DSCAGENTS: A LIGHTWEIGHT MIDDLEWARE FOR DISTRIBUTED SMART CAMERAS

Markus Quaritsch and Bernhard Rinner

Institute of Networked and Embedded Systems, Pervasive Computing Group
Klagenfurt University
9020 Klagenfurt, AUSTRIA

markus.quaritsch@uni-klu.ac.at, bernhard.rinner@uni-klu.ac.at

ABSTRACT

This paper investigates on middleware for distributed smart cameras. We describe DSCAgents, our agent-oriented lightweight middleware system. The design goal is to provide a modular and flexible middleware that takes into account the underlying hardware platform and also supports collaborative image processing.

Mobile agents are used to model image processing tasks and to manage the whole smart camera network. The agent-oriented approach simplifies application development and domain specific services unburden programmers from implementing the same functionality for each application again. Mobility of agents allows to build highly dynamic and adaptive systems where image processing tasks can move from one camera to another during operation. The evaluation of DSCAgents shows reasonable performance while keeping the resource requirements low.

Index Terms— Distributed Smart Cameras, Embedded Middleware, Mobile Agent System

1. INTRODUCTION

Smart cameras have been the subject of study in research and industry for quite some time. While in the "early days" sensing and processing capabilities were very limited, we have seen a dramatic progress in smart camera research and development in the last few years [1, 2, 3]. Recently, much effort has been put in the development of networks of smart cameras. These *distributed smart cameras (DSC)* [4, 5] are real-time distributed embedded systems that perform computer vision using multiple cameras. This new approach is emerging thanks to a confluence of demanding applications and the huge computational and communications abilities predicted by Moore's Law.

While sensing, processing and communication technology is progressing at high pace, we unfortunately do not experience such a rapid development on the system-level software side. Designing, implementing and deploying applications for distributed smart cameras is typically a complex and challenging endeavor. So we would like to get as much support as possible from system-level software on the DSC network. Such a system-level software or *middleware system* abstracts the network and provides services for the application. DSC networks, however, have significantly different characteristic compared to other well-known network types such as computer networks [6] or sensor networks [7]. Thus, we can not directly adopt middleware systems available for these networks.

From the application's point of view, the major services a middleware system should provide are for the distribution of data and control. However, DSC networks are mostly deployed to perform distributed signal processing applications. Thus, middleware systems for DSCs should also provide dedicated services for these applications. We propose to use the agent-oriented paradigm for implementing the middleware services and realizing custom applications.

In previous work [8, 9] we introduced the agent-oriented approach for distributed smart cameras, identified a number of basic services such a middleware should provide, and demonstrated the use of mobile agents for application development. In this paper we discuss our agent-oriented middleware and certain services in greater detail. In [10] we presented a middleware for embedded sensor nodes in the context of sensor fusion where we focus on services for task distribution and dynamic reconfiguration.

The reminder of this paper is organized as follows. Section 2 gives a short introduction to smart cameras and distributed smart cameras. Section 3 discusses related middleware approaches in the domain of general purpose computing and embedded systems, focusing on agent-oriented approaches. Section 4 describes our agent-oriented middleware design and its implementation as well as available services. Section 5 presents some evaluation results and Section 6 finally concludes the paper.

2. BACKGROUND

2.1. Embedded Smart Cameras

The generic architecture of smart cameras consists of a sensing unit, a processing unit, and a communication unit (cf. Fig. 1). The image sensor, which is implemented either in CMOS or CCD technology, represents the data source of the processing pipeline in a smart camera. The sensing unit reads the raw data from the image sensor and often performs some preprocessing such as white balance and color transformation. The main image processing tasks take place at the processing unit which receives the captured images from the sensing unit, performs real-time image analysis and transfers the abstracted data to the communication unit. The communication unit controls the whole processing pipeline and provides various external interfaces such as USB, Firewire or Ethernet, among others.

These generic units are implemented on various architectures ranging from system-on-chip (SoC) platforms over single processor platforms to heterogeneous multi-processor systems. The main design issues for building smart cameras are to provide sufficient processing power and fast memory for processing the images in real-time while keeping the power consumption low.

Smart cameras deliver some abstracted data of the observed scene. It is natural that the delivered abstraction depends on the camera's architecture and applications; almost every smart camera currently delivers a different output. Smart cameras perform a variety of image processing algorithms such as motion detection, segmentation, tracking, object recognition and so on. They typically deliver color and geometric features, segmented objects or rather high-level decisions such as wrong way drivers or suspect objects.

2.2. Distributed Smart Cameras

Single camera systems are very limited. Either only a very limited area is can be monitored or the number of pixels on target is too low to identify certain objects when covering larger areas. To overcome this limitation, multi-camera systems are deployed. However, most installations follow a centralized architecture where huge amounts of processing power is provided in the back office for image processing and scene analysis (e.g., [11, 12]). But processing the images of multiple sensors on a central host has several drawbacks. First of all, the communication costs are very high. Analog CCTV cameras but also digital IP cameras require dedicated high-bandwidth wiring from each camera to the back office. Centralized systems have the downside of limited scalability. The main limiting factors are the communication bandwidth that can be handled in the back office and the processing power required for analyzing the images of dozens of cameras in real-time.

Smart cameras are key components for future distributed vision systems and promise to overcome the limitations of centralized systems. Distributed computing offers greater flexibility and scalability than centralized systems. Instead of processing the accumulated data on a dedicated host, scene analysis is done in a distributed manner within the smart camera network. Individual cameras, therefore, have to collaborate on certain high-level tasks (e.g. scene understanding, behavior analysis). Low-level image processing is done on each camera. Collaboration among cameras is based on abstracted data, which influences the communication infrastructure. Smart camera networks communicate in a peer-to-peer manner and typically have significantly lower bandwidth requirements. Instead of depending on a central control instance which manages the whole system, distributed smart cameras are able to organize themselves, i.e. allocate tasks, form clusters of collaboration, etc. Fault tolerance is another aspect in favor of a distributed architecture. While the reliability of a centralized system depends on a single or a few components in the back office, distributed smart cameras, in contrast, can degrade gracefully.

3. RELATED WORK

Middleware systems are used in many application domains of distributed systems. In general purpose computing the probably most prominent middleware standard is OMG's Common Object Request Broker Architecture (CORBA) [13]. CORBA is a distributed object system which allows objects on different hosts to interoperate across the network. CORBA is designed to be platform independent and not constrained to a certain programming language. An object's interface is described in a more general language, the interface description language (IDL), which is then mapped to the native data types of a programming language. Real-Time CORBA (RT-CORBA) and Minimum CORBA [14, 15] have been specified for resource constrained real-time systems. Other approaches are Microsoft's Distributed Component Object Model (DCOM) [16, 17] and Sun's Java Remote Invocation (RMI) [18]. Unlike CORBA, these middleware implementations are limited to a certain platform (i.e., Microsoft Windows) or a certain programming language (i.e., Java).

Most agent systems for general purpose computing are implemented either in scripting languages (e.g., D'Agents [19], ARA [20]) or Java (e.g., Grasshopper [21], Voyager [22], Diet-Agents [23, 24]) and only very few are implemented in languages compiled to native code (e.g., Mobile-C [25])

Embedded systems are becoming more and more distributed in recent years. Hence, middleware systems are developed in this application domain as well, but the requirements are typically different due to tight resource constraints. For wireless sensor networks most middleware systems are based on TinyOS [26], a component oriented, event-driven operating system for motes. Middleware implementations range from

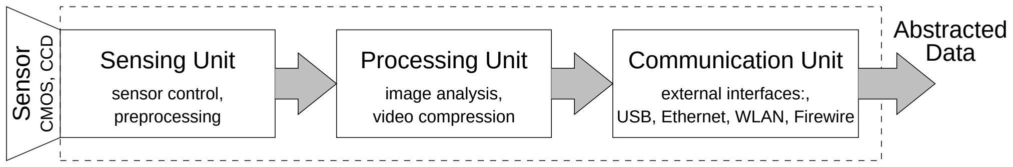


Fig. 1: Generic smart camera architecture.

a virtual machine on top of TinyOS, hiding platform and operating system details, to more data-centric middleware approaches for data aggregation and data query (cf. [27]).

In recent years, the agent-oriented paradigm has also been used to enhance software development for embedded systems in general in various application domains such as process control, real-time control and robotic, among others (cf. [28, 29, 30]) and wireless sensor networks in particular (e.g., Agilla [31], In-Motes [32]).

Typical smart camera platforms lie in-between general purpose computing and wireless sensor nodes when considering the available resources. In addition, smart cameras have to support real-time image processing as well as collaborative image processing. Hence, a middleware for smart camera networks has to find a trade-off between platform independence, programming language independence and the overhead introduced by the middleware.

4. DSCAGENTS: AN AGENT-ORIENTED MIDDLEWARE

In this section we describe DSCAgents, a lightweight agent-oriented middleware for embedded smart cameras. Agent-oriented programming (AOP) has become more and more prominent in software development during the last years. The agent-oriented programming paradigm extends the well known object oriented paradigm and introduces active entities, so-called *agents*. The agents are situated in an environment, usually called *agency*, which provides the required infrastructure and services.

We believe that mobile agents are perfectly suited to manage whole networks of smart cameras. The ultimate goal is that distributed smart cameras operate completely autonomous with no or only minimal human interaction. Mobile agents, therefore, can be used to model individual tasks within the system whereas the agents are able to organize themselves. Collaborative image processing, hence, maps to collaboration among agents.

Figure 2 sketches a smart camera network operated by mobile agents. The cameras are connected via an IP network (wired or wireless) whereas each camera hosts an agency, executed on the camera's communication unit. Mobile agents represent the image processing tasks that have to be executed. Note that the tasks are not bound to a certain camera but can be moved from one camera to another during operation.

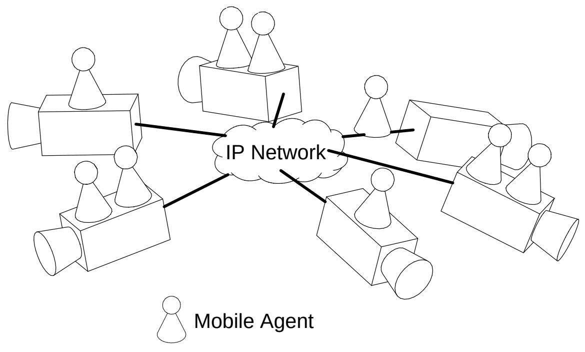


Fig. 2: Smart camera network operated by a mobile agent system.

Since DSCAgents targets embedded smart cameras, C++ was chosen as programming language for efficiency reasons. This decision influences the design of DSCAgents to some degree, especially regarding mobility of agents.

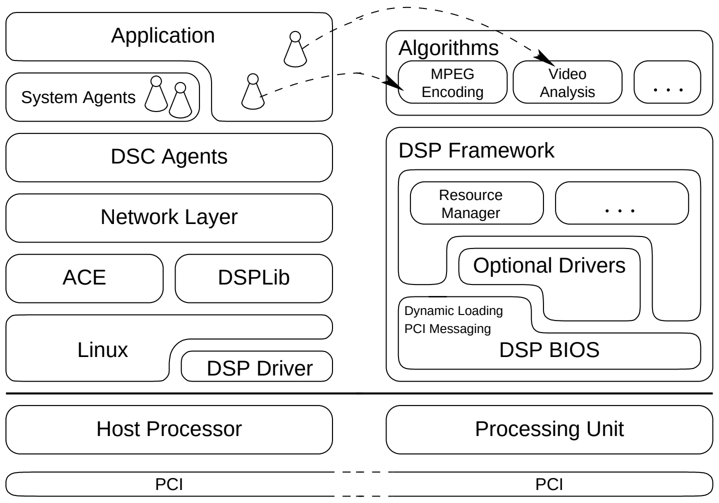
4.1. Software Architecture

The general architecture of smart cameras distinguishes between the processing unit and the communication unit (cf. Section 2.1). Consequently, this separation is reflected in the software architecture as depicted in Figure 3. The software stack on the processing unit (cf. Figure 3b) comprises a real-time operating system, drivers for additional hardware modules such as image sensors, and management modules in form of the DSPFramework [33]. The software stack on the communication unit consists of the Linux operating system, various libraries, a network layer for inter-camera communication, and DSCAgents on top of it.

Stationary system agents provide general services as well as domain specific services supporting application development such as remote agent creation, gathering monitoring information, obtaining camera parameters, and interacting with algorithms executed on the processing unit, among others.

4.2. Agent Layer

The agency layer is the core of DSCAgents on top of the network layer which provides basic inter-camera communication mechanisms. Application developers implement a number of



(a) Host processor (b) Processing unit

Fig. 3: Software Architecture.

new agents for certain tasks. Hence, the goal of this layer is to hide platform specific issues and concurrency issues so that application developers can concentrate on the main things, namely the application logic.

4.2.1. Agents

From the application programmer's point of view, each agent is executed in its own thread and agent communication—either on the same host or on a remote host—must not interfere with the other agent's thread. But due to the tight resource constraints on embedded systems it is not guaranteed to have a dedicated thread for each agent. Moreover, a pool of threads is used to execute the agents as required.

The *Agent* interface which is mandatory for all agents defines a set of methods that allow the agency to pass over control to the agent. After agent creation, for instance, the agent has the opportunity to execute initialization code; other methods are for connection handling, message handling and the like. The central component, however, is the *AgentGovernor*. It acts as a mediator between an agent and the agency as well as agents itself. Each new agent instance gets assigned a dedicated *AgentGovernor*. From the agent's point of view, the AgentGovernor represents the agency; each request of an agent to the agency is intercepted by the AgentGovernor which in turn performs some plausibility checks before delegating the request to the agency. Requests from other agents (e.g. connection requests, incoming messages) as well as requests from the agency actually go to the AgentGovernor which enqueues the request in a kind of working queue. The requests are then handled by the agent in its own context.

Communication between agents is connection oriented, message based and basically asynchronous. Upon connection establishment the agent can assign a certain context to the connection in order to keep track of the current state of the communication and use this context information when processing incoming messages. Although the asynchronous communication is more flexible and allows communication with multiple agents in parallel, it is more difficult to use. Therefore, the AgentGovernor provides a synchronous communication top of the asynchronous channel.

4.2.2. Image Processing

Mobile agents are used to represent the image processing tasks in a smart camera network. The agent, however, is executed on the communication unit while image processing is done on the dedicated processing unit. Hence, agents are split into two parts. The image processing algorithm, on the one hand, is implemented as dynamic executable for the processing unit and is devoted to low-level pixel processing. On the other hand, the agent implements the application logic and controls the image processing algorithm. Of course, both parts can communicate with each other.

This strict separation into image processing part and application logic supports flexibility and modularity of applications. A tracking agent, for instance, may use different tracking algorithms depending on the application and environment with the same strategy for following the target over the camera network.

4.2.3. System Agents

System agents provide common services simplifying application development. The *NodeManagementAget* is somewhat distinguished as it has extended privileges for interacting with the agency. This agent, among others, provides information about agents currently residing on an agency, and allows to create new agent instances either on the local agency or on a remote agency. Each agency hosts exactly one instance of a NodeManagementAgent which can be accessed via a well-known name.

The *ImageProcessingAgent* provides an agent-oriented interface to the processing unit. Hence, other agents can communicate with algorithms on the processing unit in the same manner as with other agents. The message content is, of course, specific for the algorithm. Moreover, this agent allows to load and unload algorithms on the processing unit.

For certain applications, additional system agents may be deployed providing domain specific services. Distributed computer vision, for instance, usually requires knowledge of the camera parameters but also the topology of other cameras (e.g., position and orientation) in their vicinity. The *SceneInformationAgent* is used to manage this kind of information in a distributed manner.

4.2.4. Agent Mobility

An important feature is mobility of agents, i.e., agents can migrate from one agency to another. While this feature can be implemented easily in high-level languages such as Java or .NET, it requires some effort to implement similar behavior in C++, especially when taking into account different platforms. While other C++-based agent systems interpret the agent code (e.g., [25]), we want to execute the agents natively for efficiency reasons.

Our approach to support mobility of agents, therefore, is based on remote cloning. If an agent intends to migrate to another camera, it simply creates a new agent instance on the remote camera and passes it its current internal state for initialization. After successfully creating the new instance, the agent terminates itself. This requires, that the agent's code is already available at the destination. If this is not the case, the agent code can be loaded during runtime from a dynamically linked library which is obtained from a repository in the camera network. Remote cloning also supports different camera platforms in the network since agent communication is platform independent.

4.3. Middleware Services

We have stressed our middleware implementation in different case studies [34, 35] and identified common middleware services and domain specific middleware services. In the following we describe a selection of these services.

Dynamic task loading. Mobility of agents, and thus image processing tasks, requires to load and unload these tasks dynamically during runtime. Despite the low-level support integrated into the DSPFramework on the processing unit, this functionality has to be provided to the agents as well. In our case, a dedicated system agent, the ImageProcessingAgent, gives an abstract representation of the processing unit.

Camera configuration. In image processing applications it is important to know certain camera parameters such as the image resolution, spectrum of the captured images, camera geometry, etc. All this information is stored locally on the camera and provided through the SceneInformationAgent.

Network and Camera topology. For collaborative image processing the algorithms also require information about cameras in their vicinity. This may be a list of other cameras observing the same scene or neighboring cameras in a certain direction. The SceneInformationAgent is used to provide this information as well.

Resource monitoring. Resource monitoring keeps track of the utilization of available resources such as memory, processing power, DMA channels and communication bandwidth, among others. An image processing task can only be migrated to a certain camera if there are sufficient resources available.

Time synchronization. Collaborative image processing requires a uniform time-base for all cameras. Without a system-wide synchronized clock it would be impossible to combine results from different cameras. Therefore, a dedicated agent may be used to synchronize the clocks of each processor as well as the whole camera network.

Video streaming. Although the ultimate goal in surveillance applications is to unburden human operators from observing dozens of video streams, in some situations it is still required to deliver real-time video data, either for analysis by humans or archiving. New protocols and improved encoding algorithms offer new possibilities for streaming video data [35]. A dedicated agent can be used to provide such a streaming service for all agents in the system.

5. EVALUATION

We have implemented our agent-oriented middleware on our embedded SmartCam [3], PoQoCam [36] and the PC platform respectively. Since we target embedded platforms, a small memory footprint is desired. Table 1 lists the code-size of DSCAgents and its modules (stripped, cross-compiled binaries) which is less than 3.5 MB, including all libraries. Table 2 lists the memory consumption depending on the number of agents. After startup DSCAgents consumes approximately 2.5 MB of RAM and each agent accounts for approximately 10 kB (depending on the memory required by the agent itself). Note that in this case each agent has its own thread

Libraries		2680 MB
ACE	2060 kB	
Boost	442 kB	
SmartCam Framework	178 kB	
DSCAgents		885 kB
Total		3565 kB

Table 1: Code-size of DSCAgents and its components.

Number of Agents	*Allocated Memory*
0 Agents	2488 kB
10 Agents	2768 kB
30 Agents	3020 kB
50 Agents	3280 kB

Table 2: DSCAgents' memory consumption.

of execution which accounts for the greater part of the allocated memory. The execution times of frequent operations are summarized in Table 3 (average of 20 measurements). Creating a new agent on the same agency and on a remote agency takes about 4.5 ms and 10 ms respectively. Messaging between agents is also an important factor, especially for collaboration and migration of agents. The results are depicted in Figure 4. In case of local messaging it takes approximately 2 ms, independent of the message size. Sending messages to remote agents takes somewhat longer and also depends on the message size. For small messages (< 1000 Bytes, the interface's MTU) the transmission delay is almost constant but for larger messages the transmission time increases linearly.

6. CONCLUSION

Smart cameras are embedded devices with high performance on-board processing and communication capabilities. Recent work focuses on the integration of multiple smart cameras into a network and thus building a distributed system devoted to image processing.

In this paper we presented DSCAgents, an agent-oriented middleware for distributed smart cameras. We presented the architecture and modules of our proposed middleware and describe the core components, namely the agent layer and domain specific services, in greater detail. The evaluation presents performance measures of our implementation on the embedded smart cameras.

Although we demonstrated the applicability of our middleware approach in different case studies there is still lack of a widely accepted middleware approach for distributed smart cameras. Such a middleware should on the one hand be specialized for the application domain of smart camera networks but on the other hand general and modular to support many different smart camera platforms, ranging from low-performance camera motes to high-performance multi-processor platforms. It should also be applicable for various applications in the context of smart cameras such as single and multiple target tracking, ..., among others, and thus build a solid foundation for future application development.

Creating an agent	(local)	4.49 ms
	(remote)	10.11 ms
Loading image processing algorithm		17.86 ms

Table 3: Execution times for frequent operations.

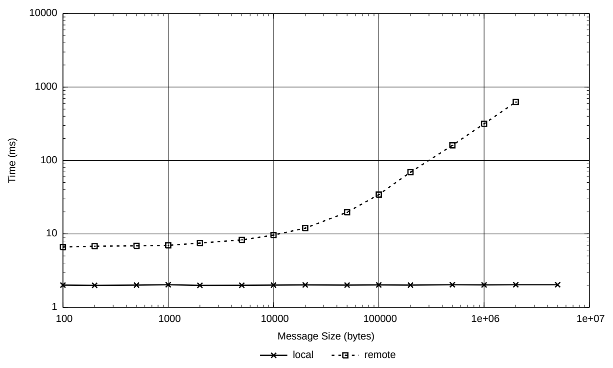


Fig. 4: Average messaging time.

7. REFERENCES

[1] Richard Kleihorst, Ben Schueler, and Alexander Danilin, "Architecture and Applications of wireless Smart Cameras (Networks)," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2007)*, Honolulu, Hawaii, USA, Apr. 2007.

[2] Wayne Wolf, Burak Ozer, and Tiehan Lv, "Smart Cameras as Embedded Systems," *Computer*, vol. 35, no. 9, pp. 48–53, Sept. 2002.

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

[4] Hamid Aghajan and Richard Kleihorst, Eds., *Proceedings of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 06)*, Vienna, Austria, Sept. 2007.

[5] Bernhard Rinner and Wayne Wolf, "Introduction to Distributed Smart Cameras," *Proceedings of the IEEE*, vol. 96, no. 10, Oct. 2008, to appear.

[6] Andrew S. Tanenbaum and Marteen van Steen, *Distributed Systems: Principles and Paradigms*, Prentice Hall, 2006.

[7] Ian F. Akyildiz, Tommaso Melodia, and Kaushik R. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, pp. 921–960, 2007.

[8] Markus Quaritsch, Bernhard Rinner, and Bernhard Strobl, "Improved Agent-Oriented Middleware for Distributed Smart Cameras," in *Proc. First ACM/IEEE International Conference on Distributed Smart Cameras ICDSC '07*, 2007, pp. 297–304.

[9] Bernhard Rinner, Milan Jovanovic, and Markus Quaritsch, "Embedded Middleware on Distributed Smart Cameras," in *International Conference on Acoustics, Speech, and Signal Processing*, April 2007, number 4, pp. 1381–1384.

[10] Allan Tengg, Andreas Klausner, and Bernhard Rinner, "I-SENSE: A Light-Weight Middleware for Embedded Multi-Sensor Data-Fusion," in *Proceedings of the 5th IEEE International on Intelligent Solutions in Embedded Systems (WISES 2007)*, Madrid, Spain, June 2007, pp. 165–177.

[11] R. Collins, A. Lipton, and T. Kanade, "A System for Video Surveillance and Monitoring," in *American Nuclear Society 8th Internal Topical Meeting on Robotics and Remote Systems*, 1999.

[12] Chiao-Fe Shu, A. Hampapur, M. Lu, L. Brown, J. Connell, A. Senior, and Yingli Tian, "IBM smart surveillance system (S3): a open and extensible framework for event based surveillance," in *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance,*, 2005, pp. 318–323.

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

[14] D. G. Schmidt and F. Kuhns, "An overview of the Real-Time CORBA specification," *Computer*, vol. 33, no. 6, pp. 56–63, 2000.

[15] "Minimum CORBA Specification," `http://www.omg.org/technology/documents/formal/minimum_CORBA.htm`, 2008, last visited: June 2008.

[16] Don Box, *Essential COM*, Addison Wesley, 2007.

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

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

[19] Robert S. Gray, George Cybenko, David Kotz, Ronald A. Peterson, and Daniela Rus, "D'Agents: applications and performance of a mobile-agent system," *Softw. Pract. Exper.*, vol. 32, no. 6, pp. 543–573, May 2002.

[20] Holger Peine and Torsten Stolpmann, "The architecture of the ARA platform for mobile agents," in *Mobile Agents*, pp. 50–61. Springer Berlin / Heidelberg, 1997.

[21] C. Bäumer and T. Magedanz, "Grasshopper — A Mobile Agent Platform for Active Telecommunication Networks," in *Intelligent Agents for Telecommunication Applications*, pp. 690–690. Springer Berlin / Heidelberg, 1999.

[22] Thomas Wheeler, "Voyager Architecture Best Practices," Tech. Rep., Recursion Software, Inc., March 2007.

[23] Paul Marrow, Manolis Koubarakis, Rolf-Hendrik van Lengen, Francisco J. Valverde-Albacete, Erwin Bonsma, Jesús Cid-Suerio, Aníbal R. Figueiras-Vidal, Ascensión Gallardo-Antolín, Cefn Hoile, Theodoros Koutris, Harold Y. Molina-Bulla, Angel Navia-Vázquez, Paraskevi Raftopoulou, Nikolaos Skarmeas, Christos Tryfonopoulos, Fang Wang, and Chryssani Xiruhaki, "Agents in Decentralised Information Ecosystems: the DIET Approach," in *Proceedings of the Artificial Intelligence and Simulation Behaviour Convention 2001 (AISB01), Symposium on Information Agents for Electronic Commerce*, 2001, pp. 109–117.

[24] Cefn Hoile, Fang Wang, Erwin Bonsma, and Paul Marrow, "Core Specification and Experiments in DIET: A Decentralised Ecosystem-inspired Mobile Agent System," in *Proceedings 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS2002), pp. 623-630, July 2002, Bologna, Italy*, 2002, pp. 623–630.

[25] Bo Chen, Harry H. Cheng, and Joe Palen, "Mobile-C: a mobile agent platform for mobile C-C++ agents," *Softw. Pract. Exper.*, vol. 36, no. 15, pp. 1711–1733, December 2006.

[26] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An Operating System for Sensor Networks," in *Ambient Intelligence*, pp. 115–148. Springer Berlin / Heidelberg, 2005.

[27] M. M. Molla and S. I. Ahamed, "A survey of middleware for sensor network and challenges," in *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*, 2006, p. 6 pp.

[28] A. J. N. Van Breemen and T. De Vries, “An Agent based framework for designing Multi-controller Systems,” in *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, Jeffrey Bradshaw and Geoff Arnold, Eds., Manchester, UK, 2000, pp. 219–236, The Practical Application Company Ltd.

[29] N. R. Jennings and S. Bussmann, “Agent-based control systems: Why are they suited to engineering complex systems?,” *Control Systems Magazine, IEEE*, vol. 23, no. 3, pp. 61–73, 2003.

[30] Carlos E. Pereira and Luigi Carro, “Distributed real-time embedded systems: Recent advances, future trends and their impact on manufacturing plant control,” *Annual Reviews in Control*, vol. 31, no. 1, pp. 81–92, 2007.

[31] Chien-Liang Fok, G. C. Roman, and Chenyang Lu, “Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications,” in *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, 2005, pp. 653–662.

[32] Dimitrios Georgoulas and Keith Blow, “Making Motes Intelligent: An Agent-Based Approach to Wireless Sensor Networks,” *WSEAS on Communications Journal*, vol. 5, no. 3, pp. 525–522, March 2006.

[33] Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach, “A Novel Software Framework for Power-Aware Service Reconfiguration in Multi-Reconfiguration in Distributed Embedded Smart Cameras,” in *Proceedings of the 12th IEEE International Conference on Parallel and Distributed Systems (ICPADS’06)*, Minneapolis, Minnesota, USA, July 2006, pp. 281–288.

[34] Markus Quaritsch, Markus Kreuzthaler, Bernhard Rinner, Horst Bischof, and Bernhard Strobl, “Autonomous Multi-Camera Tracking on Embedded Smart Cameras,” *EURASIP Journal on Embedded Systems*, vol. 2007, pp. 10, 2007.

[35] Markus Quaritsch, Mario Wiesinger, Bernhard Strobl, and Bernhard Rinner, “An Adaptive Multi-Purpose Transmission Scheme for H.264 Encoded Video in Wireless Networks,” in *International Symposium on Communication Systems, Networks and Digital Signal Processing*, June 2008.

[36] A. Maier, B. Rinner, W. Schriebl, and H. Schwabach, “Online multi-criterion optimization for dynamic power-aware camera configuration in distributed embedded surveillance clusters,” in *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, April 2006, vol. 1, pp. 307–312.