of another algorithm instantiates a *subscriber* (SO). The SO in turn registers with the PSM. In order to receive data a mailbox is created. 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. 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.

4.2 Algorithms residing on different DSPs

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 *SmartCam*. Fig. 3 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 Fig. 3 that the situation on the involved DSPs is the same as it is in the single DSP case (cf. Fig. 2). 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.

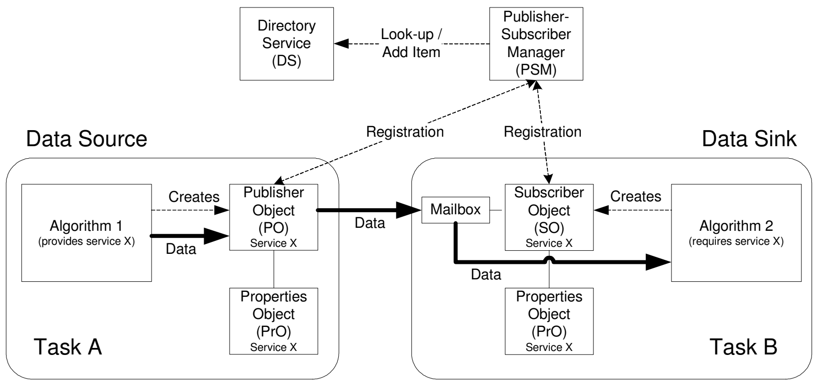
Directory Service (DS)
Look-up / Add Item
Publisher-Subscriber Manager (PSM)
Registration
Registration
Data Source
Data Sink
Algorithm 1 (provides service X)
Creates
Data
Publisher Object (PO) Service X
Data
Mailbox
Subscriber Object (SO) Service X
Creates
Algorithm 2 (requires service X)
Data
Properties Object (PrO) Service X
Properties Object (PrO) Service X
Task A
Task B

Figure 2: Principle relations between objects of the publisher-subscriber architecture.

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. Currently, that is the local PCI bus.

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 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.

5 Dynamic Reconfiguration and Algorithm Composition

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. Sec. 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.

5.1 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 Fig. 4. The DACM 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

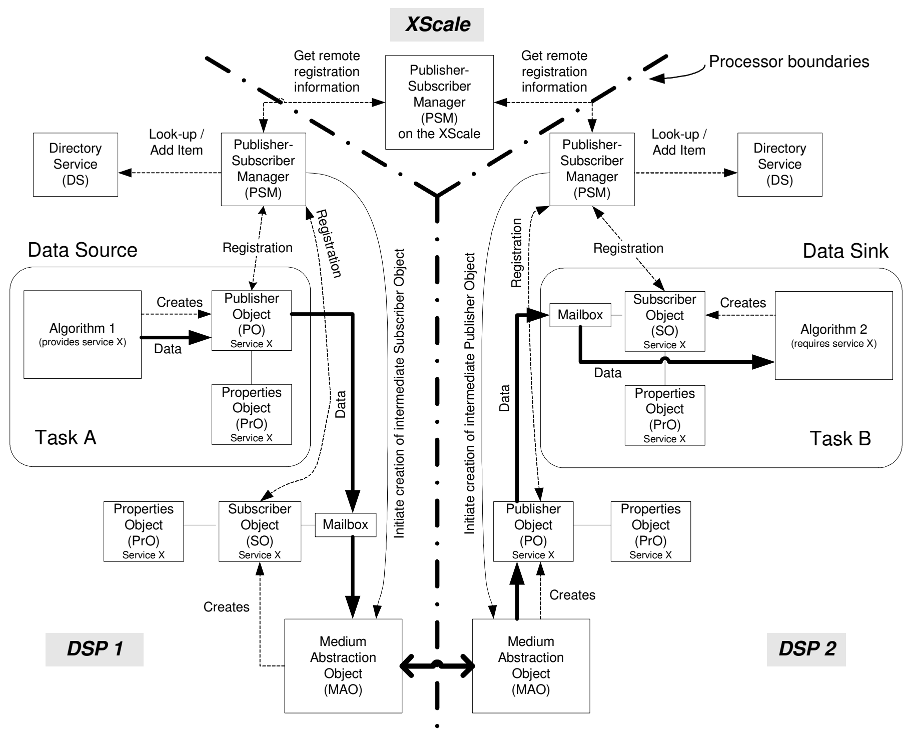
XScale
Publisher-Subscriber Manager (PSM) on the XScale
Get remote registration information
Get remote registration information
Processor boundaries
Directory Service (DS)
Look-up / Add Item
Publisher-Subscriber Manager (PSM)
Publisher-Subscriber Manager (PSM)
Look-up / Add Item
Directory Service (DS)
Registration
Registration
Registration
Registration
Data Source
Data Sink
Algorithm 1 (provides service X)
Creates
Data
Publisher Object (PO) Service X
Properties Object (PrO) Service X
Task A
Mailbox
Subscriber Object (SO) Service X
Creates
Data
Algorithm 2 (requires service X)
Properties Object (PrO) Service X
Task B
Data
Initiate creation of intermediate Subscriber Object
Initiate creation of intermediate Publisher Object
Data
Properties Object (PrO) Service X
Subscriber Object (SO) Service X
Mailbox
Publisher Object (PO) Service X
Properties Object (PrO) Service X
Creates
Creates
DSP 1
Medium Abstraction Object (MAO)
Medium Abstraction Object (MAO)
DSP 2

Figure 3: Extended publisher-subscriber architecture to connect algorithms running on different DSPs.

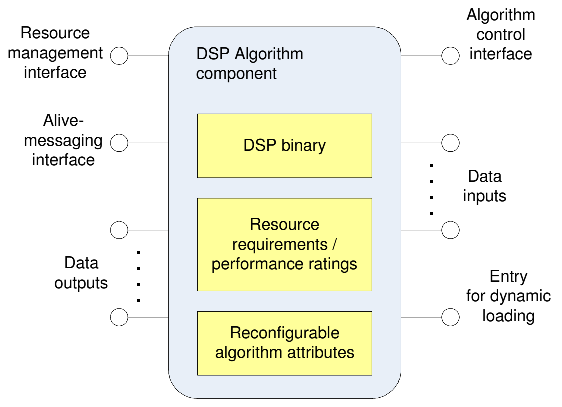
Resource management interface
Alive-messaging interface
Data outputs
DSP Algorithm component
DSP binary
Resource requirements / performance ratings
Reconfigurable algorithm attributes
Algorithm control interface
Data inputs
Entry for dynamic loading

Figure 4: Principle structure of a DACM component.

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 1: Algorithm information as provided by the DACM.

component are collected in Table 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.

5.2 Dynamic Component Composition and 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 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 the least important algorithms are removed from the system until the remaining algorithms are runnable. 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.

Execution times are constantly measured by hooks in the PS-MW at the inputs and the outputs of all algorithms. That is, a system counter is captured each time a hook function is called. 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.

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.

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 utilization is estimated from the algorithm characteristics provided by the DACM. It can be noted as $U_{EDMA} = \sum U_{EDMA,l}$, where $l = 1, \ldots, L$ are the L hardware priority queues of the EDMA controller and

$$U_{EDMA,l} = \sum_{c=1}^{K} \mathit{length}(c,l)\,\mathit{freq}(c,l) \qquad (1)$$

denotes the transfer bandwidth of priority queue l taking into account all of the K channels c. The function $\mathit{length}(c,l)$ 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, $\mathit{freq}(c,l)$ yields the number of transfers issued per second on channel c iff c is assigned priority l.

Middleware Component	Value (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 2: Memory requirements of middleware objects.

6 Experimental Evaluation

The *SmartCam* prototype has been used as the evaluation platform. It is based on 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.x 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.

An important requirement for the task communication framework on the DSPs of the *SmartCam* 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 2 that the runtime memory consumption is also low.

As the PS-MW adds some management overhead to the system we measured the times spent in the ini-

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 3: Initialization times of PS-MW components.

Transfer Mode	Value (μs)
Mailbox only	1.04
With PS-MW	1.21

Table 4: Message transfer times for plain mailbox communication and for a transfer using our publisher-subscriber middleware.

tialization 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 3. Initialization of the PSM and the DS is performed once at system startup. Creation and registration is performed whenever an according object is instantiated.

Message transfer overhead of the PS-MW compared to direct mailbox communication was measured to be 16.35%. In this experiment the time spent from sending the message at the publisher until it was received at the subscriber was measured and compared to simple mailbox transfers (cf. Table 4). Note that in this scenario one publisher with exactly one connected subscriber on the same DSP was examined.

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 from Fig. 5 that transfer time increases almost linearly with the number of subscribers.

Note also that due to the scheduler of the DSP/BIOS real-time operating system message transfer times depend on the task priorities of publisher and subscriber tasks. Fig. 5 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.

There are slight fluctuations depending on the number of overall tasks running on the DSP and their

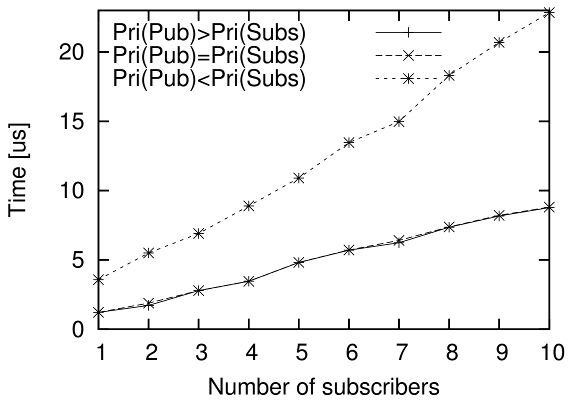

Figure 5: Transfer time increases depending on the number of subscribers and the priorities of PO and SO tasks (denoted “Pri(Pub)” and “Pri(Subs)”).

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 5: Message transfer overhead time compared to direct PCI transfers.

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.

In another experiment the transfer times between tasks on different DSPs have been analyzed (cf. Table 5). 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.

7 Conclusion

There is a strong trend towards intelligent infrastructures to ease everyday live. In traffic surveillance, e.g., networks of embedded smart cameras are introduced that provide on-site video analysis. In previous work [15, 2] we developed the *SmartCam* that is a heterogeneous multi-processor prototype of an embedded smart camera. It comprises a network processor and several DSPs.

In this work a real-time publisher-subscriber middleware (PS-MW) for the *SmartCam* platform is presented. It is a very light-weight architecture that supports loose coupling of tasks in the given dynamic application environment. By introducing 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.

Furthermore, a DSP algorithm component model is presented that allows for dynamic component composition. Algorithms are provided as binary components that have to provide the framework with well defined interfaces for reconfiguration and resource allocation. They also have to provide the framework with information on their typical performance metrics and their resource requirements. The software framework constantly monitors all algorithms to detect possible resource overloading which would compromise system integrity. In case of resource overloading graceful degradation of Quality-of-Service is undertaken to prevent system failure. Therefore, overall system dependability is increased.

An experimental evaluation on the *SmartCam* 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 in the order of several microseconds.

References:

[1] Wolf W., Ozer B., and Lv T. Smart Cameras as Embedded Systems. *IEEE Computer*, vol. 35(9), Sep. 2002, pp. 48–53.

[2] Bramberger M., Doblander A., Maier A., Rinner B., and Schwabach H. Distributed Embedded Smart Cameras for Surveillance Applications. *IEEE Computer*, vol. 39(2), Feb. 2006, pp. 40–47.

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

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

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

[6] Mascolo C., Capra L., and Emmerich W. Mobile Computing Middleware. In Gregori E., Anastasi G., and Basagni S. (eds.), *Advanced Lectures on Networking: NETWORKING 2002 Tutorials*, vol. 2497 of *Lecture Notes in Computer Science*. Springer, 2002.

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

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

[9] Schmidt D.C. Middleware for Real-Time and Embedded Systems. *Communications of the ACM*, vol. 45(6), Jun. 2002, pp. 43–48.

[10] Balasubramanian K., Wang N., Gill C., and Schmidt D.C. Towards Composable Distributed Real-Time and Embedded Software. In *Proceedings of the 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*. Guadalajara, Mexico, Jan. 2003.

[11] Hansson H., Åkerholm M., Crnkovic I., and Törngren M. SaveCCM—a component model for safety-critical real-time systems. In *Proceedings of the 30th EUROMICRO Conference*, 2004.

[12] Winter M., Genßler T., Christoph A., Nierstrasz O., Ducasse S., Wuyts R., Arévalo G., Müller P., Stich C., and Schönhage B. Components for Embedded Software—The PECOS Approach. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. ACM, 2002.

[13] Becker C., Schiele G., Gubbles H., and Rothermel K. BASE—A Micro-broker-based Middleware For Pervaisve Computing. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*. IEEE, Mar. 2003.

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

[15] Bramberger M., Brunner J., Rinner B., and Schwabach H. Real-Time Video Analysis on an Embedded Smart Camera for Traffic Surveillance. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004.