Standard Integration of Sensing and Opportunistic Diffusion for Urban Monitoring in Vehicular Sensor Networks: the MobEyes Architecture

Paolo Bellavista, Eugenio Magistretti
Dip. Elettronica Informatica Sistemistica (DEIS)
University of Bologna
Viale Risorgimento, 2 – 40136 Bologna – Italy
Phone: +39-051-2093001; Fax: +39-051-2093073
{pbellavista, emagistretti}@deis.unibo.it

Uichin Lee, Mario Gerla
Computer Science Department
UCLA
1010 Westwood Plaza - 90095 Los Angeles - USA
Phone: +1-310-825-4367; Fax: +1-310-825-2273
{uclee, gerla}@cs.ucla.edu

***Abstract*— The emerging industrial relevance of vehicular sensor networks pushes towards their adoption for large-scale applications, from traffic routing and relief to environmental monitoring and distributed surveillance. With homeland security issues in mind, we have developed MobEyes, a fully distributed opportunistic harvesting system for urban monitoring. In MobEyes, regular vehicles equipped with sensors collect and locally store monitoring data while moving on the streets. Sensors may generate a sheer data amount, especially in the case of audio/video recording, thus making traditional reporting unfeasible. MobEyes originally adopts the guidelines of locally generating summaries of sensed data and of taking advantage of vehicle mobility and opportunistic one-hop communications to pump summaries towards mobile collectors, with minimal overhead, reasonable completeness, and limited latency. To that purpose, it carefully considers standard specifications to portably integrate with heterogeneous sensors, in particular by exploiting the Java Media Framework to interwork with cameras, the JSR179 Location API to interface with heterogeneous localization systems, and the Java Communications API to access lower-layer environmental sensors.**

I. INTRODUCTION

Vehicular Sensor Networks (VSN) are emerging as a new network paradigm for effectively monitoring the physical world. In fact, vehicles, typically not affected by strict energy constraints, can be easily equipped with sensing devices (chemical spill detectors, cameras, …), powerful processing units, and wireless transmitters. That pushes towards the VSN adoption in different large-scale applications, from traffic routing/relief to environmental monitoring and distributed surveillance.

In particular with homeland security issues in mind, we have recently developed MobEyes, a fully distributed opportunistic harvesting system for urban monitoring data, specifically designed for post-facto crime scene investigation. In MobEyes, regular vehicles of common people are equipped with cameras (and possibly with additional sensors, e.g., to detect chemical attacks and pollution indicators). These vehicles collect and locally store urban monitoring information while regularly moving on the streets. On-board sensors may generate a sheer amount of data, especially in the case of recorded audio/video streams. Traditional sensor network approaches for data reporting, e.g., to police agents, are unfeasible in this scenario [1, 2]. MobEyes originally adopts the guidelines of locally generating summaries of sensed data and of taking advantage of vehicle mobility and opportunistic one-hop communications to pump these summaries towards police patrol cars, which may move during the harvesting process. The goals are i) to impose minimal communication overhead on the limited bandwidth available for car-to-car communications, ii) to achieve reasonable completeness of harvested summaries to cover the largest part of sensed data, and iii) to obtain that reasonable coverage with a limited latency, considered acceptable for the addressed application scenario.

To achieve these goals, we have designed and implemented MobEyes according to an architecture composed by two key modules: the MobEyes Diffusion/Harvesting Processor (MDHP) and the MobEyes Sensor Interface (MSI). On the one hand, MSI provides an abstraction layer to uniformly access sensed data independently of sensor implementations, thus leveraging the MobEyes adoption in industrial large-scale applications. On the other hand, MDHP implements our original opportunistic summary dissemination/harvesting protocols portably and effectively over different wireless communication technologies.

In particular, in this paper, we will report our experience in developing and deploying the Java-based MSI and MDHP modules, which strongly consider maximum portability and interoperability as their crucial implementation requirements to facilitate industrial adoption and rapid diffusion. By delving into finer details, MSI has adopted the choice of interfacing with the highly heterogeneous world of sensing devices with standard and state-of-the-art open specifications, such as the Java Media Framework for cameras, the JSR179 Location API for possibly heterogeneous positioning systems, and the Java Communications API for interfacing with lower-layer environmental sensors. In addition, MDHP uses the standard Java connectivity support (.net package for sensing nodes with Java 2 Standard Edition (J2SE) and the Connection framework

1-4244-0755-9/07/$20.00 ©2007 IEEE

for limited nodes with Java 2 Micro Edition – J2ME) to uniformly access the possibly heterogeneous wireless connectivity technologies available on different vehicles. The experience made in the MobEyes design and implementation points out the suitability of the Java environment, extended with recent standard specifications for specific goals such as JSR179, to obtain a rapid and portable prototyping of urban monitoring applications for vehicular networks.

The remainder of the paper is organized as follows. Section 2 clarifies background ideas and positions our novel approach with regard to the state-of-the-art in the field. Section 3 rapidly sketches the high-level architecture of MobEyes, while Section 4 and Section 5 delve into finer design details of MSI and MDHP, respectively. Some preliminary tests have been accomplished to validate MobEyes interfacing with heterogeneous sensors; the corresponding experimental results are in Section 6. Conclusive remarks and directions of future work end the paper.

II. Background and Rapid Overview of the MobEyes Architecture

Surveillance of critical areas, tracking of suspect felons, and reconstruction of crime events are all compelling cases for urban monitoring. An expanding range of research projects is a clear proof of the growing interest in the field. For example, Intel Research IrisNet [3] addresses large-scale monitoring environments based on statically deployed PCs equipped with off-the-shelf cameras and microphones. MIT CarTel [4] permits to inject new queries on moving vehicles equipped with sensing devices, by exploiting wireless connectivity provided by open access points. Originally if compared with these projects, MobEyes focuses on a posteriori collection of information related to events potentially monitored by distributed sensing devices mounted on vehicles. This becomes the problem of searching in a massive, mobile, and completely decentralized storage of sensed data, by establishing a distributed index via completely decentralized cooperation.

To rapidly introduce MobEyes goals and solution guidelines, let us overview it while at work in a futuristic operating scenario, i.e., urban surveillance. Cars move on the streets collecting images through cameras installed on rooftops. In that context, MobEyes helps the police to build a distributed index upon the huge amount of information collected by regular cars. If nodes diffused their whole sensed data, the network would collapse given the large size of collected multimedia contents. MobEyes proposes that vehicles locally process the sensed multimedia streams to extract some summarizing features, such as license plates of encountered cars. Only this smaller summarizing information will be diffused in the VSN. In particular, MobEyes uses opportunistic exchange protocols, based only on 1-hop communications, to spread summaries among cars. Police agents are the only entities that can harvest summaries: they maintain a local table representing the index (not necessarily complete, since its contents depend on the effectiveness and latency of harvesting protocols, as detailed in Section 4) of the sensed data currently stored on remote vehicles.

To support these tasks we have designed and implemented MobEyes according to the component-based architecture in Figure 1. The two key modules are MSI, which supports a portable and transparent access to heterogeneous sensing devices, and MDHP, which implements opportunistic summary diffusion/harvesting protocols. Both these facilities will be extensively described in the following sections. The third MobEyes component, less specific for the VSN research area, is MDP, which periodically collects sensed data from MSI, and extracts useful features, such as license plate numbers of cars in sensed video streams, through application filters. We rapidly observe that signal processing algorithms to support MobEyes target applications (e.g., accurate license plate recognition) have been recently developed and are out of the specific scope of this paper [5, 6]. To create summaries, extracted features are then combined with relevant data read from other sensors and physically situated with corresponding timestamp and geographic location. Finally, MDP stores raw sensed data and summaries in the Raw Data Storage and Summary Database, respectively, via standard functions for persistency and database management.

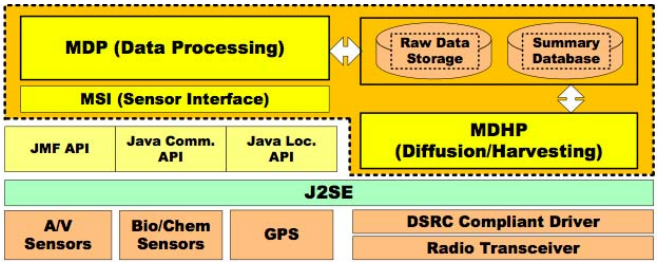

Fig. 1. MobEyes Architecture

III. MobEyes Sensor Interface (MSI)

MSI aims to facilitate the access to possibly heterogeneous sensor devices, by providing a high-level interface that exposes generic functions. In this way, MobEyes guarantees access transparency and high adaptability to changes in the devices (and in their driver implementations) available in the deployment environment and possibly discovered at runtime. For instance, if the sequence of operations to access a camera sensor changes, developers working on top of MobEyes are completely hidden from the modification. Let us rapidly observe that this dynamicity is obtained by considering only the limited and invariant set of operations needed to MobEyes. In other words, MSI has been specifically designed for MobEyes and, thus, does not permit general-purpose control operations and parameter settings on sensor devices.

MSI is built on top of the standard Java Virtual Machine. This choice grants wide portability to our implementation. At the same time, that design decision has facilitated our

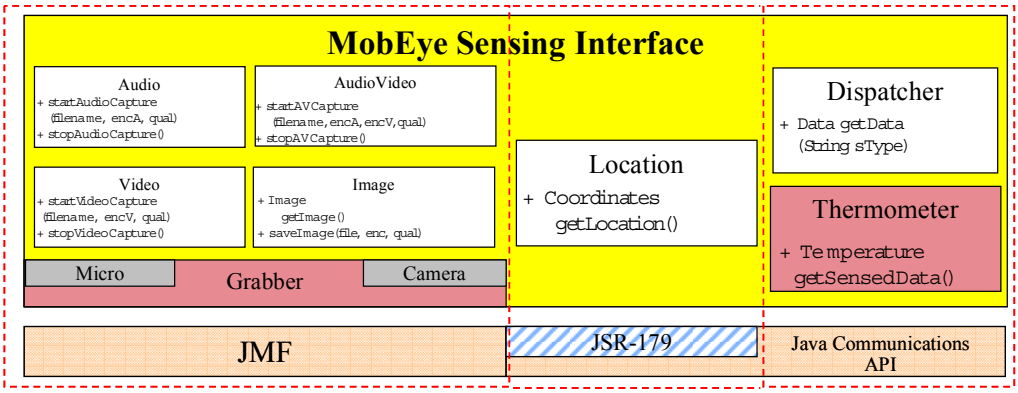

Fig. 2. The modular architecture of MobEyes MSI

development and deployment work, by allowing us to adopt several standard Java API that provide useful contributions to supporting sensor communications, control, and management. Some API, as detailed in the following, are the result of the open Sun standardization process (Java Community Process - JCP), which warrants a widespread support for heterogeneous platforms by involving all stakeholders, from developers to companies, in the definition of novel Java specifications and extensions to increase consensus on crucial design decisions [7].

MobEyes deployment scenarios call for the support of a number of different sensing devices, depending on the kind of data that police patrol agents are going to retrieve from regular cars, e.g., audio/video streams or images of the streets, temperature, weather, and road conditions, all to be tagged with location information. We identified three primary classes of sensors, corresponding to three different standard Java API. MSI exploits the Java Media Framework (JMF) API to access the first class of sensed data, which is generated by multimedia devices such as cameras and microphones. JMF provides a widespread set of functions to perform acquisition, control, and management operations on multimedia sensors (e.g., to capture images or video streams with digital web-cams, or to command/transfer the recording of audio streams with microphones) [8].

The second class includes all sensors (usually monitoring lower-layer environmental information if compared with audio/video sensors) that can be connected through an RS-232 serial interface. RS-232 can either provide access to an embedded board where sensors report analog inputs or directly receive the output signal of a single sensor. In both cases, the values made available on the serial interface are retrieved by using the Java Communications API [9]. This standard API permits to operate with serial/parallel communication ports, by hiding the details of low-level platform-dependent drivers. The Java Communications API supports both synchronous and asynchronous (event-driven) programming models. In particular, it is possible to automatically raise/receive notifications every time a signal overcomes a specified threshold. For any different sensor type (temperature sensors, carbon-oxide detectors, …) of interest for MobEyes, MSI currently implements an ad-hoc module for specialized serial data parsing. We are working on generalizing the parsing process so to provide a single parser module, possibly instructed by different XML-based descriptions of the data format provided by specific types of sensors. Mainly due to the proof-of-concept purpose of our current MobEyes prototype and to the non-negligible cost of pollution sensors, at the moment MSI includes only two specific parsing modules for temperature and hygrometer sensors.

Since in MobEyes any monitored data are useful only if tagged with space/time coordinates of the corresponding sensing location, the third crucial class of sensors includes positioning systems, i.e., "sensors" that can provide localization data. MSI can obtain geographic location of sensors by querying the positioning system hosted on board of the car. To interface with heterogeneous positioning solutions (satellite-based, such as GPS, but also signal-strength based, such as Ekahau [10]) in a standard way, MSI employs the Java Location API (JSR 179) [11]. For instance, MSI invokes JSR 179 functions to select the positioning technique to use, by simply specifying the desired location accuracy and/or response time. Other JSR 179 functions are used to get position updates either synchronously or through an event-driven interface. The latter permits either to specify periodic updating intervals or to be notified when located in proximity of a target. The usage of these standard API allows MSI to be independent of the implementation of the specific localization system available in the deployment environment.

In the following, the section specifically focuses on the main design choices behind our portable MSI realization, structured around the three previously sketched classes of sensors. Figure 2 shows the overall MSI architecture with, from left to right, the three subsystems supporting the three sensor classes.

A. Standard Interfacing with Audio/ Video Sensors

The leftmost part of Figure 1 shows the audio/video sensor interfacing subsystem. This includes four components supporting access to audio, video, synchronized audio/video, and image data. A grabber module is responsible of the interaction with JMF functions. The grabber facilitates design maintenance by decoupling high-level MSI components from

access procedures, which may be specific of each device. In short, the grabber is in charge of obtaining a JMF Player/Processor from an abstract input device (Microphone or Camera), and of connecting its output to a destination file where the sensed data will be stored.

MSI hides the details of actual data access operations, by exposing high-level methods to grab the currently sensed image and to save audio, video, and audio/video streams. For image grabbing, MSI either returns an Image object or a file of the taken picture. With regards to data streams, MSI permits to command recording start/stop; the stream is initially stored in an uncompressed format (RGB video and LINEAR audio) for efficiency reasons as motivated in the experimental result section, and encoded only offline.

MSI provides two main parameters to simply control sensing processes: format and quality, which affect occupied memory, processing time, and reproduction accuracy. The suitable parameter choice depends on application-level requirements. For example, MDP may require high quality images for post-processing to extract license plate numbers. In the case of video streams finally watched by human operators, top quality is usually not needed and it is possible to configure MSI with less resource-consuming format/quality settings.

Through JMF, MSI can support many different formats, including PCM, MPEG Layer 2 and GSM for audio streams, MPEG-1, MJPEG and H.263 for video streams. The choice of MSI quality value (with a coarse granularity from 1 to 3) directly influences the adopted encoding parameters (see Section 5).

B. *Standard Interfacing with Temperature Sensors*

The rightmost part of Figure 1 shows the sensor interfacing subsystem. Its flexible and modular architecture is based on the Sensor abstraction representing the actual device. Sensors export a generic method, *getSensedData()*, which returns an object of abstract type Data. The MSI Dispatcher rules the interaction between upper layers and Sensors. The MSI Dispatcher API includes the method *Data getData(String sType)* that, based on the requested data type, returns a Data object with current reading. In case the sensor does not properly work, the method raises an exception.

MSI directly builds on the low-level Java Communications API to access and collect sensed data. The Java Communications API permit both to synchronously read data from Sensors and to register listeners to be invoked every time new data are available on the communication port (serial and parallel ports). Both modes are fully supported and integrated in MSI.

Since MSI will likely need to support a growing number of sensors, extensibility is a crucial aspect for its design. To this purpose, the Dispatcher manages only Sensor and Data interfaces, without the need of any modification if a new sensor type is added. In that case, developers willing to extend the MSI prototype should only implement the new device class as a subclass of Sensor. The device-specific Sensor subclass is in charge of actually reading, parsing, and verifying the raw data present on the serial port.

C. *Standard Interfacing with Positioning Systems*

MSI permits to easily include in the set of sensed data also the geographic coordinates of sensors, in an open and standard way. Our Location module provides a simplified view of JSR 179 functions to MobEyes developers, by aggregating and composing API of the standard Java specification. In particular, the Location module can synchronously return the current <latitude, longitude, and (optionally) altitude> car coordinates. Similarly to the generic sensor case, the function either creates an object encapsulating the coordinates or raises an exception, e.g., in the case GPS is the only available positioning technique and cannot determine the position because the car is indoor in an underground car park.

To the best of our knowledge, no free implementation of JSR 179 was available for J2SE at the time of writing. Thus, two different design options were possible: either implementing the JSR 179 specifications, or interfacing the GPS as if it was a common sensing device, i.e., directly through the Java Communications API. Given the relevance of opening MSI via the extensive adoption of standard specifications, we decided to develop our partial implementation of JSR 179. Our implementation of LocationProvider interfaces with GPS equipment via a serial port via the Java Communications API. Currently, we are working on a portable extension of our LocationProvider to support also USB-based GPS devices, by exploiting the Java USB API (JSR 80) [12].

IV. MobEyes Diffusion/Harvesting Processor (MDHP)

After rapidly introducing the primary guidelines of the original MobEyes delay-tolerant protocols for summary diffusion/harvesting, this section focuses on the description of MDHP realization, by pointing out the motivations behind our architectural choices and their impact on the solution portability in industrial large-scale applications.

MobEyes aims at creating a distributed and partially replicated opportunistic index of the information collected by moving vehicles. MDHP is in charge of supporting fast and effective ways for mobile police agents to harvest summaries from mobile regular nodes. To that purpose, we have designed two original protocols: the first for regular nodes to periodically spread summaries through 1-hop diffusion; the second for agents to query opportunistically encountered cars to build an updated and partial distributed index of sensed data.

More specifically, a regular node periodically advertises newly generated summaries through 1-hop broadcasts. Neighbors populate a local table including copies of all the summaries received and not delivered to agents yet. Then, neighbors can either contribute to boost the diffusion by further relaying or refrain. In the first case, the diffusion speed of the summaries results dramatically improved; however, this comes at the expense of higher communication overhead and larger

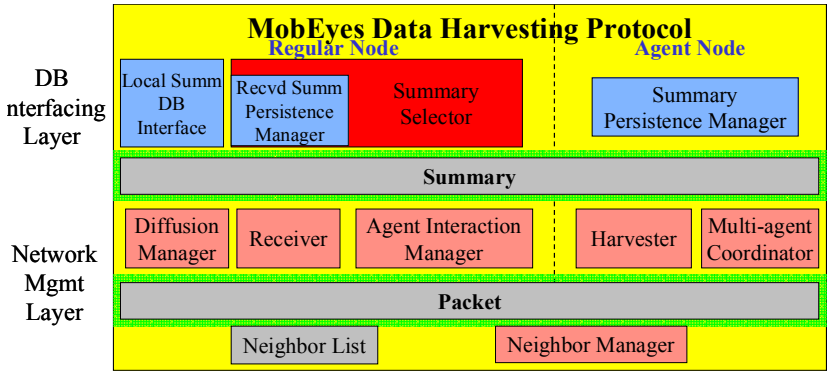

Fig. 3. The MDHP modular architecture for regular and agent nodes

memory required for local summary storage. Depending on the requirements of supported applications, MDHP permits two operating modes: a basic one (only the source advertises its packets to 1-hop neighbors) and a passive k-hop (any packet travels up to k hops as it is forwarded by j-hop neighbors, with $j<k$). Strategies to further expedite summary diffusion can be devised by allowing nodes to continuously advertise all the packets within their local tables, also if generated by other nodes.

While regular nodes move and diffuse their summaries, police agents roam with the goal of building a distributed index by harvesting as many summaries as possible. When required, e.g., after a crime event or periodically, agents may query neighbor regular nodes to obtain summaries they have not collected yet. To this end, agents advertise Bloom filters hashing the already harvested summaries [13]. Every regular node tests its local table entries against the received Bloom filter and replies by delivering only non-matching items. MDHP takes into consideration situations where multiple regular nodes are simultaneously present: in that case, MDHP adopts heuristic-based strategies to properly schedule node communications in order to avoid redundant summary deliveries [14].

Several agents will likely scour the urban area concurrently in real deployment scenarios. MDHP supports collaborative strategies to build a distributed and partially replicated index. Simple strategies have been devised so far to combine the information carried by different agents: police agents can exchange their Bloom filters through multi-hop paths as soon as they have collected a specified number of new summaries. More effective solutions based on controlling agent trajectories are under investigation to further limit communication overhead and latency [15].

A. MDHP Architecture and Portability

MDHP manages communications for regular nodes as well as for police agents. It is in charge of extracting/storing summaries from/to the local database and of implementing opportunistic protocols for summary diffusion/harvesting. Figure 3 represents both regular node and police agent MDHP components; obviously, only the suitable ones will be installed on a single vehicle depending on its type.

MDHP functions can be split into two different layers. The upper layer (DB Interfacing Layer) interworks with the summary database that maintains summaries either locally generated or obtained from neighbors and not delivered to an agent yet. The lower one (Network Management Layer) consists of the components that actually implement communication protocol operations. While the DB Interfacing layer deals with instances of the Java Summary class, the Network Management layer marshals/unmarshals summaries into packets. Any summary includes a license plate number (6 bytes), additional sensed data (10 bytes, currently a 3-byte temperature/hygrometer info and a 7-byte placeholder), timestamp (2 bytes), and vehicle location (8 bytes). Thus, each 1500-byte packet can pack up to 58 summaries, without exploiting aggregation or size-optimizing encoding techniques, which are currently under investigation.

Periodically, the Diffusion Manager at regular nodes advertises recently generated summaries (one packet with the 58 last summaries provided by the Local Summary DB Interface). In the current MobEyes implementation, the diffusion period is set to 5 seconds and new summaries are expected to be generated with a maximum rate of 0.4 Hz (so, each summary is advertised at least 29 times). These parameters, e.g., the diffusion period, are tuned depending on the MobEyes-supported application and deployment environment. The cited values permit to accurately track vehicle positions in common urban scenarios [15]. As future work, we are considering: i) to adapt the diffusion period to the changing rate of neighbor set (detected by Neighbor Manager); and ii) to combine summaries generated in different epochs in the same packet. The guideline is to maximize the usefulness of packets by advertising them to new neighbors expected not to have already collected the included summaries.

Any time a regular node receives new summaries, the Received Summary Persistence Manager updates the local database. Summaries are maintained until a police agent query is received: in that case, the Summary Selector component performs Bloom filter matching [13], prepares the set of summaries to send to the agent, ordered from the least to the most recent ones (in fact, due to random node trajectories, oldest summaries are likely to be the rarest in the neighborhood and consequently the higher priority ones), and then removes those summaries from the local database.

Actual communications between agents and regular cars are carried by the Harvester and the Agent Interaction Manager.

The Harvester coordinates neighbor communications by exploiting unicast messages for queries. The Agent Interaction Manager, instead, handles summary delivery on regular nodes. In the current prototype, as soon as an agent encounters other agents (at 1-hop distance), the Multi-Agent Coordinator exchanges the list of harvested summaries. We are now extending the MobEyes architecture to support multi-hop inter-agent communications via IETF reactive routing protocols such as AODV [16, 17].

About low-layer communication support, we built MDHP on the standard .net package for sensing nodes equipped with J2SE. Limited nodes with J2ME will benefit from a MobEyes prototype version with a different implementation of the Packet component of the Network Management Layer, based on the standard Connection framework included in the official Sun virtual machine.

V. Experimental Results

We developed current MSI and MDHP implementations on top of J2SEv1.5. We adopted: 1) the official Sun release of JMF 2.1.1 with Windows Performance Package (including enhanced audio/video decoders/encoders for Microsoft platforms); 2) our own implementation of JSR 179; 3) the official Sun release of Java Communications API v2.0.

To verify the feasibility of our approach, we tested MobEyes components in real-world scenarios. Due to the lack of space, here we present some selected results. These refer to the performance of the MSI audio/video capture functionality. Tests were performed by capturing streams at a trafficked intersection near the UCLA campus and were run on Dell Latitude D610 laptops, equipped with PentiumM2GHz, 512MB RAM, and Logitech Quickcam Chats. We aimed at evaluating three parameters: the *size* of generated files of sensed data, mainly dependent on stream length and encoding type; the *overhead time* needed to capture the stream, i.e., the gross amount of time needed to start media processor and to close processor and output file; and the *conversion time* to encode the stream.

Table I, II, and III show the results obtained while capturing audio and audio/video streams, either in raw or encoded formats. For RGB video, the frame rate was set to 4 fps and resolution to 320x240 (we are currently experimenting with higher-resolution cameras allowing 640x480 image capturing, as suggested in [5]); for LINEAR audio, sample rate was set to 44100, 16 bits per sample. Table I shows the average size of the files generated during our tests, for different recording time lengths. Both compression methods (audio/video MJPG, and audio GSM) achieve a significant file size reduction. MJPG permits to tune a "quality" parameter, influencing the conversion time, as well as the output file size. Table I shows only the results with a medium quality value: this permits to reduce the file size of about 5 times with regard to the raw version. Similar results are obtained in the case of streams including only the video track. Table II shows the overhead time needed to start the JMF processor capturing the stream and to terminate it. Results prove that audio/video capturing overhead is significantly greater than the one for a stream with only the audio track. Finally, Table III reports how long it takes to MSI to convert audio/video streams to the MJPG encoded format, depending on the chosen quality factor. Lower quality values (MobEyes quality=1, equivalent to MJPG quality=10%) impose a stable conversion time, largely compatible with typical MobEyes application requirements.

TABLE I
GENERATED FILE SIZE (IN [KB])

Rec-Time	A/V RGB	A/V MJPG (Medium)	Audio LINEAR	Audio GSM
5s	4907	934	439	9
20s	16280	3295	1747	33
60s	46518	8749	5180	95

TABLE II
OVERHEAD TIME (IN [MS])

RecTime	A/V RGB	Audio LINEAR
5s	4790	206
20s	4493	401
60s	7780	270

TABLE III
CONVERSION TIME (IN [MS])

RecTime	A/V MJPG (Low)	A/V MJPG (Medium)	A/V MJPG (High)
5s	802	1432	2099
20s	1901	2198	11354
60s	6726	6523	37429

VI. Conclusions and Future Work

Urban monitoring is becoming a research field of growing interest. With the goal of supporting a posteriori monitoring investigations, we proposed MobEyes to address searching in a massive, mobile, and decentralized storage of sensed data. In this paper, we discussed and evaluated MSI and MDHP, two key components of the MobEyes architecture. Our MSI/MDHP design and implementation demonstrate that the integration with Java-based standard solutions (JMF, JSR 179, Communications API, .net package) can allow to obtain feasible performance results with a high degree of openness, thus facilitating the adoption in large-scale existing industrial deployment environments.

The encouraging experimental results already obtained are stimulating further research work. In addition to some directions already sketched in the paper, we are currently investigating how the protocol for summary exchange among agents at multi-hop distance works when implemented upon ad-hoc routing solutions.

ACKNOWLEDGMENT

The authors would like to thank Antonio Corradi and Chiara Chiappini, whose suggestions, help, and support were crucial for the realization of this work.

REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "A Survey on Sensor Networks", *IEEE Communications Magazine*, Vol. 40, No. 8, Aug. 2002.

[2] C. Intanagonwiwat, R. Govindan, D. Estrin, "Directed Diffusion: a Scalable and Robust Communication Paradigm for Sensor Networks", *ACM Mobicom Conf.*, 2000.

[3] P. Gibbons, B. Karp, Y. Ke, S. Nath, S. Seshan, "IrisNet: an Architecture for a Worldwide Sensor Web", *IEEE Pervasive Computing*, Vol. 2, No. 4, Oct-Dec. 2003.

[4] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. K. Miu, E. Shih, H. Balakrishnan, S. Madden, "CarTel: a Distributed Mobile Sensor Computing System", *ACM SenSys Conf.*, 2006.

[5] S.-L. Chang, L.-S. Chen, Y.-C. Chung, and S.-W. Chen, "Automatic License Plate Recognition", *IEEE Transactions On Intelligent Transportation Systems*, Vol. 5, No. 1, Mar. 2004.

[6] UNO-2160 in Mobile License Plate Recognition System, http://www.advantech.com.tw/ia/newsletter/AutomationLink/January2005/Application_Story _ UNO-2160.pdf

[7] Java Community Process, http://jcp.org

[8] Java Media Framework, http://java.sun.com/prod-ucts/java-media/jmf/

[9] Java Communications API, http://java.sun.com/ products/javacomm/

[10] T. Manesis, N. Avouris "Survey of Position Location Techniques in Mobile Systems", *ACM Conf. Human Computer Interaction with Mobile Devices and Services*, 2005.

[11] Java Location API 179, http://jcp.org/en/jsr/de-tail?id=179

[12] Java USB API, http://jcp.org/en/jsr/detail?id=80

[13] L. Fan, P. Cao, J. Almeida, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocols," *ACM SIGCOMM Conf.*, 1998.

[14] U. Lee, E. Magistretti, B. Zhou, M. Gerla, P. Bellavista, A. Corradi, "MobEyes: Smart Mobs for Urban Monitoring with a Vehicular Sensor Network", *IEEE Wireless Communications*, Vol. 13, No. 5, Oct. 2006.

[15] U. Lee, E. Magistretti, B. Zhou, M. Gerla, P. Bellavista, A. Corradi, "MobEyes: Smart Mobs for Urban Monitoring with a Vehicular Sensor Network", *UCLA CSD Tech. Report 060015*, http://netlab.cs.ucla.edu/wiki/files/mobeyestr06.pdf

[16] E.M. Royer, C.-K. Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks," *IEEE Personal Communications*, Vol. 6, No. 2, April 1999.

[17] C.E. Perkins, E.M. Royer, "Ad Hoc On-Demand Distance Vector Routing", *IEEE WMCSA Conf.*, 1999.