Middleware for data sensing and processing in VANETs

Stefan Nour, Raluca Negru
Politehnica Bucarest
Bucarest, Romania

Fatos Xhafa
Technical University of Catalonia
Barcelona, Spain

Florin Pop, Ciprian Dobre, Valentin Cristea
Politehnica Bucarest
Bucarest, Romania

***Abstract* – In urban environments nowadays there are a multitude of mobile devices that have several ways of communicating with each other, which become more and more powerful and used in every day activity. The most important characteristic of these devices is that they can supply useful information for a detailed vision of the area that they cover, especially when they travel a large distance. By gathering all this particular data and then processing it, we can obtain a global image of different aspects like traffic flow, pollution, weather condition, and all this with the help of the mobile users. In this paper we present a middleware for sensing data in Vehicular Ad-hoc NETworks that uses the OSGi framework and tries to simplify the work made by application programmers by using the API prototype provided. This concludes in building, deploying and maintaining applications that will process the data acquired from sensors and deliver real-time information useful to the users.**

Keywords: VANET, OSGi, middleware, services

1. Introduction

Networks formed by daily used devices like smartphones, GPS and sensors can serve as instruments for observing the physical environment. These networks can be employed to detect traffic jams, to monitor the level of pollution in the air, or to check on the empty parking places in an area.

Static-sensor networks have been deployed in various areas of interest by different organizations. Data provided by these networks is collected and processed by a central point and the output isn't always accessible from outside the organization that owns the network. Although there are many application scenarios that can exploit the information received in this manner, they still cannot provide a so called ubiquitous sensing that permits user applications to easily access the data anytime and anywhere [1].

Thanks to recent advances in artificial intelligence and in the automotive industry, many cars are equipped with communication devices and are capable to interact with nearby vehicles and nearby fixed equipment. The network formed in this manner is called a Vehicular Ad-Hoc Network (VANET) and tends to operate without any infrastructure or legacy client and server communication. VANETs can collaborate with other mobile devices existing in urban environments, whose scope and potential of usage increase every day, or with public sensor networks deployed by municipalities, and individual sensors incorporated in buildings and roads. All of this can contribute to a large system that has the capability to collect the information and make it accessible to other parties that can implement an application for everyone to benefit from. The drawback is that wireless networking is not very reliable because of electromagnetic wave propagation, interference, and battery exhaustion, making these mobile devices come and go without warning or visible reason.

Managing and exploiting the information gathered from this type of network is not an easy task given the heterogeneity of the component devices, their intermittent connectivity, limited autonomy and storing capacity. In order to make application development easier we have proposed a middleware based on OSGi framework that will be described in the following sections. The existence of a middleware will support the development, deployment and execution of distributed applications in the heterogeneous and dynamic mobile environment dealing with issues like scalability, adaptability and configuration easiness. It adds a level of transparency regarding the access of network resources, fault tolerance, communication, data transfer and eliminates developer and language dependency. The middleware design, based on the OSGi platform, permits software decomposition into flexible units and makes it easy to replace or update certain modules without having to reinstall the whole application [5].

2. Middleware proposal

2.1 Motivation for developing a middleware

In an environment where cars are omnipresent and a significant number of them are able to collect information regarding the area in which they are moving at a specific time, we proposed a solution where we can use that information to understand and help protect the environment. And to do so, we came up with a solution that can decongest the traffic and help reduce the pollution in specific areas, by rerouting the participants on the best way possible. This can be done by using devices with communication capabilities that can transmit the information collected to other devices, whether there are fixed or mobile. But since there are so many types of devices, and they can exchange different data sources in different formats, there has to be an architecture that unifies and simplifies them in a transparent way.

Below we will explain how we can achieve fast processing and how our middleware can deliver information in real time, after which we will explain why we chose this type of layering and how this middleware is best suited for entities in VANET.

978-0-7695-4456-4/11 $26.00 © 2011 IEEE
DOI 10.1109/EIDWT.2011.33

IEEE computer society

2.2 Middleware design

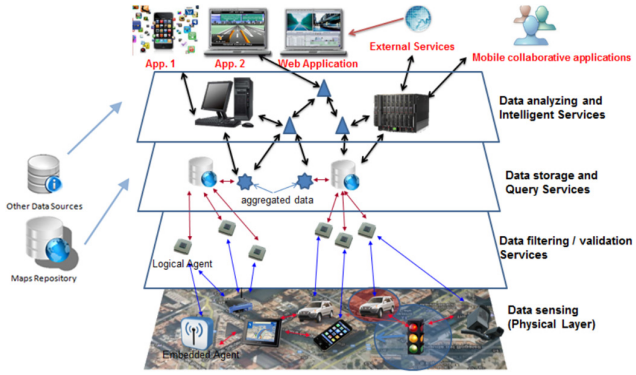

Figure 1. Middleware architecture

The middleware presented here tries to handle the problems that occur in a loosely-coupled system and defines a set of software components and services that support development and deployment of applications. This middleware is designed to resolve most interoperability aspects that may occur, like data format (XML), semantics, interface definition language and transport mechanism. This can be done very logically if we try to separate the middleware into several layers, each of them dealing with their own operations and services, having in this way the possibility to modify or implement only one part of the middleware, without affecting the other ones so much.

Also, the middleware is capable of interacting with other external services that can provide information usable for obtaining an accurate representation of the data received from the mobile sensors. For example, the representation of the road maps is an external service and can change or update regardless of the structure of the middleware.

Below we will describe in detail how every layer works, but first we want to point out the benefits of using these specific layers. We will start with the sensing layer that deals with all kind of data that a sensor might collect or transmit through the network. At this layer, we resolve interoperability with other devices and we transform the received information into usable internal data. This layer signals the filtering layer when a new message has arrived and passes it through for further interpretation. In this step, the filtering is made based on the relevance of the information received, and if necessary, a response is sent back to the source for retransmitting a lost packet or acknowledging the request that just has been made. The basic idea behind this layer is keeping track of the packets that are sent or received and to integrate a scheduler for organizing the requests of a specific type of messages. After filtering is done, the information is transmitted to either the storage layer which stores the information on multiple repositories, takes care of fault tolerance and implements a fast query system designed to ease the access to data, or either to the data analyzing layer for interpreting the data, correlate the results and provide a better and complete response for the request that is being processed. The intelligent services layer also provides scheduling for all the application that run on top of this middleware, sorts and tries to analyze if the information that was given is useful or not and handles the communication with other data repositories.

2.3 Physical infrastructure

The participating entities are heterogeneous (different computing power, storing space, operating system), characterized by high mobility and unstable connectivity and this is why it is necessary the existence of a middleware for solving eventual hardware and software incompatibilities and for making possible the interactions between them. This middleware will provide a set of services that will allow the access to information across the network through software modules.

- **GPS Device + Communication** that are the core elements for obtaining accurate information from traffic and for interacting with people.
- **Semaphores** that can change color based on information received from logical agents.
- **Fixed embedded agents** with computing power and connected through a wired network for receiving or transmitting information of any kind to cars, sensors or other agents that are positioned at approximately 250m from the intersection (one fixed agent on every road forming the intersection).
- **Logical agents** that can handle multiple embedded agents and coordinate them.
- **Fixed sensors** with communication capabilities for measuring the pollution levels and traffic congestion that are positioned in a permanent physical place.
- **Mobile sensors** on buses, emergency cars, cabs for determining pollution levels or positioned anywhere for advertizing information of any kind.
- **Fixed video cameras** in intersections for recognizing registration plates or making sure the computer predictions are accurate and according to reality.

Figure 2. Physical Layer Components

- **Mobile video cameras** attached to mobile entities that can receive tasks about following a certain car with a specific plate number or can send information to the fixed agents if some abnormal activity appeared.
- **Mobile Smartphones** that will receive information if they reach a specific area or communicate with other devices in range.
- **Central units** that collect/store information and provide global viewing of the system for application to use them.

2.4 Data sensing layer

When we think of the basic layer of communication for mobile devices we need to understand that there are multiple devices nowadays, with different need and capabilities, but they all need to transmit and receive the same type of data(we will call it "messages" from now on).

Taking into consideration what primary entities participate in this middleware, we can assume that there are only two kind of basic messages:

- Static: Messages coming from fixed sensors, fixed agents, semaphores and other entities that we can rely on and know for sure where they are found.
- Dynamic: Messages that come from mobile units that may have positioning coordinates and are loosely coupled with the infrastructure.

In the middleware that we present, we have to implement the possibility of receiving and transmitting from every type of devices (wired, wireless, Bluetooth, 2G/3G networks, etc.) and we need to have the possibility of switching from one module to another without interfering with our transmission. On this specific layer, all the necessary work that has to be done is reporting if a new message has arrived, delivering a certain message and, if possible, auto-diagnose itself to see if it works properly. Because the wired sensors or the ones that we know that are in a fixed position can have their unique ID, we will focus our attention on the dynamic ones that we don't know anything about, but they transmit us useful data that we have to take into consideration. All this has to be done respecting interoperability of the devices and protocols used nowadays.

The dynamic sensors will be able to transmit and receive information not only to/from the physical agents (static) but also to/from other dynamical entities (in our case, other cars). The only condition is to use a standard format of messages, and that is done by applications that implement the interface that we provide. The emergencies messages will be sent as fast as possible and broadcasted through every type of communication.

To support as many connections as possible, we will have physical agents in every intersection so that each one of them handles a limited number of connections. Or, if the technology permits, to have internet connection on every device and send the information direct to the server.

The basic functions that we implemented in this layer are used for receiving a message and we have an event handler that listens on all the ports for incoming data. The really difficult part is after you get the message, you have to understand it, and for this we have a class that recognizes patterns and translates from several types of format to the one used internally. For this to work, we have to assume that the application running on the device implements our provided communication interface. Another service that this layer will provide is the transmission of messages to other entities, so we take the internal message and parse it to the specifications required by the receiving party.

2.5 Data gathering, filtering and validation services layer

All the data collected from the sensors is passed through a filtering layer for classifying and interpretation purposes. This layer will keep track of the messages sent to a specific entity and the response received, will analyze the message content and establish if it concerns this specific entity or if it should be discarded.

So, to be more precise, we will group most commonly used messages in this architecture into several categories:

- Emergencies messages, that will concern everybody and will be stored and distributed as well for it to propagate as quick as possible. We will keep a hash map of all the emergencies messages received (and sent) to compare them with other messages and see if there are duplicates.
- Messages that concern only a specific number of participants like: only the cars that are going in the same direction as the source of the message, so if there is another car that receives the message, but is going in the opposite direction, then it will discarded it.
- Messages regarding a fixed node, with a specific ID.
- Requests that are already in the queue of the device and are waiting for an answer: if the filtering layer notices the same request made several times, it tries to add in the queue of delivery the new devices that made that request and fulfill the answer to all of them once it is ready.
- Advertisements that act like broadcasted messages, but with the possibility of filtering them.

This layer will also be able to have programmable filters set by the user, so that only a specific kind of messages should be stored on the device.

For the system to work as expected, information from users must be kept anonymously, so we cannot be sure where the information came from and where to send a response to a request. To suppress this issue, we came up with a system of local ID that is good only for the specific area in which the agent is placed. So when a certain car request something from

an agent, it chooses a random ID and sends a message with the request until it gets a confirmation that the agent has been notified. After it crosses the intersection, the car also sends the information about the traffic in the intersection that it just passed to the agent that is in charge of that section of the road with the same ID so that the agents in the specified intersection know in what direction that specific car went. Of course this will be used only if a car doesn't want her ID to be public and doesn't want the system to track her movement and present at the request of the user a daily status of his route.

So, in the end, the filtering layer acts like a router's Access Control List for the packets that come to the device. And for this layer to work properly, we implemented several functions that interact with the sensing layer, like sending a message and receiving a message. Sending a message can be triggered if a previous request has been resolved or if we have to transmit an acknowledgement/missing packet message. All the requests are made using a queue based on FIFO. There will be another list that remembers all the received and transmitted data, so that if we didn't get an ACK from the entity that we sent the message, we should keep trying until we make sure it is received. There is also a cleanup function that triggers at specific intervals of time and analyzes the queue for messages that have expired due to any reason (no ACK received, the request blocks the server and is marked as failed). There is also a function that lets the programmer describe what filter to apply to that specific device.

2.6 Data storage layer

The most important aspects we took into consideration when designing this layer were minimizing the time necessary to retrieve data to the upper level and the accuracy of information. The Data Filtering layer provides the information that has to be stored in order to serve upper layers requests. Whenever data is pushed to this layer, it is compared to existing saved information and only updates are stored. Data destined for further processing like statistics and measurements of the sensors are kept until their passed to an agent and then discarded. When an entity issues a query, the physical agent closest to it will respond with the information needed, either from its own database or making himself a query to the superior agent. Even if there will be a second query made by an agent to another agent, the overall time of response will be merely the same because of the wired connection between them and faster processing time of the master agent. To summarize, there won't be a significant delay in time for the extra query because we want many applications developed upon this middleware to have real time data delivery. A part of data exchanged in this system will also be stored on the communication devices for a short period of time until it is passed to agents. This type of information is destined for immediate use like urgent messages that have to be passed from one entity to another.

We will present below several types of databases used in our system, for understanding a little more about each entity and in what form data is stored.

The car database consists of speed fluctuations from the last sent message. A database entry has broadly the next fields: datetime, coordinates, direction, speed and isAnAccident field. Of course, depending on the sensors that a car might have, it could also hold information about video surveillance, weather conditions and pollution.

A road assistant database consists of a temporary table for determining the direction of each car containing the records received from that car plus a field for knowing the direction to where it's heading. After it receives the direction from one of the other agents, he adds the respective information accordingly and then deletes it. He keeps a database of average speed, time and jam distance based on the last N records (decided by running the tests) that are not expired (another constant set at the runtime).

The central processing unit's database consists of tables defining granularity of each predefined time period. Each record will contain information about speed, jam distance and environment conditions. Once a predefined time period is elapsed, the records will be archived (but only if there are new ones, otherwise it will have an id of a matching record found before).

The main functions provided by this layer are for storing and retrieving information. Retrieve functions are called whenever a client application needs information that is stored in the system databases. If the query is unsuccessful, then an error is triggered and the data analyzing layer makes another query to an agent that has knowledge of that information. The store functions are triggered when lower layers decide that the information they currently possess needs to be stored in a safe place for further inquiries.

2.7 Data analyzing and Intelligent Services

The last layer is specialized in processing the data received from the network and in obtaining relevant information for the applications. The information will most likely refer to geographical positioning and that information has to be correlated with a representation of a map. This map is often a third party application and we need to integrate it with our information for a better interpretation of the data.

Another feature that will help the application programmers will be the tracking of a specific entity (at first it will be only for cars) based on video processing and license plate recognition. In correlation with this one, the middleware will provide live video streaming from the fixed cameras and, if the bandwidth permits, from the mobile cameras as well.

A very useful and important characteristic for this system is the ability to provide real time information about what happens in the field, based on the data sent by sensors. For this to happen, the layer analyzes the data and each time a significant change is made in the system, it triggers an event that sends the new information to a superior entity for everyone to have an updated picture of the part that this entity covers.

This specific layer was designed to keep things organized in the system so we won't have to worry about cleaning data

that is no longer used, so we tried to aggregate them based on time passed since the event and making special cases every time something unanticipated occurs. And for this, we will introduce another subroutine that this middleware tries to implement: an algorithm that can predict what will happen in a short period of time based on previous scenarios and current situation. This algorithm will calculate the best route from point A to point B trying to anticipate what will happen between and keeping in mind that from point A you start from a given time. There will also be a function that triggers an alert if something goes wrong or if the intervention of the user is necessary.

3. Technologies and implementation perspective

Java ME

Java Platform, Micro Edition (Java ME) provides a robust, flexible environment for applications running on mobile and other embedded devices with intermittent network connectivity: mobile phones, personal digital assistants (PDAs). Java ME includes flexible user interfaces, built-in network protocols, and support for networked and offline applications that can be downloaded dynamically.

Configurations (Connected Device Configurations, Connected Limited Device Configurations) and profiles (Mobile Information Device Profiles) are the two main elements that comprise Java ME platform. Configurations are specifications that set out how the virtual machine and a base set of APIs can be used with a certain class of device. A configuration deals with a specific device. A profile implementation consists of a set of Java class libraries that provide this application-level interface.

Although applications based on Java ME are portable across many devices, developers have to make sure their programs sizes respect the memory restrictions of some devices. One obstacle to the popularity of Java ME is the fact that so far it has been centered on the MIDP (mobile information device profile), a proprietary standard, but developers will eventually realize the benefits and build upon it. Amongst J2ME disadvantages, the most important is the lack of network security infrastructure in current implementations.

JXTA

The JXTA (juxtapose) supports various aspects of P2P networking, including the discovery of other computers and services and the ability to exchange information or invoke services. The JXTA specifications are simply communications protocols; they do not mandate particular hardware or software. The protocols are being implemented in many different languages. Two complete implementations are available, one written in the Java programming language, the other in C. JXTA peers communicate by exchanging XML messages. Messages called *advertisements* announce the availability of a peer.

JXTA for J2ME technology, or JXME, brings JXTA functionality to MIDP devices. Considering the fact that JXME is built for P2P architecture, it is not a good choice for VANETS where the mobile devices forming the network don't have much storing capacity and rely on the agents to supply and manage the required information.

.Net Compact Framework

Designed for mobile computing, .Net CF is a lightweight version of Microsoft's .Net Framework. .Net CF contains a subset of standard .Net API libraries necessary for mobile application development. The .Net CF development tool, VS.Net, currently supports two major languages: C# and VB.Net. It runs only on Windows CE/Pocket PC-powered high-end PDAs.

A major benefit of .Net is its ability to support multiple programming languages, ensuring the reuse of existing libraries. The fact that it supports only one OS platform can be a disadvantage for many mobile developers. Compared to other technologies .Net CF doesn't provide application life cycle management (download, installation, execution, and removal) or a service gateway.

Microsoft Robotics Studio

Concurrency and Coordination Runtime (CCR) is a Dynamically Linked Library (DLL), based on .NET Framework distributed with Microsoft Robotics Developer Studio (MSRDS) [8]. It is not limited in modeling robotic behavior and is also suited for developing service-oriented applications, composed of loosely coupled software modules interacting through messages, which have to deal with asynchronous operations, concurrency, partial failure and that exploit parallel hardware. CCR supplies the infrastructure that enables multiple tasks to execute concurrently on a single computer and can provide real time performance. It has also proved to be easily integrated with existing code.

What has proved to be very useful for VANETs is the fact that using CCR one thread can be used for normal messages and the second one is reserved for high priority messages, like accidents warnings. This library also helps with processing the asynchronous inputs received from multiple sensors, cameras, agents. In order to apply this technology to vehicular networks, an embedded PC, a laptop, or a PDA running Windows has to be mounted on each device enabling it to run MRDS.

OSGi

The OSGi specification defines a standardized component oriented platform for building Service Oriented Java applications. Modularity is the basic concept behind the OSGi technology, allowing applications to be constructed from small, reusable and collaborative components. OSGi provides a service-oriented architecture that enables these components to dynamically discover each other for collaboration. The

software modules composing the applications are called bundles, each bundle consisting in a single Java JAR file. A bundle must include a manifest file in which are specified bundle name, inter component dependencies, and other static information and each bundle can provide zero, one, or multiple services to other bundles. To use a bundle, it must be installed into the framework. An installed bundle is uniquely identifiable by either its bundle identifier (a number assigned dynamically by the framework when the bundle is installed) or by its location (which is an arbitrary character string used when installing the bundle). The location string is used to retrieve the bundle JAR file and is generally an URL [6].

The services are defined as interfaces in the Java language, and registered to the OSGi Service Registry that provides the necessary means for bundles to publish and retrieve services. Once a bundle has retrieved a service, it can invoke any method described in the interface of this service. The OSGi Alliance has developed many standard component interfaces for common functions like HTTP servers, configuration, logging, security, user administration, XML and many more.

Another important aspect of OSGi, that makes it suited for mobile ad-hoc networks, is the existence of the device management mechanism that allows for automated device detection as well as for automatic driver download and attachment of drivers to devices. The device manager paradigm permits the OSGi platform to run continuously without the need of restart, and possibility for dynamic driver update. OSGi simplifies the debugging and management of applications by providing many shells that allow to remotely inspecting the connections between bundles, disable bundles, remove bundles, check consistency, and provide updated versions.

OSGi technology adopters benefit from reduced development costs and shorter time to market release because OSGi technology provides for the integration of pre-built and pre-tested component subsystems, reducing maintenance costs and the need for proprietary software. OSGi has become widely adopted, many of the world's large companies belonging to different business areas like vehicle industry, telecommunications, electronics, are currently involved in OSGi projects.

We have chosen the Open Services Gateway Initiative (OSGi) framework for implementing the middleware because it is service oriented, independent of lower-level communication protocols, has open standards and can be adapted to a variety of networking technologies.

4. Application scenarios

In this section we propose three application scenarios that exploit the most of the middleware resources. The first application is designed to help reducing air pollution caused by intense traffic in crowded areas. This is done by analyzing the data regarding pollution levels received from sensors and creating a map of highly polluted areas. When the traffic participant using this application enters the destination he wants to reach, he will receive the route he has to follow in order to comply with the given constraints, maintaining the air pollution level under a certain value. The algorithm behind this application proposes more candidate routes, each one associated with a cost that is determined considering the traffic load, the fuel consumption and all additional taxes (for example drivers may be asked to pay a fee for crossing a polluted area). Another important element in computing the route is the estimation of fuel consumption when the car is idle, waiting in a traffic jam, thus a longer route can be more efficient compare to a shorter one that implies crossing crowded intersections and paying pollution fees.

Another application aims to offer an accurate, real time overview of the city traffic or a detailed statistic of the traffic flow in the past hour, day or week. Existing traffic monitoring systems offer only limited view of the city traffic and use data gathered by fixed sensors that are unable to offer accurate and relevant information. The user can access this application and find general information like which roads are usually the most crowded, what are the rush hours or how many free parking spaces are in a certain area, but also particular, user specific, details like how long it took to get from home to work for example. The particular information are rendered based on the identifier provided by the user and can prove helpful at computing the monthly fuel consumption for example. The traffic monitoring application is very important because it can serve as a starting point for other derivate applications, with a more specific purpose.

The third application presented here is the central idea on which this middleware was constructed and is derived from traffic monitoring: reducing time spent in traffic by rerouting through the best path available at that moment. Each car has to be equipped with a communication device which also has a GPS integrated. In our research we presumed that a car communicates with an agent through wireless (its area is 250 m) and that there are as many agents in the intersections as roads are. We supposed that on an average street the car will be in an agent's area for a distance of 500 m. The agents should be arranged so that a car receiving the route will have time to position accordingly at the next intersection. When a car is 500 m away from an intersection it sends a request to find a way to the destination. Another car can respond to a request only if it has received a route from an agent and its destination is the same as the one the car requested (this is only for receiving the route faster). The car will send the information about the traffic on the road that it has been on, after leaving the intersection. When a message with a request for obtaining the best route to a specific point is received, the GPS device tries to find the best possible route that he has. After he acknowledges that his route is out of date, he makes a request to the closest agent to get the latest information. The agent receives the message, interprets it, recognizes the request for a specific route and makes a query to see if he has all the information needed to take a good decision. If it has, then it retransmits the roads on which to go through and the time for crossing each one. In the same time, it sends a message to the central unit in which to inform that a car was sent on a specific route, for game theory purposes. In this way

can be prevented cases where all the cars are send on a certain route and that one suddenly becomes full. So, either way if the car will receive the message with the new route, it sends a message composed of the acknowledged traffic information from the last known good transmission. The agent compares this information with the one that it already has and tries to interpret it accordingly.

The purpose of this application is to find the shortest path from one point to another and to keep a fluent traffic all across the city. This will help not only the participants by reducing stress, time, gas and possible accident, but also the environment by reducing emissions on multiple areas.

5. Related work

Riva [1] presented two middleware architectures, implemented in Java, that can support the development and execution of mobile sensing applications in Urbanets. The first model, *Contory,* considers Urbanets as a distributed sensor database and offers a simple SQL-like interface to application developers. The second approach, *Context-aware Migratory Services,* provides a client-server model, where services are capable of migrating to different nodes in the network in order to serve clients' requests. When a node cannot host a certain service any longer, the current service can migrate autonomously to a new node that is qualified for accomplishing the current task, without performing any service discovery. The migratory service saves all the state information necessary to resume the interaction with the client after the migration to a different node, providing at the same fault-tolerance to service failures. The solutions presented here are able to provide real-time information to users even without Internet connectivity, assuming only the cooperation among Urbanet devices.

Wu *et al.* [2] proposed a service-oriented architecture (SOA) for smart-home environments, based on OSGi and mobile-agent (MA) technology. The architecture described here is a peer-to-peer interaction model with multiple OSGi platforms. It is based on the interoperability between agent hosts using a web services-based migration mechanism for MAs. The service-oriented approach allows the components to dynamically join/leave the system, and the SOA coordinates the interactions between these platforms.

Waluyo *et al.* [3] presented a unified on-demand and data broadcast model for participatory mobile sensing system in 4G wireless network with multiple-input multiple-output (MIMO) antennae. This system implies large amounts of data that has to be effectively processed and queried for the mobile clients to access the data in real time and visualize it in an interactive multimedia interface. Using a new data structure and indexing method the clients have the possibility to issue query using on-demand or broadcast channel according to the server load and broadcast schedule, this approach offering substantial efficiency and autonomy for mobile clients when retrieving data.

6. Conclusions and future work

In this paper we present a middleware design for sensing and processing information generated in vehicular networks. The middleware is being implemented using the OSGi technology that has proved to be best suited considering the characteristics of the devices present in this type of networks. The division of the middleware into multiple layers facilitates the development and deployment of mobile applications. This middleware will provide the environment needed by software developers to create applications that can be used by the end users as well as other programmers. The solution described in this paper is suited for both large and small urban areas with the difference that in the first case the data gathered are more accurate and the problem of traffic congestion becomes more stringent.

We are currently completing a prototype of the middleware and some scenario application to validate the proposed approach.

References

[1] Oriana Riva, "Middleware for Mobile Sensing Applications in Urban Environments", Academic Dissertation, Faculty of Science of the University of Helsinki, October 2007

[2] Chao-Lin Wu, Chun-Feng Liao, Li-Chen Fu, "Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology*", IEEE Trans. on Systems, Man, and Cybernetics - Part C: Applications and Reviews,* vol. 37, no. 2, March 2007

[3] Waluyo Agustinus, Taniar David, Rahayu Wenny, Srinivasan Bala, "A Unified Query Processing for Efficient Mobile Multimedia in Participatory Embedded Sensing System", *Transactions on Embedded Computing Systems,* December 2010

[4] Jan S. Rellermeyer, G. Alonso, "Concierge: A Service Platform for Resource Constrained Devices", Proceedings of the EuroSys 2007 Conference

[5] OSGi Alliance: OSGi Service Platform - Release 4

[6] Andre L.C. Tavares, Marco Tulio Valente, "A gentle introduction to OSGi", ACM Sigsoft Software Engineering Notes, Volume 33 Issue 5, September 2008

[7] Apache Felix http://felix.apache.org

[8] Microsoft Robotics http://msdn.microsoft.com

[9] Jose Santa, Benito Ubeda, Antonio F. G. Skarmeta, "A Multiplatform OSGi Based Architecture for Developing Road Vehicle Services", 4th IEEE Consumer Communications and Networking Conference, 2007

[10] Jan S. Rellermeyer, G. Alonso, "Services Everywhere: OSGi in Distributed Environments", EclipseCon March 2007

[11] R.S Hall, H. Cervante, "An OSGi implementation and experience report", 1st IEEE Consumer Communications and Networking Conference, 2004

[12] Equinox http://www.eclipse.org/equinox/

[13] J. Dunkel, A. Fernandez, R. Ortiz, S. Ossowski, "Event-Driven Architecture for Decision Support in Traffic Management Systems", Proceedings of the 11th International IEEE Conference on Intelligent Transportation Systems, Beijing, China, October 12-15, 2008

[14] V. Gianuzzi, A. Merlo, "Beaconing support in publish-subscribe middleware for vehicular application", Proceedings of the 2nd International Workshop on Middleware for Pervasive Mobile and Embedded Computing, November 2010