A light-weight Publisher-Subscriber Middleware for Dynamic Reconfiguration in Networks of 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
`http://www.iti.tugraz.at/smartcam`

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

Abstract: Traffic video surveillance applications are increasingly implemented by networks of embedded smart cameras. These smart cameras provide on-board real-time video analysis and streaming. This work presents a light-weight middleware for a heterogeneous multi-processor smart camera platform comprising a network processor and several DSPs. It supports dynamic reconfiguration by changing and rearranging algorithms at runtime. The middleware employs a publisher-subscriber-based architecture for task communication. It's the aim of this work to provide efficient communication by imposing minimum overhead on the DSPs. Experimental analysis shows the efficiency of the approach.

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

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 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.

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. 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.

In [11] 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.

A popular inter process communication model for embedded systems is the real-time publisher/subscriber model (RT-PS) [12]. 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.

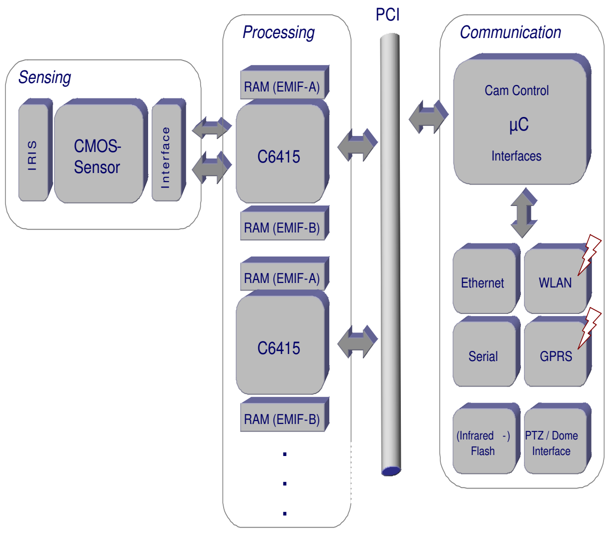

Figure 1: The scalable hardware architecture of the smart camera.

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 *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

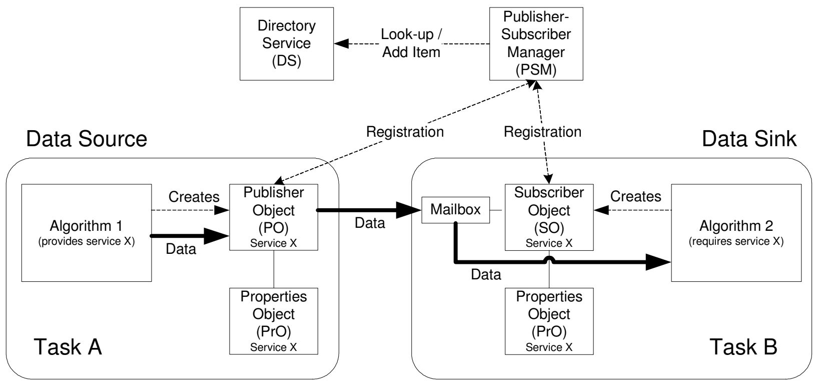
Directory Service (DS)
Look-up / Add Item
Publisher-Subscriber Manager (PSM)
Registration
Registration
Data Source
Data Sink
Algorithm 1 (provides service X)
Creates
Data
Publisher Object (PO) Service X
Properties Object (PrO) Service X
Task A
Data
Mailbox
Subscriber Object (SO) Service X
Creates
Algorithm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

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

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 Quality-of-Service (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. 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.

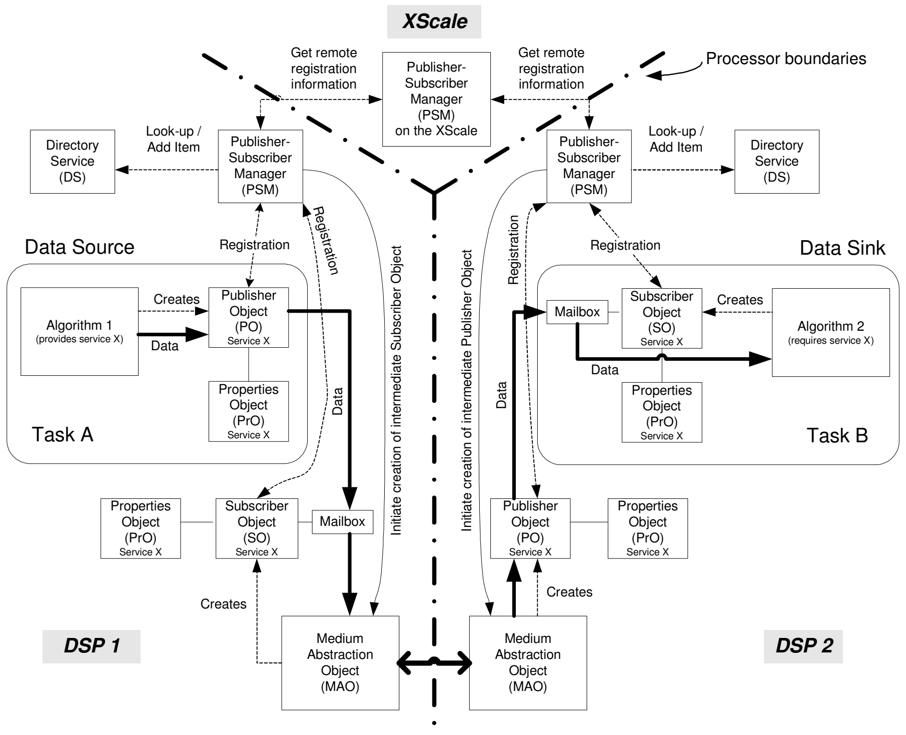
XScale
Get remote registration information
Publisher-Subscriber Manager (PSM) on the XScale
Get remote registration information
Processor boundaries
Directory Service (DS)
Look-up / Add Item
Publisher-Subscriber Manager (PSM)
Publisher-Subscriber Manager (PSM)
Look-up / Add Item
Directory Service (DS)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

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 Performance Analysis

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 1 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 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

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 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 2: Initialization times of PS-MW components.

are collected in Table 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.

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. 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. 4 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. 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.

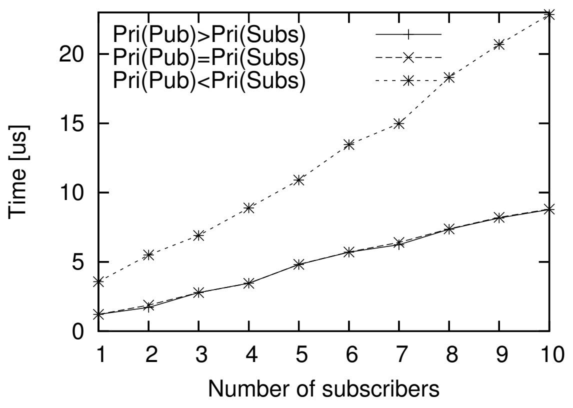

Figure 4: Transfer time increases depending on the number of subscribers and the priorities of PO and SO tasks (denoted “Pri(Pub)” and “Pri(Subs)”).

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

In another experiment the transfer times between tasks on different DSPs have been analyzed (cf. Table 3). 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.

6 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 [13, 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.

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. *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*, 2006. (to appear).

[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] 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.

[12] 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.

[13] 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.