Research Repository UCD

Title	Autonomous management and control of sensor network-based applications
Authors(s)	Ruzzelli, Antonio G., Muldoon, Conor, Schoofs, Anthony, Campana, Tiziana, O'Hare, G. M. P. (Greg M. P.), Tynan, Richard
Publication date	2009-08
Publication information	Ruzzelli, Antonio G., Conor Muldoon, Anthony Schoofs, Tiziana Campana, G. M. P. (Greg M. P.) O'Hare, and Richard Tynan. "Autonomous Management and Control of Sensor Network-Based Applications." IEEE Computer Society, 2009.
Conference details	Paper presented at the 2nd International Workshop on Adaptation in Wireless Sensor Networks (AWSN-09), in conjunction with The 7th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC 2009), August 29 - 31, 2009, Vancouver, Canada
Publisher	IEEE Computer Society
Item record/more information	http://hdl.handle.net/10197/1809
Publisher's version (DOI)	10.1109/CSE.2009.480

Downloaded 2023-04-07T10:52:29Z

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)

© Some rights reserved. For more information

Autonomous Management and Control of Sensor Network-based Applications

A.G. Ruzzelli, C. Muldoon, A. Schoofs, G.M.P. O'Hare and R. Tynan

CLARITY centre for SensorWEB technologies,

University College Dublin

{ruzzelli, conor.muldoon, anthony.schoofs, gregory.ohare, richard.tynan}@ucd.ie

Abstract—A central challenge facing sensor network research and development is the difficulty in providing effective autonomous management capability. This is due to a large number of parameters to control, unexpected changes of the network topology and dynamic application requirements. Network management is also a challenging task for the remote user due to the large-scale of the network and scarce visibility of live network happenings. Preferably the network should have autonomous decision-making capabilities as network conditions and application requirements changes. To cope with such uncertainties, firstly we consider Octopus, a powerful software tool that provides live information about the network topology and sensor data. At present, the tool can provide monitoring and require a user to control the network state manually. This paper describes how Octopus is reengineered to accommodate a multi-agent system to provide autonomic managing capabilities. In particular, we detail two distinct architectures, the static and mobile agent architectures, which can be effectively applied to deliver autonomous system management. This paper sets the basis for a full autonomous network management via a multi agent system to work with Octopus.

I. INTRODUCTION

Wireless Sensor Networks (WSNs) can be effectively used for delivering both new context-aware applications and enhancing existing applications. However, recent developments such as in [16] have demonstrated the need for adaptive tools which can cope with unexpected variations within the network operating environment while ensuring maintenance of an adequate quality of service. This is currently achieved by an exchange of information among nodes within the network and allowing a remote user who receives such data from the network to monitor and alter the network condition from the base station. The goal is to provide is to successfully deliver a certain application as well as to be able to detect when a network anomaly or an unwanted behaviour occur. In order to simplify these tasks, recent advances have permitted the development of a dashboard, namely Octopus [13], which assists developers with visual network debugging, network assessment, and the gaining of an understanding of network topology and routing patterns. Octopus provides a user-friendly tool with which to localize nodes, formulate application queries and tune parameters according to a given application requirements as well as interacting with the network by monitoring and reconfiguring a number of parameters of application and radio. Currently however, Octopus requires a user to manually monitor the network and take action accordingly. This

paper investigates how Octopus can be significantly enhanced through a Multi-Agent System (MAS) to deliver autonomous network monitoring and management.

The paper is organised as follows: Section II reviews the state of the art in terms of architectures utilised for managing sensor networks. Section III focuses on the main features of the Octopus Dashboard for visual sensor network control. Section IV describes how this dashboard is utilized as a basis to deliver autonomous network management. Finally, Section V details a practical case study of the architecture before concluding the paper.

II. RELATED WORK

Octopus [13] currently represents the most advanced tool for sensor network control and monitoring. Prior tools addressed network state monitoring and network control separately. For example Mviz [14] is the only visualization tool in TinyOS v.2 that provides a basic visualization with no features for remote network control. The closest related work to Octopus is probably Surge [15] which was developed for TinyOS v.1 and is unsupported for TinyOS v.2. Surge enables the user to visualize nodes, put them to sleep and to set node sampling period individually through the *Focused Mode*. However, Surge merely represents early work toward sensor network interaction and it depends heavily upon a set of specific protocols defined by TinyOS v.1. Surge offers no support for sensing thresholds, network queries and alerts, node localization and no energy consumption estimation. Furthermore, Surge lacks important setting requests such as duty-cycle, sampling period, and energy consumption estimation. The Octopus dashboard addresses all these drawbacks providing both developers and users with an instrument that combines an accurate insight into the network state and online node reconfiguration. However, the dashboard still requires manual intervention from a remote user who observes the live network operations and reconfigures a number of network parameters during normal network activity.

Further related works address a human-managed dynamic network control but offer no visualization service through over the air reprogramming. This approach is typified by such systems as Deluge [4] and Mate [7]. These tools are useful for injecting new code images or new modules into the networks involving a significant data overhead and network restart. This is only required for less frequent reload of large

code components or an entire application but is unsuitable for frequent tuning of application and network parameters.

The need for Autonomic Wireless Sensor Networks (AW-SN) together with the potential for agents as a candidate technology for realizing such has been recognized in the literature [9]. Historically strong agent approaches have been viewed as computationally demanding and as such inappropriate for computationally challenged devices such as sensor platforms. While a number of agent frameworks have been developed for mobile devices [1] [6] [5] [8] [3] [11] [12] [17], such as mobile phones and PDAs, as yet there are no intelligent agent frameworks for sensor nodes. To deploy such frameworks in a WSN setting, significant changes would have to be made to their networking platform services. Indeed, some of these frameworks do not include a networking capability. Agilla [2] is an agent platform for embedded devices, but it does not contain reasoning capabilities and therefore does not conform to the strong notion of agency.

There have been several frameworks developed to facilitate agent migration. At present, however, most of these frameworks are based on protocols that are not supported by WSN nodes. For such frameworks to be deployed in a WSN setting, significant modifications would be required.

III. THE OCTOPUS DASHBOARD

As shown in Figure 1, Octopus consists of a front-end Graphical User Interface (GUI) developed in Java and an embedded nesC-based application uploaded onto the sensor nodes (motest). The GUI is organized in 4 main components: (1) The **SCOUT** component listens to the serial port for incoming packets from the network and is responsible for transmitting requests into the network; (2) The **MOTE DATABASE** and (3) The **LOGGER** are collectively responsible for storing the sensed data for future analysis; (4) The **PANEL BOARD** which includes the visualization options and panels for interfacing with the network.

The standard configuration of the Octopus embedded application utilizes the interfaces of *CC2420* radio, *LowPowerListening* access control and *Collection Tree Protocol* (CTP) routing provided in TinyOS 2. However, the dashboard is independent from underlying protocols that may be defined in the Octopus configuration file for example to debug and evaluate new algorithms.

Facilitating future extensions and code reuse through conformance to certain standards is an important objective of the dashboard. Octopus presents a JavaBeans component-based architecture that distinguishes between a component-based infrastructure framework and a set of functional components. In particular, the architecture is based on the features associated to the *BeanContext* class. The framework eases much of the component composition while the abstract API enables different possible implementations of these features. The Octopus architecture enables new or existing components to be plugged-in without altering the remainder of the system. This permits a high degree of configurability and also the creation of different versions of Octopus in order to satisfy specific deployment scenarios and application requirements.

The dashboard provides 3 main views namely: (1) Network Map *NP* for visualizing the topology of the network; (2) Network Chart *NC* for plotting live nodes sensed data; (3) Floor Plan *FP* for displaying a map of the physical location where the network is placed. *NM*, *NC* and *FP* are selected by the tabs a, b, and c in Figure 1 and can either be displayed separately or in overlapping modalities. Figure 1 shows the *NM/FP* view. Initially nodes are located randomly on the board, however, the combined view permits the execution of an interactive localisation algorithm as explained in [13]. Finally, a Legend panel allows the setting and the visualisation of further network parameters.

On the right-hand side of the dashboard, a side menu contains panels for the user to formulate application requests, tune radio parameters, set email alerts, and support interactive node localization. The dashboard visualizes incoming packets live by the *Consol* panel at the bottom of the board that provides also some checkboxes to filter the messages displayed, as shown in Figure 1 tab i. A Legend panel allows selecting more visualization options on the main board and logging the data into a file.

IV. SELF-MANAGEMENT FOR OCTOPUS

Imbuing Octopus with autonomic capabilities such as network control and network reconfiguration is a non trivial task. It necessitates the development of an autonomous dashboard by which to enable network self-governance through interacting with the nodes and adjusting its parameters. This is a challenging task due to possible unexpected network behaviour, node failures and potential temporary disconnection of some part of the network. For example, a lack of data from a certain node may be linked to network congestion. On the other hand it can represent a node failure. Reducing the transmission rate of the neighbourhood can therefore be helpful in understanding whether the node should be replaced or the user should be alerted that a particular event has been detected and as a result part of the network is experiencing an increase of node activity. Furthermore, sensor nodes often offer inexpensive hardware and consequently the system should be tolerant to imprecise data emanating from the network and should be able to effect decisions with a partial information. Longitudinal examination of the history of data can ensure that the information received can be trusted before alerting the user of a particular event. The interplay between these network parameters, the potential for collaborative reasoning based upon potentially conflicting, incomplete information suggests the appropriateness of an agent based approach.

The component-based architecture of Octopus facilitates the easy integration of a multi-agent system. As described in Section III, the dashboard presents the Scout component which is responsible for managing the data streaming from and to the network. As shown in Figure 2, the **AGENT SCOUT** component implements an intelligent agent entity that communicate with the **PANEL BOARD**, the **MOTEDB** and the **LOGGER**. The **PANEL BOARD** component allows the agent to obtaining specific initial application requirements set by the user. The agent utilises the **MOTEDB** and **LOGGER**

Fig. 1. A screenshot of the Octopus dashboard

components to autonomously monitor the parameters of the network, to examine the history of data received before taking an action and log data into a file respectively. The agent is empowered with all information available to interface with the sensor network and set parameters in order to match the user desires. In the case of an event that would require a closer look, the **SCOUT AGENT** component is connected with the **AGENT FACTORY** component to enable the creation of a migrating agent to be injected into the network. A practical example is that of an asset tracking within a warehouse where a migrating agent is able to jump from one node to another in order to trace the location of a given entity and to provide the user with information about its state. In the next section we describe a typical case study where the architecture can be effectively deployed to deliver autonomous network management.

V. A CASE STUDY: IGALLERY

We now consider how the Octopus agent framework introduced in Section IV can be applied in the intelligent management of an art gallery media service.

Many existing art galleries are introducing a media service such as handheld devices and headphones that guides the customer through the gallery and provides him/her with information about exhibits. However, current systems lack flexibility when the collection is reorganized or if some individual exhibits are removed/added, each handheld device must be reprogrammed with a new information guide. This is a tiresome task that involves entire or partial re-registering of the guiding voice possibly in different languages and then a manually uploading on each device.

This issue can be addressed by the agent-based framework within Octopus through a self-discovery algorithm that locates each exhibit and thereafter delivers the appropriate context over the air according to the user. The agent framework

Fig. 2. The Octopus architecture with support for a Multi Agent System (MAS)

within Octopus can even enable tracking of customers so as to identify a particular pattern of user interest and deliver personal recommendations of a particular exhibit in the gallery from a certain author.

VI. AGENT ARCHITECTURE

Octopus has been designed as a modular system that enables plugins to be used that augment the system functionality with capabilities specific to a particular application. In this section, we discuss two such plugins that have been developed for the intelligent art gallery(igallery), namely the static agent infrastructure and the mobile agent infrastructure. There are several differences in the manner in which these two plugins operate. With the static agent approach, an agent resides on the Octopus host machine. That is, the agent executes within an Octopus plugin. The agent monitors the applications sampling rate and duty cycle, altering values in accordance with certain quality of service needs. With the mobile agent approach, again there is an agent operating in an Octopus plugin, but in this case as users enter the gallery, it creates user agents that migrate on to the users' mobile phones. In addition to the network management functionality of the static agent approach, the mobile agent infrastructure enables the application to modify its behaviour in accordance with perceived user needs. It monitors the user's location and path through the gallery and updates a statistics database as the user leaves the gallery. These statistics are subsequently used to identify optimal positions for exhibit locations. Key to the development of both systems is Agent Factory Micro Edition (AFME), which shall be discussed next. Following on from this, we shall describe the static and mobile agent applications in greater detail.

VII. AGENT FACTORY MICRO EDITION

AFME is a minimized footprint deliberative agent platform designed specifically for use with resource constrained devices. AFME [10] was originally intended for use with cellular digital mobile phones, but has since been ported to the leaf nodes of a WSN and specifically Sun SPOT motes. AFME is based on Agent Factory, a preexisting framework for the development and deployment of multi-agent systems for desktop and server computers. It comprises a four-layer architecture, which includes a development methodology, an integrated development environment, an agent-oriented programming language, and a runtime environment. This is supplemented with debugging, visualization, and development tools, such as the VIPER tool for Agent UML.

The agent programming language is based on a logical formalism of belief and commitment. Each agent comprises a set of roles, which are adopted at various points throughout execution. Each role consists of a trigger condition and a set of commitment rules. Once an agent adopts a belief that matches the trigger, the role is adopted and the set of commitment rules within the role are added to the agent's mental state. Subsequently, the commitment rules are evaluated on each iteration of the agent's control process until either the role is retracted or the agent is terminated. This improves the efficiency of the system in that the commitment rules of the role are only evaluated at appropriate times throughout execution. If an agent is no longer performing a role, it no longer has the reasoning overhead for the role.

The set of rules adopted when a role is triggered specify the conditions under which the agent adopts commitments. Originally, these conditions only included the agent's beliefs, but more recently, in AFME, support has been added for equalities, inequalities, and rudimentary mathematic operations. This is useful because it allows developers to specify, at a declarative level, relationships among beliefs. For instance, if an agent had beliefs about the cost of bread and butter, the developer could encode conditions such as if bread costs more than butter or if bread costs less than butter minus 10. With the original approach, this would not be possible without writing imperative code to compare the beliefs or belief arguments. Once commitments have been adopted, the agent commences the commitment management process. Various arguments are passed to the commitment when it is adopted, such as the time at which it should commence, to whom the commitment is made, and the maintenance condition of the commitment. An identifier is specified, which acts as a trigger for the plan or primitive action to be performed. In subsequent iterations of the control algorithm, the commitment is invoked subject to the arguments specified. The following is an example of an AFME commitment rule:

```
message(request, ?sender, removeData(?user))
>
deleteRecord(?user);
```

The truth of a belief sentence (text prior to the > symbol) is evaluated based upon the current beliefs of the agent. The result of the query process is either failure, in which case the belief sentence is evaluated to false or to a set of bindings that cause the belief sentence to be evaluated to true. The ? symbol represents a variable. In this example, if the agent adopts a belief that it has received a message from another agent to remove user data, it adopts a commitment to delete the record related to the user. At an imperative level, a preceptor, which is written in Java, monitors the message transport service, which contains a server thread that receives incoming messages. Once a message is received, it is added to a buffer in the service. Subsequently, the preceptor adds a belief, which is a first order structure Java class, to the agent's belief set. The interpreter periodically evaluates the belief set. If the conditions for a commitment are satisfied (that is, all of the beliefs prior to the > symbol in a rule have been adopted), either a plan is executed to achieve the commitment or a primitive action or actuator is fired. In this paper, we shall only consider primitive actions. When an actuator is created, it is associated with a symbolic trigger. In this case, a delete record actuator, written in Java, is associated with the trigger *stringdeleteRecord(?user)*. Once the commitment is activated, the ?user variable is passed to the actuator and the imperative code for deleting the file is executed.

An AFME platform comprises a scheduler, several platform services, and a collection of agents. The scheduler is responsible for the scheduling of agents to execute at periodic intervals. Rather than each agent creating a new thread when they begin operating, agents share a thread pool. A platform service, such as the message transport service, is a shared information space between agents on a local agent platform

that provides functionality that the agents can avail of through the use of actuators and perceptors.

AFME delivers support for the creation of agents that follow a sense-deliberate-act cycle. The control algorithm performs four functions. First, preceptors are fired and beliefs are updated. Second, the agent's desired states are identified. A subset of desires (new intentions) is then chosen, and added to the agent's commitment set. It should be noted that if the agent has older commitments, which are of lower importance, they will be dropped if there is not enough resources available to execute all commitments. This is handled through the knapsack procedure. Fourth, depending on the nature of the agent's commitments, various actuators are fired.

A. Static Architecture

In this application, the nodes commission a localization algorithm in order to identify a painting's position. Originally in the Octopus system the user had to monitor and control the network through the user interface. In the Agent Octopus system, some of the rudimentary tasks formerly performed by the user, are automated through the use of an agent.

In the static agent architecture shown in Figure 3, an agent resides within an Octopus plugin. The agent monitors the network state and alters its parameters by broadcasting messages to all nodes. For instance, if there are a large number of users or paintings within the gallery, the agent will lower the sampling rate so as to minimize message loss due to collisions and obstacles. This increases the longevity of the network in that fewer messages are retransmitted. Such WSN are inherently energy constrained systems depending typically upon a battery source. It also has an impact on the quality of service, however, the agent must ensure that the sampling rate is lowered as the number of users decreases.

Fig. 3. Static Agent Architecture

In terms of the duty cycle, our framework enables sensor nodes to sleep for the majority of the time in order to conserve energy. The radio duty cycle of a node is the portion of time during which the radio is awake. The choice of radio duty cycle impacts on correct data delivery, battery lifetime, and packet latency in the network. At present, the agent sets the radio duty cycle to ensure correct data delivery, leaving battery lifetime and packet latency as optimizations for future work.

With the static approach, all information related to painting is transferred to the user's device when they enter the gallery.

As the user walks around, the device communicates with nodes located on the painting frames to determine the user's location and the painting identification number. Once the number has been obtained, the phone plays a short audio clip related to the painting.

B. Mobile Architecture

Figure 4 illustrates the mobile agent architecture. In the mobile agent architecture, the agent within the Octopus plugin creates a user agent when a user enters the gallery. The user agent subsequently migrates to the user's mobile device. In this approach, the painting information is obtained in an ad hoc manner. Agents reside on Sun SPOT motes, which are attached to the painting frames. The same localization algorithm, as in the static approach, is used to determine the painting that the user is in the vicinity of. Once the user agent is aware of the painting ID, it initiates a conversation with the painting agent and requests an audio clip for the painting, which it subsequently downloads. This reduces the persistent storage requirements of the user's device in that only relevant clips are obtained. That is, if the user does not view a painting, they do not obtain the clip for that picture. Additionally, if a painting must be moved, for whatever reason, the system will dynamically adapt.

Fig. 4. Mobile Agent Architecture

As the user is viewing the painting, the agent monitors the amount of time spent there. If the agent identifies that the user has spent a long period of time at a particular artist's work, it will recommend other paintings by the artist and other artists from the same school. Additionally, it will provide the user with directions as to how to get to the other exhibits. For instance, if the user is viewing a painting by Claude Monnet, works by Pierre-Auguste Renoir will be recommended, as he was also a French Impressionist. The system is able to provide such recommendations because it obtains ontology information from the painting agents, such as the artist's school, style etc., which it uses to infer user preferences.

As the user is walking around the gallery, the user agent records all exhibits visited in addition to the time at which they were viewed. When the user leaves the gallery, the agent migrates back to the Octopus host machine; from there it updates a statistics database with this information. An additional agent monitors the database. It identifies congestion

problems, such as having a large number of people in a small room at the same time, and makes suggestions to the Octopus user as to alternative locations for paintings. So, for instance, if the Mona Lisa and Starry Night were placed in the same room, it would recommend that one of them be put somewhere else or, perhaps, that they be put in a bigger room. The database agent updates the locations of the paintings if they have been moved. The user does not have to enter such information manually. It is received through the use of the localization algorithm.

The mobile agent architecture has a number of advantages over the static agent approach. It provides the user with additional functionality and hence a better quality of service. The primary drawback is that it requires the execution of an agent on the user's phone. This obviously has an overhead in terms of battery life, memory, and processing cycles. These drawbacks are offset, however, by the fact that the agent only downloads information it believes is relevant to the user. This saves both power resources and storage space.

VIII. CONCLUSIONS

This paper has considered the issue of imbuing wireless sensor networks with autonomic capabilities. It has advocated multi-agent systems as an appropriate technological solution. Specifically Octopus a state of the art software tool for the monitoring, control and visualization of wireless sensor networks is considered. In particular the paper describes how this tool was reengineered to accommodate a MAS approach. Two distinct architectures are described those of the static and mobile architecture. The paper is novel and pioneering in that it represents the first documented attempt to deploy MAS techniques in the self management and self regulation of wireless sensor networks a particularly challenged environment in terms of its loss, lack of robustness, and being highly constrained in terms of hardware, memory and energy.

IX. ACKNOWLEDGEMENTS

The authors would like to acknowledge the kind support of (SFI) under grant 07/CE/I1147

REFERENCES

- [1] M. Berger, S. Rusitschka, D. Toropov, M. Watzke, and M. Schlichte, *The Development of the Lightweight Extensible Agent Platform*, EXP in Search of Innovation **3** (2003), no. 3, 32–41.
- [2] C.-L. Fok, G.-C. Roman, and C. Lu, *Rapid development and flexible deployment of adaptive wireless sensor network applications*, Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'05), IEEE, June 2005, pp. 653–662.
- [3] P.T. O'Hare G.M.P. O'Hare and T.D. Lowen, *Far and a way: Context sensitive service delivery through mobile lightweight pda hosted agents*, Proceedings of 15th International Florida Artificial Intelligence (FLAIRS) Conference. AAAI Press. May 14th-16th, Pensacola, Florida (2002).
- [4] Jonathan W. Hui and David Culler, *The dynamic behavior of a data dissemination protocol for network programming at scale*, SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems (2004), 81–94.
- [5] Sana Khaliq, Sana Farooq, Hafiz Farooq Ahmad, Hiroki Suguri, and Arshad Ali, *Sage-lite: An architecture and implementation of light weight multiagent system*, ISADS **0** (2007), 239–244.

- [6] Fernando Koch, John-Jules Meyer, Frank Dignum, and Iyad Rahwan, *Programming Deliberative Agents for Mobile Services: the 3APL-M Platform*, AAMAS'05 Workshop on Programming Multi-Agent Systems (ProMAS05) (2005).
- [7] P. Levis and D. Culler, *Mate: A tiny virtual machine for sensor networks*, In proceedings of ACM SIGOPS Operating Systems Review (2003).
- [8] T.D. Lowen, G.M.P. O'Hare, and P.T. O'Hare, *Mobile agents point the way: Context sensitive service delivery through mobile lightweight agents*, Ed. C. Castelfranchi C. & Johnson W.L., Proceedings of First International Joint Conference on Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2002), Bologna, Italy (2005).
- [9] D. Marsh, R. Tynan, D. O'Kane, and G.M.P. O'Hare, *Autonomic wireless sensor networks*, Engineering Applications of Artificial Intelligence Journal **17** (2004), no. 5, 741–748.
- [10] C. Muldoon, G.M.P. O'Hare, R. Collier, and M.J. O'Grady, *Agent factory micro edition: A framework for ambient applications*, In Proceedings of Intelligent Agents in Computing Systems (IACS 2) Workshop held in Conjunction with International Conference on Computational Science (ICCS) 2006 Reading, UK, May 28th-31st, 2006 Lecture Notes in Computer Science (LNCS) **3993** (2006), 335–353.
- [11] M.J. O'Grady and G.M.P. O'Hare, *Gullivers's genie: Agency, mobility, adaptivity*, Computers & Graphics Journal, Special Issue on Pervasive Computing and Ambient Intelligence, Mobility, Ubiquity and Wearables Get Together **28** (2005), no. 5, 677–689.
- [12] M.J. O'Grady and G.M.P. O'Hare, *Mobile devices and intelligent agents-towards a new generation of applications and services*, Special Issue on Intelligent Embedded Agents, Journal of Information Sciences **171** (2005), no. 4, 335–353.
- [13] A.G. Ruzzelli, R. Jurdak, M. Dragone, A. Barbirato, and G.M.P. O'Hare, *Octopus: A dashboard for sensor network visual control*, ACM Mobicom'08, demo paper, Sept 14–19, 2008 San Francisco, CA, USA (2008).
- [14] UBERkeley, *Mviz visualization tool*, <http://www.tinyos.net/tinyos-2.x/apps/MViz/>, 2006.
- [15] UCBerkeley, *Surge demo for the multihop sensor networks*, <http://firebug.sourceforge.net/surgepics.html>.
- [16] Megan Wachs, Jung Il Choi, Jung Woo Lee, Kannan Srinivasan, Zhe Chen, Mayank Jain, and Philip Levis, *Visibility: a new metric for protocol design*, SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems (2007), 73–86.
- [17] William Wright and Dana Moore, *Design considerations for multiagent systems on very small platforms*, AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems (New York, NY, USA), ACM Press, 2003, pp. 1160–1161.