Ella: Middleware for Multi-camera Surveillance in Heterogeneous Visual Sensor Networks

Bernhard Dieber*†, Jennifer Simonjan*, Lukas Esterle*†, Bernhard Rinner*
Georg Nebehay‡, Roman Pflugfelder‡, Gustavo Javier Fernandez‡
* Alpen-Adria Universität Klagenfurt, Austria
Email: {firstname.lastname@aau.at}
†Lakeside Labs
Klagenfurt, Austria
‡Austrian Institute of Technology, Austria
Email: {firstname.lastname@ait.ac.at}

***Abstract*—Despite significant interest in the research community, the development of multi-camera applications is still quite challenging. This paper presents Ella - a dedicated publish/subscribe middleware system that facilitates distribution, component reuse and communication for heterogeneous multi-camera applications. We present the key components of this middleware system and demonstrate its applicability based on an autonomous multi-camera person tracking application. Ella is able to run on resource-limited and heterogeneous VSNs. We present performance measurements on different hardware platforms as well as operating systems.**

I. Introduction

Increasing interest in Visual Sensor Networks (VSN) [1] drives the development of multi-camera applications in private and public areas.

Developing distributed applications in a multi-camera sensor network often includes repetitive tasks like implementing sensor data capturing, transport or visualization. The implementation of infrastructural mechanisms like data transport, node discovery and node failure detection are typically time consuming and error prone. Doing this in each new application significantly increases development time. However, facilities for component reuse, communication and fault tolerance mechanisms can be outsourced to a middleware system.

In this paper we present a middleware with a special focus for smart-camera networks featuring heterogeneous hardware, software, and networking technology. We show how the dedicated middleware facilitates and speeds up application development. We show the applicability of our architecture in a concrete use-case of distributed multi-camera person tracking.

The remainder of this paper is organized as follows: in Section II we present relevant related work, in Section III we discuss the general properties of VSN applications and how they can be supported by a middleware and we also show our use case as an example, Section IV describes our middleware system in more detail, Section V outlines how the application for our use case is built on top of the middleware. Finally, we show an evaluation of the application and the architecture in Section VI.

II. Related Work

Middleware systems specially designed for VSNs aim at providing support for developers of VSN applications in speeding up and facilitating development.

Detmold et. al [2] present a multi-layered middleware for distributed video surveillance. This middleware however, builds on top of services like HTTP transport which cause a major overhead and make the middleware hardly suitable for embedded systems. Thus it builds upon an infrastructure of backend servers which handle CPU-intensive processing tasks.

In [3] the authors present a publisher-subscriber middleware for DSP-based smart cameras. It can be used for dynamic reconfiguration, i.e. for changing the tasks running in the system. Publishers provide data which is then consumed by subscribers. Both can reside either on the same DSP or on a remote processor connected via PCI. This middleware is extended in [4] and [5].

[6] presents an agent-oriented approach to middleware for DSP-based smart cameras. Here, autonomous agents are used to transfer processing functionality between nodes. In a multi-camera tracking case study, agents hop from one node to another in order to continuously track an object moving through the surveyed area. The authors present an evolved version of the middleware with improved performance characteristics in [7]. This middleware system is then further developed and presented in [8].

A publish-subscribe middleware for wireless sensor networks has been recently presented in [9]. A special feature of this system is that it is able to connect clusters of a sparse wireless sensor network by piggybacking data on mobile nodes (e.g. mobile phones) moving between clusters. However, its HTTP-based communication causes some overhead in the communication channel.

III. Multi-camera Applications in a Heterogeneous VSN

While there are many different application areas for Visual Sensor Networks, still many parts of these applications are similar to each other. Sensor data capturing, data transport or visualization remain the same in most cases. Thus, constructing applications from existing building blocks and only adding application-specific modules can drastically speed up the development and enables the reuse of well-tested and proven application parts.

In a Visual Sensor Network, data can be processed in several ways: within a single node using multiple modules (e.g. from the sensor to a processing module), between multiple nodes (e.g. data aggregation from multiple sensors) or from

a node to a sink (e.g. to display images). Ideally, this communication is done transparently to the application dependent where the receiver of data is located.

In order to be scalable as well as robust, newly started and failing nodes should be detected respectively. Nodes started during run should seamlessly integrate into the existing network and collaborate in the application, failing nodes must be compensated by the remaining cameras.

Heterogeneity often poses problems in developing VSN applications. Cameras may not be homogeneous in terms of hardware and software. Other nodes (like the operators PC) may be integrated into the network. Providing a platform independent substrate for developing VSN applications can reduce implementation time and minimize errors in the resulting application.

Use case: Person tracking in a heterogeneous VSN. In our case-study application, we perform distributed person tracking on multiple cameras where at any point in time, only a single camera is responsible for tracking the person. In the event that the person leaves the field of view (FOV) of the currently tracking camera or is otherwise lost, the camera initiates a handover during which the other cameras try to find the person. This is either done on all cameras, or on a subset of only neighbouring cameras. The handover is performed by means of an auction as described in [10]. In any case, each camera typically streams its currently captured video stream to an operator PC running a graphical user interface.

Looking at the global dataflow of this application, this means that the tracking module is not always active on every camera. Only after winning the handover auction, the tracker is activated.

Camera Network. In a heterogeneous network, the application must be runnable on different hardware platforms, make use of different network technologies and support different operating systems.

In our use-case we work in a multi-camera network environment which is heterogeneous in three different aspects: (i) camera and sensor hardware, (ii) operating systems and (iii) networking technology. First, we use Intel Atom-based smart cameras built by SLR Engineering[1]. As a second platform we use custom-built PandaBoard[2]-based camera systems. The operator application runs on a standard PC. While the smart cameras run a Linux derivative, the operator PC runs Windows 8.

The SLR cameras are equipped with an Intel Atom processor running at 1.6 GHz and an 100 MBit Ethernet interface. Furthermore, the SLR camera has a CCD image sensor with a native resolution of 1360×1024 pixels.

In contrast, the PandaBoard platforms are equipped with Logitech HD Pro c920 webcams. They are based on Texas Instruments OMAP 4430 system-on-chip which features a dual core ARM Cortex-A9 MPCore CPU running at 1.2 GHz and uses a 802.11 b/g/n wireless connection to the network. The connected Logitech c920 webcam operates with a native resolution of 3MP. The PandaCam is the current result of our efforts to build an energy-efficient smart camera with high computing power.

[1]www.slr-engineering.at

[2]http://www.pandaboard.org

IV. Middleware System

In order to be useful in the context of Visual Sensor Networks, a middleware must match several requirements besides simplifying application development. First, it must be robust enough to detect and possibly correct failures in nodes. Second, it must be flexible enough to allow for application restructuring and quick exchange of functionality. Thus, applications should be assembled from several modules instead of being monolithic. Third, it should be made easy for developers to port their existing code to the middleware system. Further the middleware should provide mechanisms to decouple individual modules, i.e. make them independent wherever possible.

We introduce *Ella*, a distributed middleware system implementing the publish/subscribe paradigm. Although, Ella is designed to be generic and independent from a specific application domain, it offers many features useful in VSN applications. Its componentized approach (dividing an application into single modules) enables the composition, extension and reconfiguration of applications using single building blocks. Further, it can handle all data and control communication and is able to construct the data flow in an application purely from subscription requests. The mode of distributed operation in a smart camera network can be very well supported by a middleware system. Typical operations like data transport, coordination, node discovery or failure detection can be supported by a middleware and do not need to be implemented in every application individually.

A. Publish/Subscribe

Publish/subscribe is an event based middleware paradigm, defining two different roles: (i) the publisher, which produces and publishes data, and (ii) the subscriber, which shows interest in events using subscriptions [11].

Publish/Subscribe, as illustrated in Figure 1, is a mechanism which allows for elegant decoupling of functional elements within an application. A module may be publisher and subscriber at the same time, i.e. it processes data from another publisher and publishes the results itself. A component for publish/subscribe management takes care of decoupling. Instead of directly connecting the publisher and subscriber modules, a publisher announces its events and a subscriber can indicate interest in certain types of events. The publish/subscribe manager takes care of matching published events and subscriptions and is also responsible for delivering the published data to all subscribers. A key requirement of publish/subscribe is that neither publishers nor subscribers need to be aware of each other. A publisher does not need to keep track of where its data is going and how many subscribers exist for its events, and a subscriber does not need to care about where publishers are located and where their data is coming from (i.e. the local node or a remote node). All this is transparently handled by the publish/subscribe middleware.

Figure 1 shows an approach for distributed publish/subscribe. Here, each node runs a local publish/subscribe manager. This manager keeps track of the subscriptions to its local publishers and of the other nodes in the network running the same middleware system.

1) Decoupling: A publish-subscribe system enables decoupling in the following dimensions:

- Space decoupling: Modules do not need to know where they and other modules are located in the

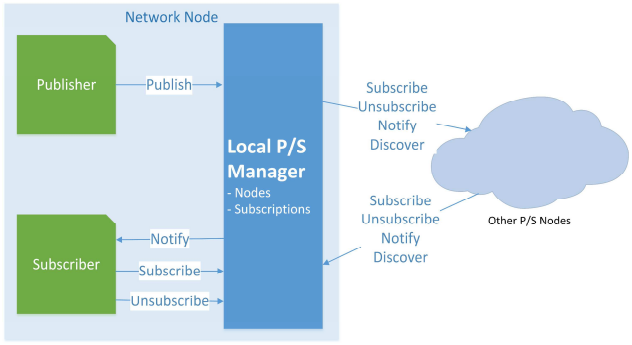


Fig. 1. A distributed publish/subscribe system. The local manager component provides operations for publishing, (un)subscribing and notifies subscribers of new events.

network. This means that publishers do not hold any references to subscribers and vice versa. In a VSN, e.g., a publisher of images does not need to care if they are delivered to one or more displays or other modules.

- Time decoupling: Publishers and subscribers do not need to participate in an interaction at the same time. The publisher might for instance publish an event while there is no subscriber connected. Publishers which start after a subscriber can still be matched to an earlier subscription request. In a VSN, cameras may not start up at the same time, still they must form one distributed application.
- Synchronization decoupling: Preparing events does not block the publishers, and subscribers can be notified of an event, even though they are currently executing another activity. As an example, a publisher of images can capture the next image while the current image is still delivered to subscribers.

The three forms of decoupling described above are especially important for applications in the VSN domain because they enable heterogeneous networks which are (i) scalable, (ii) flexible and (iii) fault tolerant.

B. Implementation Details

Since Ella is fully distributed, it provides a mechanism to discover other nodes in the network, that run Ella. This means, on startup Ella tries to find all other nodes in the network by performing an IP broadcast. Every node receiving this message, replies directly to finish this initialization process. Hence, every node is aware of all available remote nodes within the network.

The following sections provide an overview over the most important functions and parts of Ella.

1) Subscription System: Ella uses type-based subscription, i.e. a subscriber specifies a certain data type to subscribe to. By default, the subscriber will be subscribed to all matching publishers. As an addition, Ella provides the possibility to request a template object from each subscriber. The middleware will then ask each publisher (which is matching in type) to generate such a template object and will hand it to the prospective subscriber. The subscriber can then decide whether this specific publisher is accepted or not. Further, the subscriber can optionally disable the inquiry on remote nodes.

2) Network management and remote operation: To support a convenient way of developing and deploying software modules for Ella, the middleware provides a transparent node discovery mechanism which is used to detect any running Ella instances on other nodes in the network. This relieves the developer of the need for managing other nodes in the network. As soon as an Ella instance is detected, it is registered as a known host and it will also be checked for suitable publishers of events requested by local subscribers. This way, it is much easier to scale an existing application without having to modify existing code. As soon as Ella detects other instances, it will include them in its operation.

On startup, Ella tries to first discover other nodes in the network. By default this is done with a UDP broadcast. This broadcast also contains connection information necessary to address this node in the network. However, this may be exchanged with any other suitable discovery provider (e.g. for non-IP compatible media like ZigBee). Upon reception of a broadcast message a node will send a unicast answer to the broadcasting node with its own connection information. Thus, each node keeps a local directory of known remote hosts. This directory is used when searching for matching publishers on other nodes.

Whenever a subscriber requests a new subscription, all remote hosts will be inquired about matching publishers. If some are found, proxy objects at the remote node and stubs at the subscriber node will be created which act as transparent transport points for published event data. A proxy acts as subscriber at the remote node, serializes the event data and sends it to the stub. The stub deserializes it and publishes it as a local publisher for the original subscriber to receive.

The requested subscription types of each subscriber module are cached by the local Ella instance. Whenever a new node is discovered in the network, it will also be inquired about suitable publishers.

3) Communication: Ella instances on remote nodes use an efficient message structure to exchange data. A binary protocol is used to encode message types and to transport any necessary data. For any given message payload, only 9 bytes of overhead are added, one byte for the message type, and four bytes each for the sender node ID and the message ID. For small networks this can be reduced by only using single bytes for the sender node ID.

Besides data communication, Ella provides also a control channel which can be used to exchange application-specific messages between publishers and subscribers.

Ella has been developed in C#.Net. It is capable of running in the open source Mono[3] runtime and can thus be deployed on all major operating systems and many other platforms. Since it is only performing high-level tasks like I/O and management of subscriptions, its overhead compared to a native implementation is very low. The choice of a bytecode based language makes Ella modules inherently platform-independent (for purely managed modules). In addition, it is easily possible to integrate native code components into any .Net application. Thus, performance critical application parts can be written in e.g. C++ and be integrated into Ella with low effort. Of course, also pre-existing native code can be integrated but must be platform-specific.

Ella modules can either be automatically detected and

[3] http://www.go-mono.org

loaded at runtime, or may be explicitly instantiated by a host application and then passed to the Ella system.

V. Case Study: Multi-Camera Single Person Tracking

In this section, we present a case study of our architecture. We have built a multi-camera person tracking system which is able to track a selected person through the network and hand off tracking responsibility to neighbouring cameras whenever the object leaves the FOV of the camera currently tracking it. The cameras also transmit their current view along with the results of the tracker to an operator PC where this information is shown in a graphical user interface. Each module of our application is an Ella publisher and/or subscriber.

As depicted in Figure 2, three major modules are concerned in the realization of this system. The *CV* component is responsible for performing the computer vision tasks of acquiring images from the sensor and performing tracking as soon as the tracking responsibility is assigned to the respective camera node. It publishes the acquired images as well as tracking results indicating the location of the tracked object. The *Handover* component takes care of agreeing with other cameras on the tracking responsibility. In case a tracked object is about to leave the field of view of the camera, it will initialize an auction by publishing an *AuctionState* event. Cameras submit bids for the object using control messages. The camera with the highest bid will win the auction and will then be responsible for further tracking the object. The *UI* module is a visualization component for the security operator who will initially select the object to be tracked.

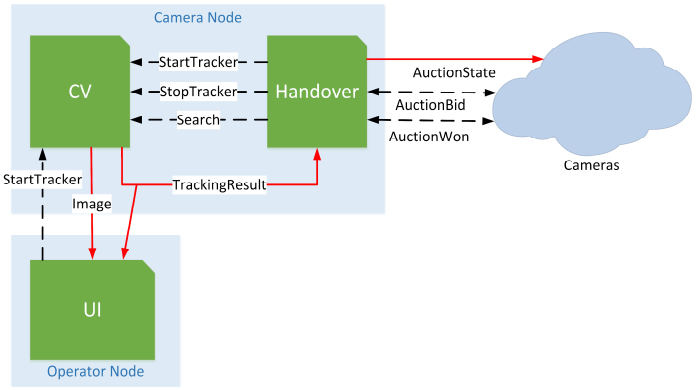


Fig. 2. The components in our application. Each camera runs a handover and a tracking module. The operator PC runs a user interface. Red solid lines indicate published events, black dashed lines indicate control messages.

Using the publish/subscribe mechanism and the control channel in Ella, this application can be realized with an elegant decoupling of components. First, *CV* is publishing images (to be displayed in *UI*) and tracking results (containing the location in the image and the confidence of the tracked object) in case it is responsible for tracking. Both, *UI* and *Handover* are subscribed to this tracking information. *UI* uses it to show the bounding box of the tracked object as overlay to the camera image stream. *Handover* is informed of the current tracking status with this information and can react to e.g. a lost object.

Handover publishes an auction state event which initializes and ends an auction in case a handover is necessary. If a camera wants to contribute to an auction with its own bid, it will use the control channel to directly address the auction initiator. *Handover* uses tracker control messages on the control channel to start and stop the tracker and to pass the model of the object to be tracked to the tracker. *UI* does the same once the operator has selected an object to track in one of the cameras' views.

A. Object Tracking

The native tracker and image acquisition implementations are wrapped in an Ella module called *CV*.

As the people tracking application is intended to run in realtime on low-power hardware, the design of the tracking algorithm is constrained by the need for low computational complexity. In the following, we briefly describe our current implementation.

In our current setup, cameras are mounted statically, therefore allowing background modelling algorithms to be used. As a first processing stage, we identify foreground pixels by employing the method of Schreiber and Rauter [12]. In the next step, we perform a labeling of connected components using the linear-time labeling algorithm from [13]. We then calculate the smallest bounding box around each component and refer to them as foreground regions.

For each foreground region, we compute a feature vector f by employing background-weighted histograms in RGB space using 8 bins for each color channels, leading to $n = 256$ bins in total. Background weighting is performed by computing

$$f = w \cdot h,$$

where h is the normalised histogram of the foreground region and w is a weight vector. We calculate w on the initial selection of a person by employing the method of Comaniciu et al. [14] in order to compute the individual components w_u of w by

$$w_u = \min\left(\frac{b_{min^*}}{b_u}, 1\right),$$

where b is the normalised histogram of the background computed on an empty image frame and b_{min^*} is the smallest non-zero element of b. The feature vector M that is calculated on the region selected by the user constitutes the model of the object of interest.

For associating foreground regions to the model, we employ the Bhattacharyya coefficient

$$d(f, M) = \sum_{i=1}^{n} \sqrt{f_i \cdot M_i}$$

followed by a thresholding $d(f, M) > \theta$. θ can be tuned in order to allow for a more conservative association, meaning that high values of θ lead to fewer false positives and more false negatives. By experimenting with different values of θ, we found that $\theta = 0.5$ provides a satisfing tradeoff. The association mechanism can be extended to tracking multiple objects by storing one model $M_1, \ldots, M_n$ for each object of interest.

During the development of this algorithm, we have experimented with possible alternatives to the RGB color space (LAB, HS) and to the Bhattacharyya coefficient as a distance metric (L2, EMD). However, all of our experiments showed that the combination of background weighted RGB histograms performs better than all other combinations. We found that image normalisation gives an additional improvement to our current implementation, but drastically increases the computational burden, so we did not include it.

Hardware	Running components	∅CPU	∅ Mem. Usage
SLR Camera	CV + Handover	71.7%	87 MB
PandaCam	CV + Handover	70.5%	106 MB
Desktop	GUI	2.6%	140 MB

TABLE I. PERFORMANCE MEASUREMENTS GENERATED IN THE SMART CAMERA DEMONSTRATOR.

VI. EVALUATION

In this Section, we present evaluations on our middleware, the application and the camera hardware used in our setup. We will show that the middleware-specific overhead is very low and that the application delivers significant performance in its operation.

We use both, the SLR cameras as well as the PandaBoard based systems and measure the CPU load and memory consumption during the execution. The graphical user interface of the application has been executed and measured on a desktop computer equipped with an Intel Core i7 running at 3.4GHz and 16 GB memory.

We use live data instead of test videos in order to include the overhead generated by capturing images directly from the sensor. While grabbing images from a video makes the results more reproducible, it is computationally less expensive and thus the measurement would not reflect the real application. We captured the load information during approx. one minute of operation with frames of 640×480 pixels. In these tests, three SLR cameras and two PandaCams were in use.

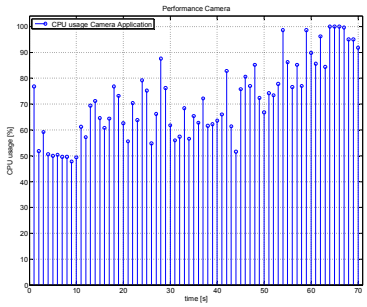
In a second experiment, we measure the overhead generated by the middleware and the application alone without measuring the load generated by image acquisition and tracking. To achieve this, we first generate tracker results for each camera from concurrently recorded videos. This processing is done offline. The results are fed in textual form of comma separated value files to each camera. A special module emulates the tracker by reading the values from the CSV file. On top of that, the application remains unchanged, the *Handover* component receives tracking results and performs the handover as usual. Further, no images are transferred to the user interface. In this configuration, we can measure the load generated by the handover component and the middleware.

A. Results

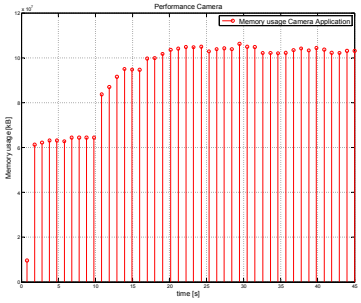
Table I shows an overview of the average CPU and memory load on each platform in the network.

The respective camera where we measure the load is at first not responsible for tracking but receives the tracking responsibility at a later point in time. This can be seen in the increased CPU and memory load. The PandaBoard (Figure 4) based cameras and the SLR cameras (Figure 3) show a comparable performance. However, our PandaBoard-based camera is also targeted at energy-efficient operation. It consumes only approx. 3 Watts in operation, while the SLR cameras consume up to 20 Watts. From this point of view, the PandaCam is a good choice for battery-based operation.

Figures 5 and 6 show the load generated by handover and middleware without the computer vision module running. The figures clearly show that the overhead generated by the middleware is very low, and thus it is very well suited for operation on low-power devices. Since it is written in a high-level language, it provides high developer efficiency at a

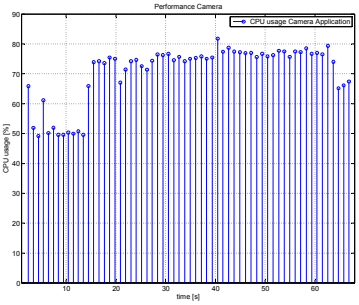


(a) CPU Utilisation

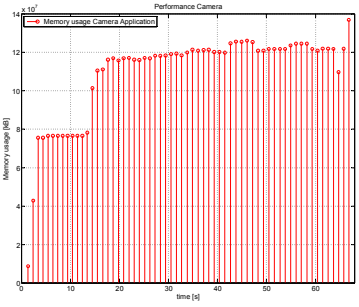


(b) Memory Usage

Fig. 3. CPU and memory performance of the SLR cameras.



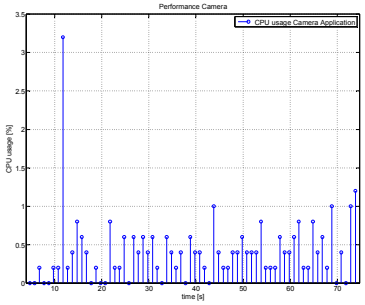
(a) CPU Utilisation



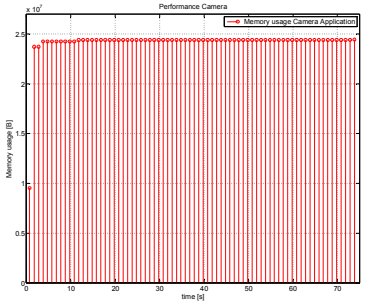
(b) Memory Usage

Fig. 4. CPU and memory performance of the PandaBoard based cameras.

reasonable performance cost.



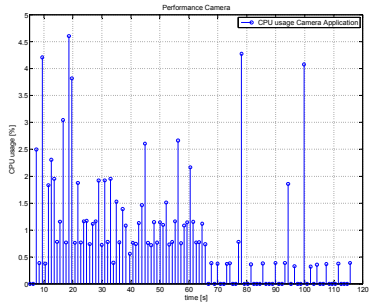
(a) CPU Utilisation



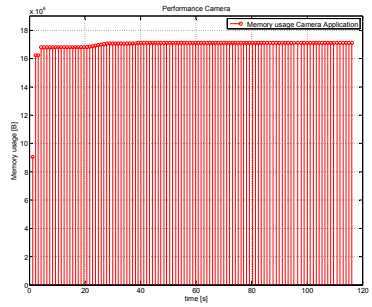
(b) Memory Usage

Fig. 5. CPU and memory performance of the SLR cameras without the *CV* module running.

[4]http://ella.codeplex.com



(a) CPU Utilisation



(b) Memory Usage

Fig. 6. CPU and memory performance of the PandaBoard based cameras without the *CV* module running.

VII. Conclusion

In this paper, we have presented Ella, a publish/subscribe middleware to ease development of multi-camera applications. Ella reduces implementation effort by providing facilities for standard tasks in a VSN application but still has a low resource footprint. In a usecase we showed the performance of our system as a whole and of single components. Ella has recently been released as an open-source project[4].

Acknowledgments

This work is supported by Lakeside Labs GmbH, Klagenfurt, Austria and is funded in part by the European Union Seventh Framework Programme under grant agreement n° 257906.

References

[1] B. Rinner and W. Wolf, "Introduction to distributed smart cameras," *Proceedings of the IEEE*, vol. 96, no. 10, pp. 1565–1575, October 2008.

[2] H. Detmold, A. van den Hengel, A. Dick, K. Falkner, D. S. Munro, and R. Morrison, "Middleware for distributed video surveillance," *Distributed Systems Online, IEEE*, vol. 9, no. 2, p. 1, feb. 2008.

[3] A. Doblander, B. Rinner, N. Trenkwalder, and A. Zoufal, "A lightweight publisher-subscriber middleware for dynamic reconfiguration in networks of embedded smart cameras," in *Proceedings of the 5th World Scientific and Engineering Academy and Society International Conference on Software Engineering, Parallel and Distributed Systems*, 2006.

[4] A. Doblander, A. Maier, B. Rinner, and A. Zoufal, "An efficient middleware for power-aware service reconfiguration in multi-dsp smart cameras," in *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, vol. 2, 0-0 2006, pp. 2985 –2990.

[5] M. Jovanovic and B. Rinner, "Middleware for dynamic reconfiguration in distributed camera systems," in *Intelligent Solutions in Embedded Systems, 2007 Fifth Workshop on*, june 2007, pp. 139 –150.

[6] B. Rinner, M. Jovanovic, and M. Quaritsch, "Embedded middleware on distributed smart cameras," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007.*, vol. 4, april 2007, pp. IV–1381 –IV–1384.

[7] M. Quaritsch, B. Rinner, and B. Strobl, "Improved agent-oriented middleware for distributed smart cameras," in *Distributed Smart Cameras, 2007. ICDSC '07. First ACM/IEEE International Conference on*, sept. 2007, pp. 297 –304.

[8] M. Quaritsch and B. Rinner, "Dscagents: A lightweight middleware for distributed smart cameras," in *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, sept. 2008, pp. 1 –8.

[9] X. Tong and E.-H. Ngai, "A ubiquitous publish/subscribe platform for wireless sensor networks with mobile mules," in *Proceedings of the 2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, may 2012, pp. 99 –108.

[10] L. Esterle, P. Lewis, M. Bogdanski, B. Rinner, and X. Yao, "A socio-economic approach to online vision graph generation and handover in distributed smart camera networks," in *Distributed Smart Cameras (ICDSC), 2011 Fifth ACM/IEEE International Conference on*, aug. 2011, pp. 1 –6.

[11] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003. [Online]. Available: http://doi.acm.org/10.1145/857076.857078

[12] D. Schreiber and M. Rauter, "GPU-based non-parametric background subtraction for a practical surveillance system," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE, Sep. 2009, pp. 870–877. [Online]. Available: http://dx.doi.org/10.1109/iccvw.2009.5457610

[13] F. Chang, "A linear-time component-labeling algorithm using contour tracing technique," *Computer Vision and Image Understanding*, vol. 93, no. 2, pp. 206–220, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2003.09.002

[14] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564–577, May 2003. [Online]. Available: http://dx.doi.org/10.1109/tpami.2003.1195991