ROBÓTICA 2007 - 7th Conference on Mobile Robots and Competitions
Centro Paroquial de Paderne, Albufeira, Portugal
April 27, 2007

Data centric middleware for the integration of wireless sensor networks and mobile robots

Pablo Gil, Iván Maza, Aníbal Ollero and Pedro José Marrón

***Abstract*— This paper describes the implementation of a data-centric middleware for wireless sensor networks in the scope of the European project AWARE. The middleware implements a high-level abstraction for integration of wireless sensor networks with mobile robots. This is achieved by providing data-centric access to the information gathered by the wireless sensor network, which includes mobile robotic nodes. Nodes in the network organize themselves to retrieve the information needed by the robots while minimizing the number of transmitted packets in order to save energy. The implementation has been tested on *tmote* and *Mica2* nodes.**

I. INTRODUCTION

Wireless sensor networks (WSN) are collections of small devices equipped with sensors that communicate with each other using wireless technology and are able to organize themselves in order to interact with their environment. In contrast to classic sensors that usually need to be placed in specific locations carefully engineered to get the most out of expensive equipment [1], WSNs rely on redundancy and cheap hardware, rather than on accurate sensors, to make good estimations of the variables to monitor.

Applications of WSNs range from environmental data gathering to public safety, and might include integration with robotic systems. An example of integration with robotic systems consists of a robot which obtains data from its environment through a WSN. These nodes gather temperature, light, humidity or even acceleration readings. Radio signals emitted by the nodes might also be used for location and tracking purposes, thus allowing the robot locate itself even in environments where GPS is not available [2], [3].

The integration of WSNs and robotic systems is two fold. First the robots can be considered to be mobiles nodes that provide additional sensorial information, improve/repair the connectivity and collect information from static nodes. On the other hand, the WSN can be considered as an extension of the sensorial capabilities of the robots.

The use of robots as mobile nodes in WSNs has been explored in the last years. Particularly, nodes based in *Crossbow* products, like the sensor node *Mica2* have been used in several experiments. These mobile nodes are usually applied in indoor environments. Several types of robotic mobile nodes can be found in the literature such as MICAbot [4], CostBots [5], Robomote [6] and Millibots [7]. In [3] the use of the sensor network to guide a robot with minimal sensor capabilities is proposed. On the other hand, some research activities on the use of robots to deploy WSNs and to improve their performance has been initiated [8].

The AWARE project (whose name stands for platform for Autonomous self-deploying and operation of Wireless sensor-actuator networks cooperating with AeRial objEcts) is sponsored by the Information for Society Technologies (IST) and aims to provide a middleware for integration of the information gathered by different type of sensors -including a WSN- and mobile robots [9]. This integration between mobile robots and a WSN is a problem which does not have a complete solution yet, thus the interest of research on this subject.

The rest of the document is structured as follows. Section II describes the AWARE system, in which this middleware is integrated. Section III points out the characteristics of the proposed solution. Section IV deals with the implementation details of the software. Experimental results are described in section V. Finally, a conclusion and an overview of future work on this field is outlined in section VI

II. SYSTEM DESCRIPTION

The AWARE platform consists on two different networks [9], a high bandwidth network (HBN) and a low bandwidth network (LBN). The HBN is composed of personal computers, cameras and mobile robots capable of transmitting data through IEEE 802.3 or IEEE 802.11 networks. A WSN is also present on the system. This second network is formed by nodes with very limited computing and data transmitting capabilities, and it is also called the low bandwith network (LBN). HBN and WSN are connected through a *gateway*. Some mobile robots might be also part of both networks. This situation is shown in Fig. 1, where an Unmanned Aerial Vehicle (UAV) and an autonomous ground vehicle both have WSN nodes attached. Any device capable of direct communication with both networks might act as a gateway. The purpose of the middleware in the AWARE system is to provide seamless communication between entities in both networks. The middleware described in this paper runs entirely on the WSN and aims to test the suitability of a data-centric approach [10], [11] access to the information gathered by the WSN in the AWARE system.

This work was partially funded by the Information Society Technologies and the Spanish Science and Education Ministry

Pablo Gil, Iván Maza and Aníbal Ollero are with the Robotics, Control and Vision Group
University of Seville, Camino de los descubrimientos S/N, 41092, Seville, Spain.
{pgilmon,imaza,aollero}@cartuja.us.es
Pedro José Marrón is with the Distributed Systems Research Group
University of Stuttgart, Universitaetsstrasse, 38, D-70569, Stuttgart, Germany
pedro.marron@informatik.uni-stuttgart.de

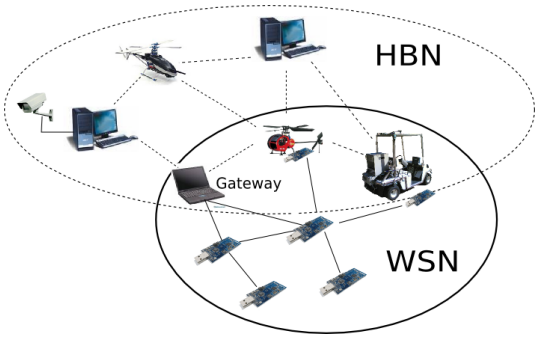

Fig. 1. AWARE platform elements

For the rest of the document, we are going to consider a WSN already deployed over an area in which certain events or variables need to be monitored. Some of the nodes might be robotic mobile nodes. After deployment, nodes are in a quiescent state. The user, which might be a human operator or an application running on any IP-networked device, denotes interest in certain events by defining *groups*. The user also defines *channels*, which indicate what kind of information is to be obtained from the groups. Group and channel definition is done in terms of *attributes*, which are numerical values (e.g.: a sensor reading, the position of the node). Interaction between the user and the wireless sensor network always involves the gateway. In our particular case, the user is a mobile robot requiring information from the WSN.

The system dynamically adapts to changes in network topology due to addition of new nodes, changes in the environment and node failures. This is important because nodes in the WSN are subject to energy constraints [12], [13] and are prone to stop working due to battery depletion. When new nodes are added to a deployed system, configuration information is transmitted to these new nodes by the previously deployed ones, without any intervention by either the gateway or the user. This configuration information consists on group and channel definition (see sections II-A and II-B for details). Once the new nodes have receive these data, they have all the information required to fully integrate with the deployed system and begin operation. The system is quite robust in this sense, one only surviving node is enough to ensure that newly added nodes are automatically configured.

A. Groups

Information about the types of groups defined by the user is distributed epidemically through the network. One type of group defines some environmental conditions. Whenever the values read by the sensors of one node meet these conditions the node belongs to that type of group. The environmental conditions that determine membership to a group type are fixed when the group is defined, and consist of several conditions that one or more attributes must meet. Group definition might be changed on-line by the user. Fig. 2 shows several objects in the environment that are mapped as groups in the WSN.

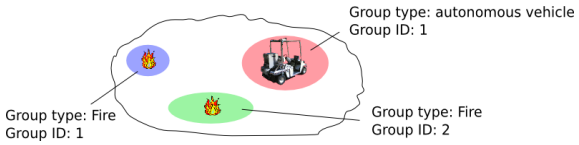

Fig. 2. Different objects mapped as groups

A restriction is introduced to ensure that different objects are mapped as separate groups: communication between nodes that belong to the same group must be possible without passing through nodes that do not belong to that group. This permits the middleware to identify groups of the same type that are physically separated as different groups, even if they are of the same type. Communication takes place via multi-hop routing algorithm, therefore groups might be larger that the area covered by node's radio transceivers.

Each group is identified by two numbers: *group type* and *group identifier (group ID)*. It can be considered as a two-level hierarchical identifying system.

Groups form in those areas where certain sensor values are satisfied. If those areas move, groups follow them, keeping the same identifier. This way, a certain physical object (e.g.: fire, gas escape, a robot) can be tracked. Whenever two groups of the same type merge, one of the two ID's is adopted by all the nodes, so at the end there is only one group. Groups of the same type merge when communication between their their members is possible by a multi-hop route comprised exclusively of nodes that belong to those groups. When a group splits into two or more groups, new group ID's are generated in order to name the additional groups that appear after the split.

Each group has a group leader, which is elected as the group is created. The election algorithm guarantees that there is only one leader in each group. When the leader goes out of the area of the group, either because environmental conditions change or because nodes are moving, it sends a handover message to a member of the group, so that the group ID is maintained. When two groups of the same type merge, one of the leaders leaves the position and becomes a member of the merged group. If the leader dies, a new leader election takes place. The leader regularly sends a beacon to build a group routing tree and to let other members of the group know that it is still alive. Group routing tree is used for data aggregation inside the group (see section II-B for more details).

B. Channels

Information about active channels is epidemically distributed on the WSN. When a channel is defined, the user must select which sensor is to provide the data for that channel, and the group type and ID to which the channel is associated. It is also possible to associate the channel with all the groups or even with all the groups of the same kind. Information about the sampling rate is also part of the channel definition, and might be changed later by the user. These changes are made on-line and take effect immediately.

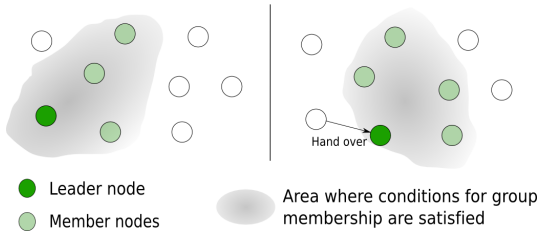

Fig. 3. Situation previous to (right), and after (left) handover

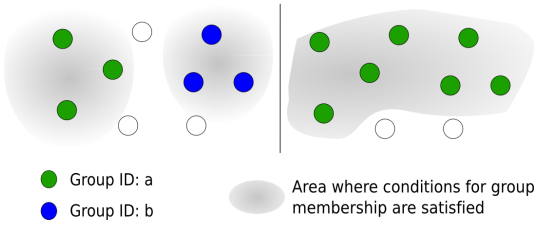

Fig. 4. Two groups before (right) and after (left) merge

Data aggregation is performed by every node in the group, in order to minimize energy consumption. The group leader performs final aggregation, minimizing the data sent to the node attached to the gateway. Once the leader finishes data aggregation it sends the data to the gateway through a multi-hop routing tree.

Command channels are supported too. Commands are also distributed epidemically. When a new command packet arrives at one node, the middleware calls the functions associated with that command. These functions are provided by the application in the nodes (see section IV-A).

III. DATA-CENTRIC APPROACH

The idea behind group creation based on environmental conditions is to provide an abstraction that allows the user of the network to make references to objects that exist in the environment, such as a fire, a water spill, etc. The user just has to provide the conditions that define a group. In the case of a fire it could be high temperature and certain readings of gas sensors. The middleware then organizes groups that match those conditions (e.g.: if there are two separate fires the user must have an different identifier for each fire). Those identifiers give the user the possibility to address objects in the environment in order to obtain data from them. This is done by defining *channels*.

Group management also involves object tracking. In the previous example, if one of the fires is spreading, groups are reorganized, so that the group that corresponds with that fire *moves* along with the fire while maintaining the same identifier. This is done by adding new nodes to the group in the direction where the fire is spreading and removing nodes from the group where there is no longer fire. In case the leader stops belonging to the group either because the object (e.g.: fire) is moving or because the node itself changes its position, a hand over message is transmitted by the leaving leader. This hand over message is sent to another member of the group, which then becomes the new leader. The member of the group to which the message is sent is known by the leader from the information received from channel data (see section IV-F). An illustration of hand over situation is shown in Fig. 3.

In addition, two objects can merge in the environment (e.g.: two fires might join to form a larger one). Group management takes this fact into account and allows two groups of the same type merge. When two groups defined by the same conditions get together they become one only group with one identifier. One group can also split in two or more smaller groups (see figure 4).

Groups always have a *type identifier* and a *group ID*. The type identifier is associated with the conditions that define that group. Thus, two groups of the same type are always defined by the same conditions. However, there might be more than one group of the same type. For instance, there might be two separate fires, and so there should be two different groups of the type that defines the conditions for *fire*. Group ID allows the user to distinguish between two groups of the same type. Every time a new group is created the leader generates a group ID, which is mantained throughout the whole life of the group, even if the initial leader leaves the group. Group ID is a 16-bit value in current implementation, so it is highly unlikely, although not impossible, for two groups of the same type to have the same group ID. Group merging and splitting imply destruction and generation of group ID's as described in section II-A.

IV. IMPLEMENTATION

An implementation of the middleware has been developed and tested on *tmote* and *Mica2* nodes. The implementation supports three group types, with a virtually unlimited number of groups of the same type. With respect to the channels, three simultaneous channels are supported. One group might have more than one associated channel. These figures were chosen for testing convenience and might be increased by changes in compile-time constants. The middleware also supports delivering of commands to the application running on the nodes for local execution in the mote.

Information about the commands, active channels and active group types is provided by the user and epidemically injected in the wireless sensor network from the gateway. This communication scheme has been chosen to provide the maximum robustness to this communication. This is necessary because the global performing of the system depends on these data being delivered correctly. Epidemical distribution provides one of the most reliable communications possible in a wireless sensor network. The chosen implementation for epidemical distribution is based on TinyOS Drip component, which includes several mechanisms for bandwidth and battery saving [14]. Drip provides the robustness of epidemical distribution with a reasonably low battery consumption.

Channel data published by the nodes reaches the gateway by using a hierarchy of collecting trees consisting in two types of routing trees: a global one, and another one asso-

ciated to each group. Group routing trees are exclusive to each group and are used for data aggregation inside a single group. Several group routing trees might exist in the system at any given time, since every group has its own routing tree. Group routing trees are built by taking advantage of leader beacon signals (see section II-A): as the beacon progresses through the group, nodes make use of it to infer the shortest path in terms of number of hops to the group leader.

There is another routing tree which is global (i.e.: there is only one in the whole network). This global tree is used by leader nodes to send the aggregated data related to their groups to the gateway.

A. Software architecture

Software on the motes runs under TinyOS and has been written in nesC language. TinyOS is an operating system designed for small devices with limited resources, with focus on WSN devices [15], [16]. TinyOS and nesC allow a highly modular programming scheme [17]. The software is composed of different modules that can be effortlessly exchanged to add new functionalities or to improve existing ones. Software running on the motes is structured in two layers: *Application* and *Middleware*. The middleware layer provides the necessary functions to allow access to the motes in a data-centric *publish/subscribe* [18] approach from the gateway. The application layer is in charge of providing the middleware with information about the node itself: which kind of data the node can provide, and how to obtain that data. This architecture allows the use of the same middleware in all the motes, even if they have different types of sensors. *Attributes* also contribute to make the middleware hardware-independent. Different sensors might provide information about the same physical variable. This information might be represented in a standard unit (e.g.: SI units) by an attribute value. The application must provide the proper function to convert the raw sensor reading to the appropriate attribute value, freeing the middleware from this hardware-dependent task. Attributes are provided by TinySchema [19], a collection of TinyOS components available with TinyOS standard distribution. The use of this modular structure allows dealing with heterogeneity in the WSN hardware, and makes it easy to upgrade and maintain the software.

B. Group management algorithm

Nodes periodically check whether the sensor readings match the necessary conditions to determine membership to every type of group. When conditions for some group are satisfied, the node checks whether it knows any group of that kind in its vicinity. If that is the case, the node joins that group as a member node, and becomes a publisher of any channel to which that group might be associated. If there is no group of that type in its vicinity the mote creates a new group and becomes the leader of that group. When a new group is created, several leaders might appear. This situation is solved by a leader election algorithm, that guarantees that only one leader prevails. In fact, this situation is similar to a group merge, so the same algorithm solves both problems.

C. Leader election

When a mote creates a group it becomes leader of that group. The group ID is then set to the ID of the creator mote, and the *weight* of that group leader is set to zero. As soon as one mote becomes leader of one group it begins to send *leader beacons*, that let the motes in the vicinity know about the existence of a group.

Leader beacons are rebroadcasted by all the motes that belong to the group, so they eventually reach all the members of the group, regardless of how many hops are necessary to reach them from the leader. Motes that do not belong to the group do not forward the beacons, but they listen to them. Thus, motes that are just outside the border of a group know that the group exists. Leader beacons include a sequence number. This is necessary to prevent multiple retransmissions of the same beacon.

Motes that know about the existence of a group join it as soon as their sensor readings satisfy the conditions of that group type. Once they have joined the group, they began to transmit data regarding the channels to which that group is associated. Data aggregation is performed in the group members (see section IV-F) and the data eventually reaches the leader through the group routing tree. The leader uses the data received from the members to increase the field *weight* in its beacon. Thus, the more data a leader has received from the member, the higher its weight is. When two groups merge, or an spontaneous leader appears, one of the leaders must prevail over the other. This task is accomplished by two mechanisms: first, members only forward beacons from the *heavier* leader they know; second, when a leader receives a beacon from a heavier leader it becomes a member of that leader's group. When two or more leaders of the same weight are in conflict, group ID is taking into account, so the one who has the higher group ID is selected. This last feature is implemented in both mechanisms described before.

D. Association between groups and channels

When a channel is established, it is possible to choose which group should publish information on that channel. Current implementation allows the user to establish the group type and the group ID associated with that channel. There are special values for *any group type* and *any group ID*. Thus, it is possible to require all the groups of a certain type to publish information on one channel, as well as all the groups or just one single group. In case there is more than one group publishing information in one channel, it is possible for the user to distinguish the source of every piece of data, because every data packet includes the group ID of the source.

E. Channel publishing mechanism

Motes regularly check whether there is any active channel associated with any of the groups to which the mote belongs. In case a node is to publish on a channel, it obtains the attribute value at the appropriate rate, and publishes it using an aggregation algorithm and the group routing tree.

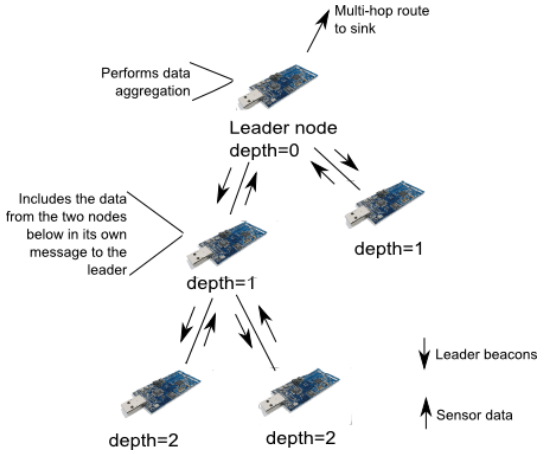

Fig. 5. Channel data aggregation

F. Data aggregation

Data aggregation is very important in a WSN [20], where it is necessary to reduce the amount of data transmitted due to power constraints. In this implementation, aggregation of channel data is performed at every node in the group routing tree. Every node receives the data to be published on the channel from its child nodes and sums all the values received. Then the node sends to its parent node this information, along with the number of measurements that have been used to perform the sum. Eventually, the leader receives the sum of all the readings from all the nodes, as well as the number of readings, so it can perform the average of all readings. This approach provides the average (AVG) of all the readings in the group. The software architecture is designed to support additional well-defined aggregation functions. Once the leader node has finished the aggregation it sends the data through the global routing tree to the gateway. Fig. 5 represents this data aggregation process.

V. EXPERIMENTAL RESULTS

Preliminary experiments on the integration of WSNs and robots have been also performed. For these experiments, *Mica2* nodes have been used. The experiment consisted on tracking Romeo 4R autonomous vehicle by using a group. In order to detect the presence of Romeo 4R a node was attached to it (see Fig. 6).

The node attached to Romeo 4R emitted a beacon signal that was recognized by the rest of the nodes. A group was created based on the Received Signal Strength Indicator (RSSI) associated with the signal emitted by the mote on Romeo 4R. As the robot moved, the group automatically followed it, either adding or removing members as necessary. A proper threshold value was chosen for membership to the group. RSSI decreases its value as received power increases. We assumed that a higher received power corresponds to a nearer emitter, so the group was formed by nodes whose RSSI value was below the threshold. For the experiments, a 12.5 x 11 meter area was populated with nodes as shown in Fig. 7. Romeo 4R autonomous robot then passed through

Fig. 6. Romeo 4R autonomous vehicle and detail of WSN node on the robot

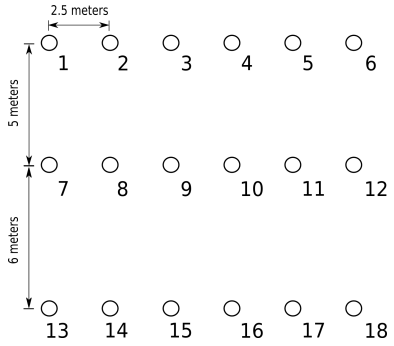

Fig. 7. WSN nodes situation in the experiments

the populated area, and the nodes performed a tracking of the robot. A sequence showing the nodes that belong to the group tracking the mobile robot is shown in Fig. 8. RSSI values were recorded during the experiments. Fig. 9 shows some of these values compared with the RSSI threshold that determines membership to the group.

VI. CONCLUSIONS AND FUTURE WORK

A. Conclusions

Experiments carried out show the ability to track a mobile robot by using this approach, as well as obtaining information from objects in the environment just by associating groups to them. This middleware represents a totally autonomous implementation of data-centric capabilities on a WSN. Data-centric capabilities are desirable in a WSN because they allow access to data in an abstract and simplified way, freeing the user from remotely monitoring sensor readings not of their interest and allowing an easier integration with other systems. The experiment described in this paper shows the possibilities of this data-centric approach for integration with mobile robots. The mobile robot is able to obtain data from its surroundings just by querying the group

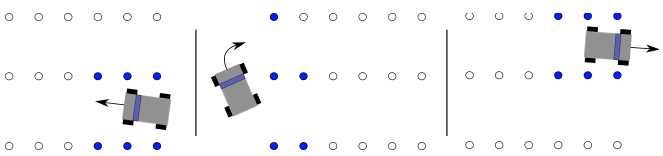

Fig. 8. Sequence showing group tracking a mobile robot

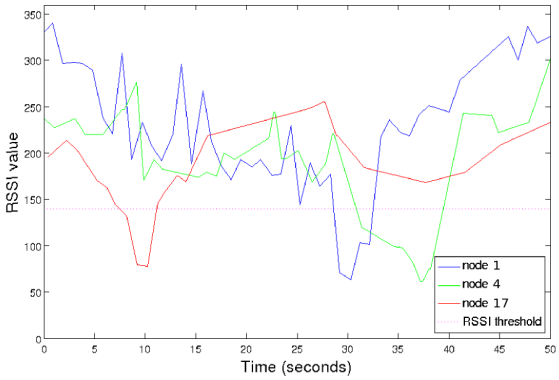

Fig. 9. RSSI values for motes 1, 4 and 17

created. The middleware takes care that the group is always formed by the nodes that are near the robot and delivers the data to it, even if the robot and the nodes move, ensuring that the robot always gets the expected data.

B. Future work

Group management in terms of attributes allow addition of countless features to the system. Attributes are numerical values that might be related to any variable available on the motes. An interesting application of this fact is to associate an attribute with the position of the mote. The middleware would then be able to take the position into account. Since the position would be the value of an attribute, no change in the system architecture is necessary to add this feature. The middleware would treat it as an additional attribute. Research is currently being carried out in the field of node estimation [2], and the results could be integrated with the middleware. Once the motes become location-aware, the middleware allows to perform very useful tasks such as location-associated monitoring and event localization for data coming from mobile nodes. Adding positioning capabilities to the object tracking features of the middleware allows to monitor an object also in its position. The combination of these features with the middleware described in this paper would allow a robot to monitor its environment by using the WSN while, at the same time, obtaining positioning information about the events or objects in its surroundings.

According to the system description in Fig. 1, it is possible to integrate information gathered by the middleware with the mobile robot's navigation system. In the scenario depicted in Fig. 8, the robot could act as a data collector, as well as modifying its trajectory depending on data obtained through the middleware.

VII. ACKNOWLEDGMENTS

This work was partially funded by the European Commission under the AWARE project (IST-2006-33579), and by the Spanish Government under the AEROSENS project (DPI-2005-02293). The first author visited the University of Stuttgart with the funding of the Embedded Wisents Coordination Action (FP6-004400). The authors gratefully acknowledge the support of the Sensor Networks research group, Distributed Systems Department, University of Stuttgart, as well as the assistance for the experiments of Francisco Real and other members of the Robotics, Control and Vision Group of the University of Seville.

REFERENCES

[1] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, E. Cayirci, *A survey on sensor networks*, IEEE Communications Magazine, vol. 40, issue 8, August 2002, pp 102-114.

[2] Vaidyanathan Ramadurai, Mihail L. Sichitiu, *Localization in Wireless Sensor Networks: A Probabilistic Approach*, Proceedings of the 2003 International Conference on Wireless Networks, 2003, pp 275-281.

[3] Maxim A. Batalin, Myron Hatting, and Gaurav S. Sukhatme, *Mobile Robot Navigation using a Sensor Network*, Proc. of the 2004 IEEE Intl. Conference on Robotics and Automation, April 2004.

[4] M. Brett McMickell, Bill Goodwine, and Luis Antonio Montestruque, *MICAbot: A Robotic Platform for Large-Scale Distributed Robotics*, Proc. of the 2003 IEEE, Intl. Conference on Robotics and Automation, September 2003.

[5] Sarah Bergbreiter, and K.S. J. Pister, *CostBots: An Off-theShelf Platform for Distribuited Robotics*, Proc. of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, October 2003.

[6] Karthik Dantu, Mohammad Rahimi, Hardik Shah, Sandeep Babel, Amit Dhariwal and Gaurav Sukhatme, *Robomote: Enabling Mobility in Sensor Networks*, ACM Journal Name, Vol.1 No. 1, December 2004.

[7] Luis E. Navarro-Serment, Robert Grabowski, Christiian J.J. Paredis, and Pradeep K. Khosla, *Millibots*, IEEE Robotics and Automation Magazine, pp. 31-40, December 2002.

[8] P.Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme, *Autonomous Deployment and Repair of a Sensor Network using an Unmanned Aerial Vehicle*, Proc. of the 2004 IEEE Intl. Conference on Robotics and Automation, April 2004.

[9] The AWARE project team *AWARE project webpage*, http://www.aware-project.net, last visited: 24-02-2007.

[10] Chien-Chung Shen, Chavalit Srisathapornphat and Chaiporn Jaikaeo, *Sensor Information Networking Architecture and Applications*, IEEE Personal Communications, vol. 8, issue 4, August 2001, pp 52-59.

[11] Abdelzaher T., Blum B., Cao Q., Chen Y., Evans D., George J., George S., Gu L., He T., Krishnamurthy S., Luo L., Son S., Stankovic J., Stoleru R., Wood A., *EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks*, Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04).

[12] Kay Romer, Oliver Kasten, Friedemann Mattern, *Middleware Challenges for Wireless Sensor Networks*, Mobile Computing and Communications Review, vol. 6, Number 4.

[13] Yang Yu, Bhaskar Krishnamachari and Viktor K. Prasanna, *Issues in Designing Middleware for Wireless Sensor Networks*, IEEE Network, vol. 18, issue 1, January/February 2004, pp 15-21.

[14] Gilman Tolle *Drip 1.0 documentation*, http://www.tinyos.net/scoop/story/2005/2/16/174147/450, last visited: 05-02-2007.

[15] TinyOS team - various authors *TinyOS-1.x web documentation*, http://www.tinyos.net/tinyos-1.x/doc/index.html, last visited: 05-02-2007.

[16] Philip Levis, *TinyOS programming*, http://csl.stanford.edu/ pal/pubs/tinyos-programming.pdf, last visited: 05-02-2007.

[17] nesC team - Eric Brewer, David Culler, David Gay, Phil Levis, Rob von Behren and Matt Welsh *nesC project site*, http://nescc.sourceforge.net/, last visited: 05-02-2007.

[18] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, Anne-Marie Kermarrec, *The many faces of publish/subscribe*, ACM Computing Surveys, vol. 35 , Issue 2, June 2003, pp 114-131.

[19] Wei Hong and Sam Madden, *TinySchema: Managing Attributes, Commands and Events in TinyOS*, http://telegraph.cs.berkeley.edu/tinydb/tinyschema_doc/index.html, last visited: 05-02-2007.

[20] Krishnamachari, L. Estrin, D. Wicker, S., *The impact of data aggregation in wireless sensor networks*, Proceedings of the 22nd International Conference on Distributed Computing Systems, 2002, pp 575-578.