# A Resource-Aware Distributed Event Space for Pervasive Smart Camera Networks

Wolfgang Schriebl
Pervasive Computing/Institute of Networked and Embedded Systems (NES)
Lakeside B02b
9020 Klagenfurt, Austria
wolfgang.schriebl@uni-klu.ac.at

Bernhard Rinner
Pervasive Computing/Institute of Networked and Embedded Systems (NES)
Lakeside B02b
9020 Klagenfurt, Austria
wolfgang.schriebl@uni-klu.ac.at

## ABSTRACT

Pervasive smart cameras (PSC) are an emerging technology with the goal of providing user-centric and ubiquitous visual sensor networks. System autonomy and resource-awareness are challenging requirements making local image processing for event-based communication a necessary design constraint. In this paper we present the distributed event space (DES)—a middleware service for the resource-aware management of distributed event data. The DES architecture is based on a decentralized tuple space which describes local events by tuples consisting of position and time of the events as well as a set of features describing the detected objects. The DES supports prompt distribution of detected local events and application-specific functions for sophisticated event filtering. We present a distributed tracking application to demonstrate its applicability.

## Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; I.4.9 [**Image Processing and Computer Vision**]: Applications

## General Terms

Design

## Keywords

Smart Cameras, Middleware, Event Distribution, Distributed Person Tracking

## 1. INTRODUCTION AND MOTIVATION

The concept of *Pervasive Smart Cameras* (PSC) [6] is driven by the quest for user-centric applications on ubiquitous camera networks. Evolved from distributed smart cameras, a PSC network consists of cost effective camera nodes which can easily be deployed and applied by non-expert users. Such capabilities require a high autonomy, a decentralized network architecture and power-aware hardware and software design.

PSCs are related to visual sensor networks; these networks are typically built from resource-constrained camera nodes connected over low bandwidth wireless networks [9, 7]. As a consequence of this strong resource limitation, the image processing must be distributed in the network and raw data transfers among nodes must be avoided as much as possible. Thus in a typical PSC application, the camera nodes perform image processing and store the data of detected events locally. Parts of this local event data is communicated over the network to derive a global event state which can then be communicated to the user. Object detection and tracking is a prominent example for such a PSC application where the camera nodes perform detection and tracking of objects within their field of view (FOV). Location and tracks of objects in the monitored area can then be determined by jointly checking the local event data stored in the camera nodes.

This paper focuses on the resource-aware management of event data in PSC networks, i.e., we develop efficient methods for storing, accessing and distributing event data on the camera nodes. We integrate these methods in our middleware framework [8] which is now able to provide a *distributed event space (DES)* over the PSC network. To demonstrate the feasibility of the DES, we have implemented a distributed person tracker on our PSC network of uncalibrated cameras. The user interface for this tracking application is realized on hand-held devices. Thus, users can send requests on the DES and check the status of the PSC with their mobile devices.

The DES is based on the concept of *tuple spaces* [3, 4]. The local event data is represented by a tuple consisting of an identifier, a set of features and some meta data describing the event. To achieve a common event space these event tuples must be distributed in the network. There are two principle distribution methods possible: In the *push approach*, the camera nodes initiate the tuple distribution whenever a new event has been detected. In the *pull approach*, the transfer is initiated after a request has been received, and only tuples which match with the request are transferred. The push approach supports prompt event tuple updates but may introduce a high network traffic. Our implementation of the DES is therefore based on the pull approach.

The rest of this paper is organized as follows. Section 2 sketches related work on middleware support for peer-to-peer communication and object tracking PSC. In Section 3 we present the architecture of the PSC platform used in this paper, the network architecture and the middleware. Section 4 describes the architecture of the distributed event space and introduces available operations. Section 5 presents a distributed person tracking application on our PSC network to demonstrate the applicability of our DES. Section 6 concludes the paper and gives an outlook to future work.

## 2. RELATED WORK

For giving an overview of how event distribution is handled in visual sensor networks, we point to selected related work including camera networks applied for distributed object tracking, middleware for camera networks and middleware for sensor network based on tuple-spaces.

In decentralized camera networks, information about disjoint views as well as typical object tracks can be used to coordinate object information in the network. In [10] Velipasalar et al. present a scalable peer-to-peer camera system for tracking multiple objects. Each camera autonomously performs tracking of objects in the local FoV and record object information in its local storage. Based on FoV lines, cameras with a joint view are selected, and position and label of objects are requested when necessary. For communication, non-blocking messages sent via MPI are used. The communication overhead for the distributed tracking is small, but as MPI is a statically configured system, it does not support mobile or semi-static nodes. In [11] very similar concepts are mapped onto radio-enabled smart cameras. The nodes are based on CITRIC boards, which are powered by an ARM processor. TelosB, an MSP430 based sensor board is attached to the baseboard for enabling 802.15.4-radio. Short messages are used to exchange label and position of objects between cameras with a joint view. The network is evaluated by tracking RC cars and by defining regions-of-interest for raising events. In [5] Quaritsch et al. applied a distributed smart camera architecture for autonomous tracking. The camera nodes are equipped with an ARM and multiple DSP processors, the latter used for image processing and analysis. The middleware of the network is based on a mobile agent framework, which allows deploying software agents running autonomously in the network. For distributed tracking, each detected object is assigned to an agent which follows the object when moving to another nodes. For efficient migration, knowledge about the structure of the network and the path of objects is used. The evaluation shows substantial overhead for migration, which makes this approach hardly feasible for resource-constraint networks.

*Tuple spaces* [3] are a concept for distributed associative memory using data tuples as the basic entity for storage. While the original Linda tuple space was organized using a central server, several decentralized middleware systems based on tuple spaces where demonstrated for sensor networks. The LIME model [4] proposed by Murphy et al. adds support for mobile nodes to the Linda tuple space. A mobile node has access to an interface tuple space, which contains tuples physically co-located with the host as well as tuples stored on nodes in communication range. LIME furthermore defines reactions, which allow to specify a code segment which is executed when a matching tuple is found in the shared tuple space. In the TeenyLime model [2] nodes share tuples within a single hop only, which gives every node another view on the tuple space. The restriction of the view to single hop distance maps well to many applications in wireless sensor networks, but limits applicability for visual sensor network. In [12] Welsh et al. proposed a middleware system providing operations for abstracting communication, data sharing and collective operations. The operations are applied on abstract regions, which include nodes sharing a common property, e.g., radio connectivity.

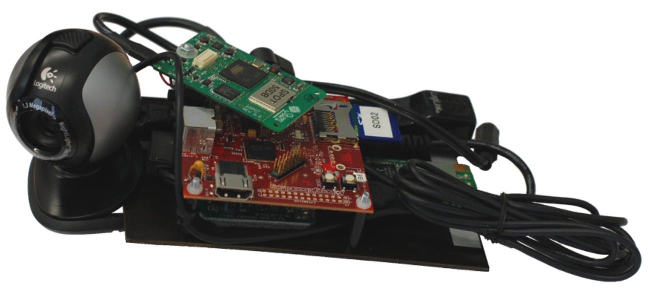

**Figure 1: The prototype of the PSC camera node consists of an embedded board, a VGA camera and networking peripherals connected via USB. The camera nodes are further equipped with Ethernet and Power-over-Ethernet to ease application development and testing.**

## 3. PSC NETWORK ARCHITECTURE

The PSC network consists of dedicated camera nodes interconnected in a peer-to-peer manner using wireless short- and mid-range network technology.

### 3.1 Node Architecture

The camera nodes (fig. 1) are based on the *Beagleboard* extended by off-the-shelf video- and network facilities connected via USB. The principle design is motivated by flexibility and extensibility needed for research and development.

The Beagleboard is powered by an OMAP-3530 dual-core processor, composed of a 480Mhz Cortex-A8 ARM and a 400MHz C64x+ DSP. The DSP supports relieving the general-purpose core for image analysis, processing and compression tasks. The on-board memory comprises 256MB SDRAM and 8GB of SDHC-type flash memory for storing operating system, programs and data. Video input is delivered by a VGA-resolution webcam in YUV422 and Motion-JPEG formats. Our camera node currently provides 802.11 WiFi and 802.15.4 LowPan wireless network connections and runs a Debian ARM based GNU Linux distribution modified with an OMAP3 specific kernel and some libraries using the floating point and SIMD capabilities of the ARM.

### 3.2 Network Architecture

Figure 2 shows the principle architecture of the PSC network. The core network includes semi-static camera nodes connected in a P2P-manner via ad-hoc WiFi and LowPan within a single hop. WiFi and LowPan are provided in parallel to exploit the advantages of both networks, i.e., lower start-up time and lower power consumption for LowPan and higher range and bandwidth for WiFi [13]. A dedicated node acts as gateway to the Internet and as access point for consumer nodes via 2G/3G services.

To interact with the network, hand-held consumer devices

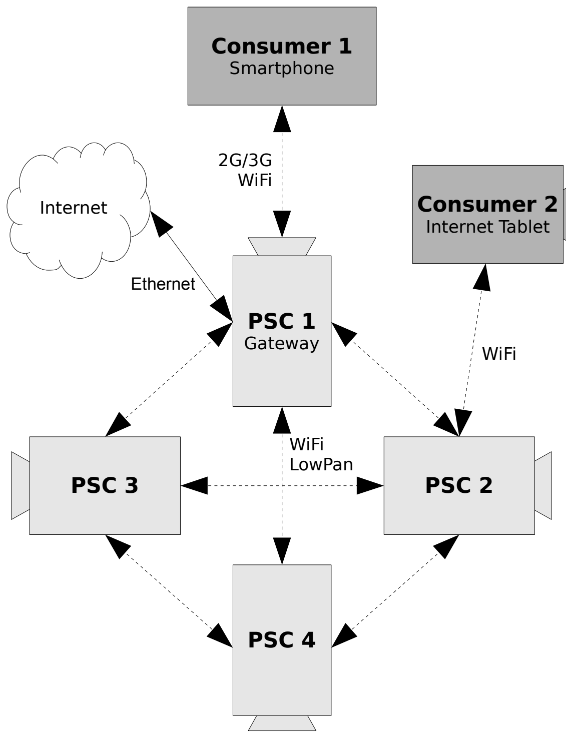

**Figure 2: A typical set-up for the PSC network. The core network consist of PSC nodes (*PSC1*, ..., *PSC4*) connected in a peer-to-peer manner using WiFi and LowPan. *PSC1* also acts as gateway to the Internet and as an access point for *Consumer1* via 2G/3G service such as GSM-SMS. *Consumer1* is a smart phone based user device running no compatible middleware layer and connects directly with applications at a camera node. *Consumer2* is an Internet tablet device connected via WiFi. It is fully integrated into the PSC network by running a PSC compatible middleware.**

can be connected with the PSC network. The architecture of consumer nodes is not restricted to any specific hardware; they must basically provide network connectivity, input facilities and a camera for vision-enabled services. We use (1) x86 notebooks running Linux, (2) Nokia N810 Internet tablets running Maemo Linux and (3) recent smart phones running Windows Mobile or Symbian as such hand-held devices. Depending on the provided software environment, the hand-held devices can either act as camera nodes running compatible middleware or are connected with the network via the gateway node at application level.

## 3.3 Software Architecture

The software of the camera nodes is based on a modified ARM Linux distribution. On top of the operating system, a middleware framework provides infrastructure for developing dataflow-oriented applications. The PSC middleware framework defines reusable processing blocks which consume and produce data. Producer and consumer blocks are connected via shared memory which is automatically transmitted when blocks are running on different physical platforms. In [8] we demonstrated this middleware for distributed person detection, where soft decisions of distributed classifiers are aggregated on a single camera to get a final hard decision.

The middleware framework is available for Linux as library for C/C++ and Python, and can be adopted to new platforms very easily.

## 4. DISTRIBUTED EVENT SPACE

Monitoring and security applications for distributed camera networks are usually based on local object detection and analysis of object tracks. An object detected in the camera's FOV is typically represented as an event which contains location and time information as well as a description of the object. In most applications we are not interested in every event, but want to select and abstract them based on some criteria imposed by the application. For example, a person tracking application can be configured to show only the trace of a particular person.

In our PSC middleware, applications are currently designed by connecting processing blocks which consume or produce data. This connection between blocks located on the same or distant nodes is transparently realized via a "virtual" shared memory, which is very efficient transmitting large amounts of data. With this mechanism a consumer-block can receive data from all cameras, but a data selection based on some associative matching is not supported.

To support associative memory, we extended the PSC middleware by a *distributed event space (DES)*. A DES is a resource-aware distributed storage providing multiple writers and readers a facility to exchange events.

### 4.1 Architecture

A DES is a virtual storage, containing event data which belong to the same class of objects. Physically, the event data is stored in *event spaces* located on different nodes in the network. The DES abstracts all event spaces to a joint storage, and provides an API to transparently write and read event data independently of the node they are located on. For running multiple DES in the PSC network, all related event spaces share the same name which is also used by applications to refer to a concrete instance of the DES.

Each event space stores events in a circular buffer. The size of the circular buffer is defined by the application depending on the number of stored events needed as well as on the available resources, and can vary for event spaces belonging to the same DES. When inserting a new event into the circular buffer, the oldest element is automatically overwritten, which makes a dedicated delete operation unnecessary. Beyond storing events, the local event space also holds a list of appending readings from consumer-applications, applied on newly inserted events for supporting reactions. Event spaces are never manipulated directly by the application, but always by the DES.

An event or event-tuple is defined as a set of primitive data types fitting the requirements of vision-based applications. Every event contains an unique identifier, the id of the node, the point of time when the event appeared, a feature vector identifying the object and metadata with optional information:

$$event := \langle uid, nodeid, time, \{featv\}, metadata \rangle$$

Depending on the application, fields can be added to allow filtering on additional data. Events belonging to the same class share the same primitive types, as well as the dimension and the semantics of the feature vector.

Figure 3 shows an example for a network of four nodes, including three cameras running person-detectors as producer-applications and a consumer device running a tracker as the consumer-applications. The nodes share information about detected persons using the DES named "Persons". While in this example consumer- and producer-roles are assigned to separate applications, applications can also take both roles in parallel.

## 4.2 Operations

The DES adopts two basic operations from the tuple space, namely **out**$(t)$ for synchronous writing and **rdp**$(t)$ for non-destructive asynchronous reading. Deletion is done automatically by the circular buffer, therefore a dedicated deletion function like **in**$(t)$ is not provided. This also avoids updating of events stored in the event space.

### *Writing Operation.*

The writing operation adds an event-tuple to the DES. The DES directly delegates the tuple to the local event space without involving the network, thus a blocking call is efficient and eases error handling.

The operation $DES.\mathbf{out}(event)$ takes an *event* as a parameter, which gives object-related information as well as location, time and unique-id of the event. By the DES, the event is forwarded unchanged to the local event space by calling $EventSpace.\mathbf{out}(event)$. Two fields, *nodeid* and *uid* can not be specified by the caller, but are replaced with the id of the local node and by an automatically generated value, respectively. Time, which is defined as the number of seconds past a common origin, can be given explicitly or as a wildcard ***, which is replaced automatically with current time. After completion, the event-tuple is matched with persistent readings and added to the circular buffer.

### *Reading Operation.*

The reading operation reads an event from the DES without removing it. In difference to the writing operator, the reading operator is not limited to the local event space, but can also be sent to other event spaces in the network. Furthermore, a reading can stay persistently at the event space allowing reactions on future events. For avoiding blocked applications while waiting for results, readings are supported asynchronously only.

A tuple is read by a consumer-application using the operation $DES.\mathbf{rdp}(event)$. Which tuples to read is defined by *event*, which contains a template used for matching. The DES forwards the template to the local event space, as well as to other event spaces located on nodes in the network. After receiving a template via $EventSpace.\mathbf{rdp}(event)$, the event space matches all entries in its circular buffer and sends matching tuples back to the caller.

Each primitive of a template defines a pattern used for matching the corresponding primitive of an event-tuple. For a positive match of an event-tuple, all primitives have to match positively. Beyond concrete values for matching equality and wildcards known from Linda tuple space, DES also supports comparative operators and user-defined functions. User-defined functions are registered at the local event space at runtime. For example, for comparing feature vectors a function *bd* calculating the *Bhattacharyya distance* of two vectors can be specified. By using the function bd, the following template positively matches all events which appeared at node 5, at a point of time prior 7654, with a feature vector having a Bhattacharyya distance smaller or equal .15 compared to $\{.11, .15, .89, .12\}$:

$$\langle *, 5, < 7654, \$bd(\{.11, .15, .89, .12\}) \leq .15, * \rangle$$

In a template, the primitives *nodeid* and *time* are not only used for matching, but also influence the way how the reading is distributed and handled. When defining a concrete node-id then the DES distributes the reading to the specified node only. Therefore, by setting *nodeid* to the id of the local node, sending the reading to the network is avoided. The time, on the other hand, allows registering persistent readings for automatic reactions. A template is kept persistently by an event space until the time specified by the template does not match the current time or any time in future, e.g., setting time to $< 1000$ keeps the template persistent until current time becomes 1000, setting time to $> 10000$ keeps the template persistent forever independently of the current time. Persistent readings can be updated or deleted by calling reading with the same template but a different *time*-primitive to the DES.

## 5. DISTRIBUTED PERSON TRACKING

To demonstrate the applicability of the DES, we implemented an application for distributed person tracking. In the network, cameras track persons in their local FoV and store information about detected persons to the DES. By querying the DES, a user equipped with a mobile device can get the current location as well as a cell-based track of specific persons.

## 5.1 Distributed Event Space

For realizing the cell-based tracking, the user application needs information about which person was detected at which time at which camera. These information can be mapped directly to the primitives of the event-tuple used by the DES, which allows to adopt the DES very easily for this application. The event tuple is in detail:

$$\langle *, location, time, \{f_0 \ldots f_{15}\}, * \rangle$$

The name of the DES, *PERSTRACK*, is shared between all parts of the application involved by the tracking. The name allows the local trackers running on the cameras to put detected events into the DES, as well as the user application running on a mobile device to consume the events.

## 5.2 Local Person Tracking

Several prominent approaches for robust tracking in a multi-camera network can be found in literature, e.g., [5, 11, 14]. When considering the uncalibrated sensors as well as the resource-constraints of the PSC network, an approach using simple features for local tracking as well as for person matching is advisable. We selected color-histogram features demonstrated in [8] for multi-camera person classification, and adopted them for meeting the requirements of the tracking application.

On each camera, a local tracking application applies object detection and feature extraction on QVGA sized input frames. For each frame, foreground blobs are extracted by frame differencing with a simple running-average background model. To remove noise pixels and to connect small

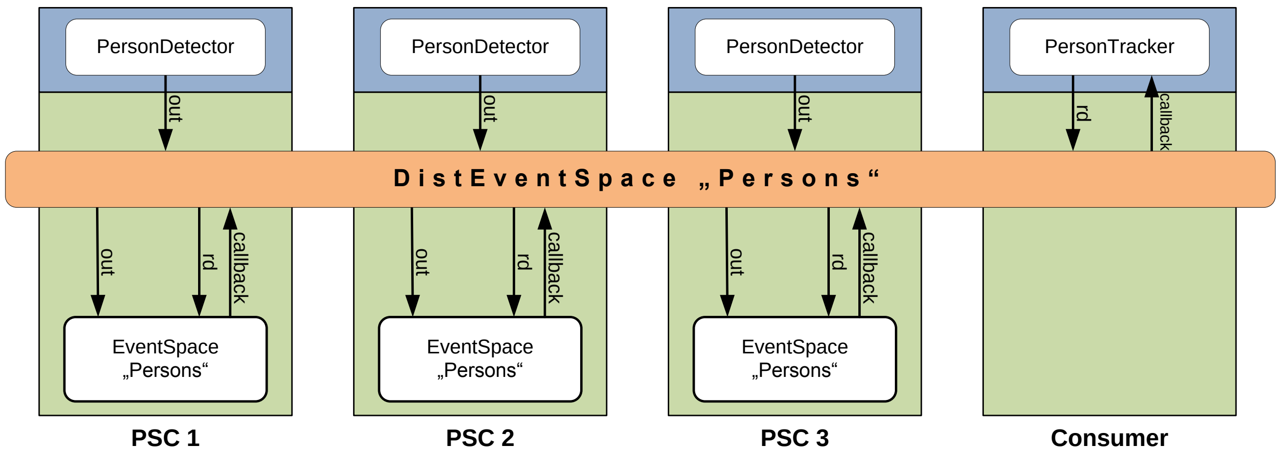

**Figure 3: A distributed event space used for exchanging information about detected persons. Three person detectors running on the camera nodes PSC1, PSC2 and PSC3 store events to the DES "Persons". These informations are read by the person tracker, running on the consumer node. As the person tracker acts as consumer only, no local event space is hosted by the consumer node.**

foreground objects, opening and closing operators are applied on the result. Blobs covering a minimum pixel area are tagged as persons and the feature vector as well as additional properties are determined. The final output of the person detection stage of a frame is a list of person objects.

For each person moving in the FoV, the tracking application instantiates a separate tracker handling exactly that person throughout visibility. The list of person objects of a new frame is compared to running trackers based on position (Euclidean distance) and features (Bhattacharyya distance). When a person object matches a tracker, the tracker updates its internal state and the person object is removed from the list. If a person object cannot be matched to any running tracker, a new tracker handling the person object is instantiated. Running trackers which are not updated within a frame are deleted.

For cell-based tracking, distributing the position of a person once per node is sufficient. Therefore, the local tracking application adds an event-tuple to the DES just after a new person enters the FoV. The feature vectors for persons newly appearing are usually not very distinctive, which is caused by having only a partial or very distant view on the person as well as by inaccuracies in segmentation. To improve discriminativity, the tracker averages the feature vector with each update, and delays writing to the DES for 6 consecutive updates.

The tracking application is implemented using the dataflow model presented in [8], using C++ and the IVT library [1], an image processing library optimized for fixed-point architectures.

## 5.3 Object Features

For local object tracking as well as for feature-based matching in the DES, features derived from the color distribution of the objects are used. The color of an object is very tolerant to scale and rotation and therefore good suitable for a non-calibrated environment. To reduce the effects of illumination (amount, distance and direction), features are based on the chrominance channels without luma component of the color model.

The feature vector $v$ is composed by the normalized bin values of the 2D color histogram spanned by chrominance-blue ($Cb$) and chrominance-red ($Cr$) shades of the object. In difference to a multi-modal histogram which contains separate histograms for each channel, a multi-dimensional histogram combines the distribution of pixels belonging to a given combination of color-values, e.g., $Cr = 0.33$, $Cb = 0.57$, in a single histogram. The resolution for both axis is defined as 4, leading to 16 bin values. Each value represents the percentage of the area of the object belonging to an interval of a Cb and Cr combinations, e.g., $v_0 := \#pixels|0.0 \leq Cb_i < 0.25, 0.0 \leq Cr_i < 0.25$.

## 5.4 User Application

The user of the distributed object tracking application is equipped with an vision-enabled mobile device. The device, a Nokia N810, is connected with the PSC network via ad-hoc WiFi and runs the same middleware, which enables direct access to the DES. An application offers the user a graphical representations of cell-based tracks of persons. Persons can be selected by taking an image using the integrated camera, or by loading an image from a file.

After grabbing or loading an image, the user masks the region containing the person-of-interest. The application determines the feature vector using the color histogram as used by the local trackers, and creates a tuple-template for reading from DEP. For matching, the Bhattacharyya distance with a maximum distance of .10 is chosen. For enabling persistent readings, the time of the event-template is set to 60 seconds in future and is committed to the DEP. After the event is distributed in the network, the event spaces on the camera deliver all events with a matching feature vector to the user application. With help of the node-id and the time the user application can graphically represent the path of the object.

## 6. CONCLUSION AND FUTURE WORK

In this work we presented the distributed event space (DES), a distributed storage used for exchanging visual events in resource-constraint visual sensor networks. Inspired by tuple spaces, the DES stores data organized in event-tuples which can be managed using simple operations. To demonstrate the applicability of the DES, we applied it for exchanging events in a person tracking application. The application is deployed on a PSC network, and is controlled by a user equipped with a hand-held device running compatible middleware.

## 6.1 Future Work

While the DES presented in this paper provides a solid base for resource-aware data exchange, several questions related to middleware in PSC networks are open.

*Clustering and Data Aggregation.* The proposed distributed event space was designed for single-hop systems, which limits scalability and the size of the monitored area. Ongoing work deals with multi-hop routing and clustering. While clusters based on the network topology are essential for network performance, dynamic clustering by a joint view on an object are interesting for data aggregation and joint processing. In future work clustering and multi-hop communication are investigated with the aim to extend the DES by cluster-based addressing and data aggregation.

*Resource-Aware Middleware.* Even though the distributed event space is resource-aware in terms of communication overhead, systemwide concepts for power optimization are not addressed, e.g., network interfaces of consumers always stay in listening mode. Starting with an analysis of the energy consumption of of the current system, in ongoing work approaches for a power efficient middleware for PSC networks are developed.

*Middleware Services.* The distributed event space is used for distributing visual events only, but could also be used for providing data needed for middleware services, e.g., calibration, network configuration or node management. In future work we demonstrate reasonable extensions to the DES, supporting exchange of generic data for providing middleware services.

*Multiple Platform Support.* The PSC middleware is implemented using C++ and Python and is currently available for a wide range of Linux-powered devices. To allow users equipped with general-purpose smart phones interacting with the PSC network, concepts for running the middleware on platforms with a restricted system API are investigated in further work.

## 7. REFERENCES

[1] P. Azad, T. Gockel, and R. Dillmann. *Computer Vision: Principles and Practice.* Elektor-Verlag, 2008. 320 pages.

[2] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco. TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks. In *MidSens '06: Proceedings of the international workshop on Middleware for sensor networks*, pages 43–48, New York, NY, USA, 2006. ACM.

[3] D. Gelernter. Generative Communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[4] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A Middleware for Physical and Logical Mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, volume 0, page 524, Los Alamitos, CA, USA, April 2001. IEEE Computer Society.

[5] M. Quaritsch, M. Kreuzthaler, B. Rinner, H. Bischof, and B. Strobl. Autonomous Multicamera Tracking on Embedded Smart Cameras. *EURASIP Journal on Embedded Systems*, 2007, 2007. Article ID 92827, 10 pages.

[6] B. Rinner, T. Winkler, W. Schriebl, M. Quaritsch, and W. Wolf. The Evolution from Single to Pervasive Smart Cameras. In *Proc. 2nd ACM/IEEE International Conference on Distributed Smart Cameras ICDSC '08*, 2008. 10 pages.

[7] B. Rinner and W. Wolf. Introduction to Distributed Smart Cameras. *Proceedings of the IEEE*, 96(10):1565–1575, October 2008.

[8] W. Schriebl, T. Winkler, A. Starzacher, and B. Rinner. A Pervasive Smart Camera Network Architecture applied for Multi-Camera Object Classification. In *Proc. 3st ACM/IEEE International Conference on Distributed Smart Cameras ICDSC '09*, Aug. 2009. 8 pages.

[9] S. Soro and W. Heinzelman. A Survey of Visual Sensor Networks. *Advances in Multimedia*, 2009:1–21, 2009.

[10] S. Velipasalar, J. Schlessman, C.-Y. Chen, W. Wolf, and J. P. Singh. SCCS: A Scalable Clustered Camera System for Multiple Object Tracking Communicating Via Message Passing Interface. In *Proc. IEEE International Conference on Multimedia and Expo*, pages 277–280, July 9–12, 2006.

[11] Y. Wang, M. Casares, and S. Velipasalar. Cooperative Object Tracking and Event Detection with Wireless Smart Cameras. In *Proc. Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance AVSS '09*, pages 394–399, Sept. 2–4, 2009.

[12] M. Welsh and G. Mainland. Programming Sensor Networks using Abstract Regions. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 3–3, Berkeley, CA, USA, 2004. USENIX Association.

[13] T. Winkler and B. Rinner. Pervasive Smart Camera Networks Exploiting Heterogeneous Wireless Channels. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1–4, Mar. 2009.

[14] W. You, H. Jiang, and Z.-N. Li. Real-time Multiple Object Tracking in Smart Environments. pages 818 –823, Feb. 2009.