EMBEDDED MIDDLEWARE ON DISTRIBUTED SMART CAMERAS

Bernhard Rinner, Milan Jovanovic and Markus Quaritsch

Graz University of Technology
Institute for Technical Informatics
Inffeldgasse 16, 8010 Graz, AUSTRIA

ABSTRACT

Two trends emerge in recent image processing research: distributed computing and embedded processing. Both trends are exemplified in smart cameras which combine image sensing, image processing and communication on a single embedded device. Networks of distributed smart cameras help to overcome some hard problems that are inherent in single-camera systems.

Designing, implementing and deploying image processing applications for cooperating distributed cameras is much more complex than for single-camera systems. In this paper, we focus on software services required for distributed embedded image processing on a network of smart cameras. We identify important middleware services and present our lightweight middleware implemented on our distributed SmartCams. A multi-camera tracking application demonstrates the benefits of our approach.

Index Terms— image processing, embedded systems, smart cameras, distributed systems, middleware

1. INTRODUCTION

Image processing has been a very active research area over the last decades, and results of this research can now be found almost everywhere. The market for image processing technology is rapidly increasing; its applications range from machine inspection over security to consumer products.

Two trends emerge in recent image processing research: *distributed computing* and *embedded processing*. These trends are very well exemplified in *smart cameras* [1, 2] which combine image sensing, image processing and communication on a single embedded device. Smart cameras have a substantial processing and communication infrastructure onboard and can perform advanced image processing algorithms such as motion detection, segmentation, tracking and object recognition in real-time. They typically deliver color and geometric features, segmented objects or rather high-level decisions such as wrong way drivers or suspect objects. The abstracted results may be transferred either within the video stream, e.g., by color-coding, or as a separate data stream. Note that the onboard computing infrastructure of smart cameras is often exploited to perform high-level video compression.

Developing distributed image processing applications is challenging, and a substantial system-level software would strongly support the implementation. However, embedded platforms with limited resources, typically, do not provide middleware services well-known on general-purpose platforms. In this paper, we focus on such system-level software for distributed image processing, i.e., we identify important domain-independent services for deploying, networking and operating a smart camera network. We then discuss application-specific services for spatial and temporal calibration.

The remainder of this paper is organized as follows: Section 2 identifies the potential of distributed smart camera systems. Section 3 discusses middleware services for distributed smart cameras. Section 4 presents our smart camera architecture (SmartCam) and the middleware services we have implemented. Section 5 demonstrates the benefits of our middleware by a multi-camera tracking application. Section 6 concludes this paper with a brief discussion.

This research has been partially supported by the Austrian Research Promotion Agency under grant 810072.

2. DISTRIBUTED SMART CAMERAS

Distributed image processing definitely introduces several complications. However, we strongly believe that the problems which this approach solves are much more important than the challenges of designing and implementing a network of *distributed smart cameras (DSCs)*. DSCs help to overcome some very hard problems that are available in single-camera systems.

Occlusion is a major problem for single cameras. When multiple views of a subject are available, it is much more likely to see parts of an object that is occluded in the field of view of one camera by switching to a different camera.

In a multi-camera setting, it is more likely that one camera is closer to an object than in a single camera setting. Thus, objects of interest can be captured with higher resolution.

Local processing close to the image sensor abstracts the resulting data, and onboard compression significantly reduces the amount of the output. Thus, distributed processing helps to reduce the required bandwidth for communication among the camera network.

Real-time considerations also argue in favor of distributed computing. Local processing avoids round-trip delays to a server and sharing common resources which in turn increase the predictability of the processing time.

By having access to cameras with different views of the scenario, we may overcome failures of individual cameras. Distributed computing enables fault-tolerance and helps to increase the reliability of the multi-camera system.

3. MIDDLEWARE SERVICES FOR DISTRIBUTED SMART CAMERAS

3.1. Related Middleware Approaches

Designing, implementing and deploying image processing applications for distributed smart cameras is much more complex than for single-camera systems. On general-purpose platforms, distributed applications are often based on a middleware system which provides services for networking and data transfer [3].

On DSC networks we would like to take advantage of middleware services as well. However, the requirements of a middleware for distributed image processing on embedded devices are significantly different. Component-based middleware such as DCOM or CORBA are targeted for general-purpose computing and are not suitable for resource limited devices. The CORBA technology has been adapted to resource constrained real-time systems, e.g., by the Real-Time CORBA (RT-CORBA) specification and its "TAO" implementation [3]. However, this approach is still very resource consuming.

On the other hand, recent research in *wireless sensor networks (WSN)* has come up with some interesting middleware concepts as well [4]. Due to the nature of WSNs these middleware systems especially focus on reliable services for ad-hoc networks and energy-awareness [5].

DSC networks are different to WSNs in various aspects. First, the amount of data to be processed is much higher in DSC networks than in WSNs. Second, individual processing nodes in a DSC network are more capable than in WSNs. While resource constraints on the embedded smart cameras are important, the resource limitations, especially energy, are of top priority in WSN. Third, due to ad-hoc networking, communication in WSN has a very dynamic nature. DSCs, on the other hand, are typically connected via wired networks providing higher communication bandwidths.

3.2. Middleware services

In the following, we briefly identify important services required for an efficient deployment and execution of image processing applications on DSC networks. These services are described from an application-oriented perspective.

Deployment services. Deployment services help to initiate the DSC network and to start up the image processing application.

The *dynamic loading service* enables to change the functionality of an individual camera after compile-time. The new functionality is provided by a dynamically loadable executable. Using this service functions can also be removed from individual cameras. Functional units are often referred to as tasks.

The *allocation service* helps to find the optimal allocation of individual tasks among the cameras. In general, task allocation is a complex problem, and finding the optimal solution requires some form of modeling of the overall application and the available resources of the DSC network.

The *monitoring service* delivers the actual state of an individual camera including the available resources and the allocated tasks. The camera's state information is important for optimizing the resource allocation and for providing fault-tolerance mechanisms.

Operational services. Operational services are responsible for the efficient coordination and configuration of the DSC network during runtime.

Synchronization provides a well-defined temporal relation among the cameras. This temporal relation is important for collaborative image processing. However, the required temporal accuracy depends on the image processing algorithm and may range from frame synchronization among the cameras to a very relaxed synchronization.

Code mobility enables the migration of code and data from one camera to another during runtime. Mobile code is a powerful programming paradigm which supports the design and execution of distributed applications.

Reconfiguration is related to allocation in the sense that it supports changing the allocation during runtime. Thus, the allocation of tasks can be be dynamically modified. The DSC network can then react to changes in its environment and the camera's internal state.

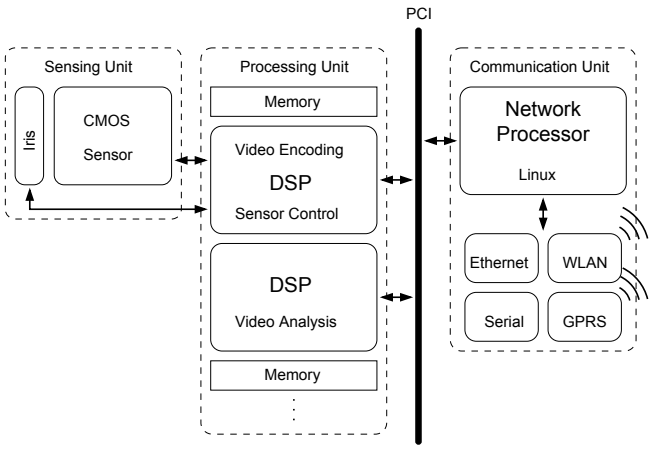

Fig. 1. The hardware architecture of our smart camera.

Networking services. Networking services establish transparent services for communication, data transfer and resource management in the DSC network.

Data and naming services provide the basic mechanism for efficient communication among the cameras. For distributed image processing, two important services can be separated: Streaming services deliver image data at a certain QoS over the network. Messaging services are required to coordinate the cameras.

Resource management helps to keep track of all resources in the DSC network.

Application-specific services. These services are specific to image processing and may vary strongly among different applications.

Calibration is required in various degrees in distributed image applications. Determining the spatial relation among the individual cameras is important. The required spatial accuracy also depends on the application and may range from a pixel-based registration of the individual field of views to a graph-based neighborhood relation.

QoS services provide a network-wide QoS of the image processing application.

4. MIDDLEWARE IMPLEMENTATION

4.1. SmartCam Architecture

Figure 1 depicts the hardware architecture of our smart camera *(SmartCam)* [2] which is comprised of a sensing unit, a processing unit and a communication unit. A CMOS image sensor is the core of the sensing unit. It delivers color images up to VGA resolution at 25 frames per second to the processing unit via a FIFO memory. The processing unit is composed of a variable number of digital signal processors (DSPs) which are connected via a local PCI bus. The image processing algorithms are executed on these DSPs. An ARM-based network processor controls the communication unit which has two main tasks. First, it coordinates the internal communication among the DSPs as well as the DSPs and the network processor. Second, it provides IP-based communication channels to the outside world.

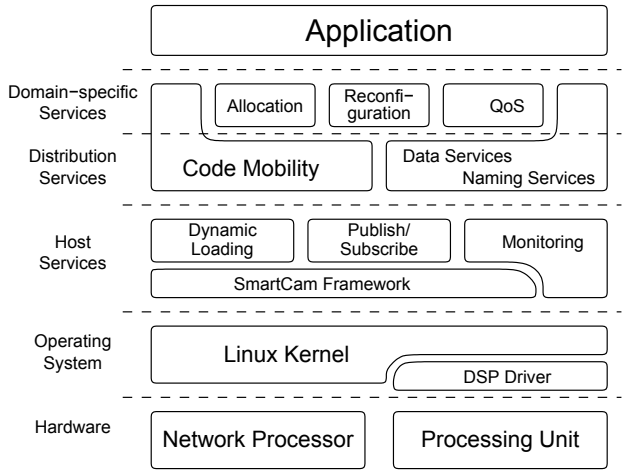

Fig. 2. The architecture of the SmartCam middleware.

4.2. SmartCam Middleware

On our *SmartCam* we employ a lightweight middleware which aims to ease the development of distributed image processing applications. The different services provided by the middleware can be organized in a layered architecture. The functionality of each layer is adapted from [3]. Figure 2 sketches the architecture of our SmartCam middleware.

Operating system. The operating system along with its hardware drivers and communication channels builds the basic layer of our architecture. On the network processor, a standard Linux kernel version 2.6.17 is used. A custom kernel module (DSP driver) is responsible for managing the communication between the DSPs as well as the DSPs and the network processor via the PCI bus. The DSP framework [6] running on each DSP abstracts the DSPs, handles the communication with the network processor and provides an environment for the video processing algorithms.

Host services. The SmartCam framework [6] builds the foundation of the host services. It provides a basic message-oriented communication mechanism between applications on the network processor and image processing algorithms on the DSPs. The services for (1) dynamic loading, (2) publish/subscribe, and (3) monitoring operate on top of the SmartCam framework.

Dynamic loading allows to load image processing algorithms dynamically during runtime on the DSPs. On the network processor, a dynamically loadable executable is sent to the DSP whereas the DSP framework is responsible for loading and starting the executable on the DSP.

The publish/subscribe service provides a flexible communication mechanism. Data sources and data sinks are connected dynamically whereas the data source and data sink can be located on different processors on the smart camera.

The monitoring service observes the utilization of various resources on an individual smart camera. Our monitoring is dedicated to important resources on the embedded platform such as CPU usage, memory utilization as well as the utilization of the communication channels on each processor, i.e., DMA channels on the DSPs, PCI bus, utilization of network channels.

Distribution services. While the lower layers provide services for applications on a single camera, this layer integrates multiple smart cameras to a distributed image processing system. In our implementation we use a mobile agent system as foundation for distributed applications.

An agent system usually supports communication between agents independent of the current host an agent resides on. The agent system further facilitates the abstraction of image processing tasks by mobile agents. Low-level image processing is implemented as DSP executable while the agent contains the application logic and controls the image processing algorithm. Code mobility is also inherent to a mobile agent system. Mobile agents can migrate between the hosts as required. Exploiting dynamic loading of DSP executables allows to migrate the image processing algorithm as well.

Domain-specific services. On top of the distribution services operate the domain-specific services (1) allocation, (2) reconfiguration, and (3) quality of service (QoS). We have modeled the task allocation as a constraint satisfaction problem (CSP) [7]. A solution of the CSP corresponds then to an allocation of tasks to cameras. The allocation is performed before runtime as an deployment service.

The overall objective of dynamic reconfiguration is to transfer the network into a specific configuration which better captures the current requirements. Dynamic reconfiguration is performed at runtime and requires some form of reasoning about the specific configuration. Thus, we need some information about the state of the current configuration as well as some objective for the next configuration.

We have implemented dynamic reconfiguration in three steps. First, *context sensing and analysis* gathers information about the current state of the DSC network. The current state is retrieved using the monitoring service. Second, the *computation of new configurations* is determined by an optimizer which uses the contextual information of the DSC network and the reconfiguration objective as input. In our implementation, the reconfiguration objective is specified by policies. Third, the *reconfiguration enforcement* performs the actual transfer from the current to the next configuration. This enforcement is implemented by our mobile agent system and the dynamic loading services. All three steps are executed periodically in a so-called reconfiguration loop.

Our middleware supports combined management of power and QoS (PoQoS) [2]. PoQoS dynamically configures the power and QoS level of the cameras' hard- and software to adapt to user requests and changes in the environment.

5. CASE-STUDY: MULTI-CAMERA TRACKING

In this section, we present our implementation for tracking objects in a network of smart cameras which is based on the middleware infrastructure described in section 4. Tracking an object on a single camera uses the well-known CamShift [8] algorithm which in turn has been extended to track objects among multiple cameras. In contrast to [9], our approach focuses on an autonomous, decentralized solution for tracking an object among multiple cameras. Compared to [10], we consider object tracking as an additional task which is started on demand and our implementation targets on an embedded platform.

The basic idea of our multi-camera tracking approach [2] is to employ a single instance of a tracker for each object of interest. This tracking instance always resides on the camera which observes the object and follows the target from one camera to the next as it moves within the supervised area. This means, that only a single camera has

to perform the tracking task. The tracking instance consists of (1) the tracking algorithm which extracts the trajectory of the object from the video stream acquired by the camera, and (2) a mobile agent. The mobile agent encapsulates the tracking algorithm and contains the application logic. It is responsible for controlling the tracking algorithm and following the target among the camera network. The tracking agent itself acts autonomously depending on its strategy in order to follow the target.

The handover of an object from one camera to the next is fully decentralized and involves only adjacent cameras. When the tracked object is about to leave the field of view of the current camera, the tracking agent has to continue tracking on the camera which will observe the object next. Therefore, each camera has defined a set of so-called migration regions which are basically a polygon in the 2D image space. Each migration region is assigned to one or more neighboring cameras. An additional direction vector helps to distinguish among several smart cameras assigned to the same migration region [11].

In order to ease the development of distributed image processing tasks, the application uses, among others, the following services provided by the underlying middleware:

Abstraction of image processing. The application logic is implemented in the mobile agent. In the case of our decentralized object tracking, this means that the agent is responsible for following the target from one camera to the next. But in order to perform the correct actions, i.e., ascertain to reside on the camera observing the object, the agent depends on the information obtained from the tracking algorithm. The concrete tracking algorithm used is exchangeable and can be adapted to the current needs.

Code migration and dynamic loading. Code migration is fundamental for our tracking approach. The tracking algorithm is executed only on the camera which observes the target. When the target leaves the field of view of the current camera, the agent has to migrate to a neighboring camera. While agent migration is inherent to the mobile agent system, the tracking algorithm also has to be aware of this situation. Hence, the tracking algorithm has to be able to store its internal state and also reinitialize itself from a previously stored state. Migrating an image processing task from one camera to another also relies on dynamic loading and unloading of DSP algorithms.

Transparent messaging. From the class of networking services, the presented tracking application exploits the service of transparent communication. In order to visualize the position of the observed object, its position can be displayed on a PC. Therefore, the tracking agent sends messages containing the position of the target to a visualizer agent which in turn draws the current position on the screen or logs it to a file. Hence the communication is handled by the agent system, both agents can migrate to another host and still communicate with each other.

6. CONCLUSION

In this paper, we have presented the potential of distributed smart cameras for solving important problems of current single-camera systems. A middleware system would strongly support the application development on the DSC network. We have implemented such a middleware for our SmartCams which provides *some* services for deployment, operation and networking. Our middleware focuses on efficiency and resource-awareness. It has been demonstrated on a multi-camera tracking application.

From our experience, such a middleware eases the development of distributed image processing applications in the following way. First, it provides a clear separation between image processing algorithm implementation and coordination of these algorithms among different cameras. Second, it strongly supports scalability, i.e., to develop applications for a variable number of cameras. Third, the available resources can be better utilized by dynamic loading and dynamic reconfiguration services which are usually not available on embedded platforms.

7. REFERENCES

[1] Wayne Wolf, Burak Ozer, and Tiehan Lv, "Smart cameras as embedded systems," *Computer*, vol. 35, no. 9, pp. 48–53, Sept. 2002.

[2] Michael Bramberger, Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach, "Distributed embedded smart cameras for surveillance applications," *Computer*, vol. 39, no. 2, pp. 68–75, 2006.

[3] Douglas C. Schmidt, "Middleware for real-time and embedded systems," *Communications of the ACM*, vol. 45, no. 6, pp. 43–48, June 2002.

[4] Mohammad M. Molla and Sheikh Iqbal Ahamed, "A Survey of Middleware for Sensor Networks and Challanges," in *Proceedings of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, Columbus, Ohio, USA, Aug 2006, pp. 223–228, IEEE.

[5] Yang Yu, Bhaskar Krishnamachari, and Viktor K. Prasanna, "Issues in designing middleware for wireless sensor networks," *IEEE Network*, vol. 18, no. 1, pp. 15–21, Jan/Feb 2004.

[6] 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*, vol. 5, no. 3, pp. 574–581, March 2006.

[7] Michael Bramberger, Bernhard Rinner, and Helmut Schwabach, "A Method for Dynamic Allocation of Tasks in Clusters of Embedded Smart Cameras," in *Proceedings of the IEEE International Conferens on Systems, Man and Cybernetics*, October 2005, pp. 2595 – 2600.

[8] Gary R. Bradski, "Computer vision face tracking for use in a perceptual user interface," *Intel Technology Journal*, , no. Q2, pp. 15, 1998.

[9] Sven Fleck, Florian Busch, Peter Biber, and Wolfgang Straßer, "3D Surveillance – A Distributed Network of Smart Cameras for Real-Time Tracking and its Visualization in 3D," in *Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop*, Jun. 2006, pp. 118 – 126.

[10] Senem Velipasalar, Jason Schlessman, Cheng-Yao Chen, Wayne Wolf, and Jaswinder Pal Singh, "SCCS: A Scalable Clustered Camera System for Multiple Object Tracking Communicating via Message Passing Interface," in *Proceedings of IEEE International Conference on Multimedia and Expo 2006*, 2006.

[11] Markus Quaritsch, Markus Kreuzthaler, Bernhard Rinner, Horst Bischof, and Bernhard Strobl, "Autonomous multicamera tracking on embedded smart cameras," *EURASIP Journal on Embedded Systems*, vol. 2007, 2007.