2010 Seventh International Conference on Information Technology

QARI: Quality Aware Software Deployment for Wireless Sensor Networks

Wouter Horré, Sam Michiels, Wouter Joosen
IBBT-Distrinet
K.U.Leuven
3001 Leuven, Belgium
{*firstname.lastname*}*@cs.kuleuven.be*

Danny Hughes
Computer Science and Software Engineering
Xi'an Jiaotong-Liverpool University
215123 Suzhou, China
daniel.hughes@xjtlu.edu.cn

***Abstract*—If we are to deploy sensor applications in a realistic business context, we must provide innovative middleware services to control and enforce required system behavior; in order to correctly interpret collected temperature data, for example, sensor applications require guarantees about minimal coverage and the number of available sensors. The extreme dynamism, scale and unreliability of wireless sensor networks represent major challenges in contemporary software management. This paper presents QARI, a middleware service for decentralized and quality aware software deployment, which offers a simple yet flexible way to define, enforce, and maintain software deployment specifications. We have evaluated QARI on the LooCI component model and the SunSPOT platform; results confirm that QARI enables quality aware software deployment for a single as well as multiple applications, and even in the presence of node failure and mobility.**

Keywords: fault tolerance, large scale networks, management, middleware, quality aware

I. Introduction

Wireless Sensor Networks (WSNs) are inherently dynamic, unreliable and large in scale. In addition, WSNs are evolving towards interconnected, multi-purpose sensing infrastructure that is expected to host multiple applications. These applications may be deployed by diverse actors, who make use of sensor network infrastructure in return for direct payment or reciprocal application hosting.

Middleware is traditionally used to facilitate software management of such interconnected, multi-purpose network infrastructure; the complexity of WSNs is orders of magnitude greater than traditional distributed systems and requires sophisticated software management support. In previous work, we have presented (1) a novel component and binding model for resource-constrained sensor devices [1] and (2) a hierarchical, adaptive, graph-based approach to supporting reconfiguration in WSNs [2]. In this paper, we present QARI (Quality Aware Reconfiguration Infrastructure), a middleware service for decentralized, quality aware software deployment.

The characteristics of interconnected, multi-purpose WSNs make it difficult, if not impossible for a central entity to perform reliable and efficient software deployment throughout the whole (sensing) infrastructure. This is in part because (1) no central entity can have perfect knowledge of system state, and (2) traditional deployment semantics do not consider WSN-specific characteristics.

The decentralized approach of QARI eliminates the need for global knowledge of system state and allows QARI to exploit WSN-specific characteristics in the deployment process. In QARI, deployment is delegated to local management entities near the deployment target. Developers specify the required quality using a simple, yet flexible *quality aware deployment specification*. The local management entities use this specification to inform the efficient deployment of software, based upon locally gathered context data and the specific characteristics of the deployment target.

The current class of wireless SunSPOT (and alike) sensor devices [3] forms an ideal prototyping environment to evaluate the next-generation middleware services that are needed to support critical business scenarios (e.g. monitoring high-value pharmaceutical products throughout the supply chain). We have realized a prototype implementation of QARI for the LooCI component model and the SunSPOT platform. This prototype uses a simple coverage metric for specification of the required quality level. We evaluated the QARI approach using the prototype on both a physical testbed and a simulated sensor network.

In the remainder of this paper we first discuss related work (Section II). We then present and evaluate QARI, our decentralized, quality-aware deployment service (Section III and Section IV). Finally, Section V concludes and discusses directions for future work.

II. Related work

This section analyzes the state of the art in software deployment support for WSNs and discusses quality awareness in WSNs. Three categories of runtime support for deployment and reconfiguration can be distinguished [4]:

- **Monolithic:** replace all functionality during the update by re-flashing and re-starting the nodes.
- **Script-based:** change the behavior of previously deployed functionality by injecting lightweight scripts.
- **Modular:** replace coarse-grained units of functionality (modules) at run-time.

Deluge [5] is a reliable epidemic code dissemination protocol that is used to support monolithic flashing of a

978-0-7695-3984-3/10 $26.00 © 2010 IEEE
DOI 10.1109/ITNG.2010.132

network of TinyOS [6] motes. Deluge focuses upon achieving good network performance, providing high throughput and imposing minimal additional overhead due to control messages.

Replacing a complete code image using Deluge with only small changes to the behavior implies a large energy overhead for a small functionality change. Script-based approaches such as Maté [7] address this by supporting the injection of lightweight scripts that are interpreted on the sensor nodes to drive the execution of pre-deployed TinyOS functionality. DVM [8] combines this approach with the dynamic modules of SOS [9] to remove the pre-deployed limitation of Maté.

The Sun SPOT [3] and Sentilla Perk [10] WSN platforms are Java based and allow for modular reconfiguration via the dynamic deployment of MIDP 1.0 compliant Java applications. SOS [9] and Contiki [11] provide similar support for the deployment of binary application modules that are dynamically loaded into the operating system. While this is an improvement over monolithic approaches, the relationships between modules are opaque and may not be reconfigured.

On top of these runtime systems, reconfigurable component systems (such as OpenCOM [12] and LooCI [1]) provide additional support to change the composition of reconfigurable functional blocks (components). A combination of these runtime types might be used to achieve this: for example, DAViM [13] uses a modular runtime to update coarse grained units of functionality (components) and a script based approach to change the interaction between these components.

We aim to provide quality aware management of deployed functionality regardless of the deployment approach used.

The quality-of-service (surveillance) that can be provided by a particular WSN is commonly expressed in terms of k–coverage or related coverage metrics. Most approaches [14], [15] calculate coverage after deployment, propose node deployment heuristics or otherwise use coverage information in single application WSNs. We propose to exploit the knowledge of coverage requirements to optimize the software deployment process in shared WSN scenarios.

MiLAN [16] also addresses the topic of determining the optimal subset of sensors that is required to reach a desired quality-of-service level. MiLAN confirms that there are usually multiple subsets of sensors that can provide the required data, but with different quality-of-service properties. We believe that this knowledge can be exploited to improve and optimize the software deployment process in shared sensor network scenarios.

III. Quality aware software deployment for WSN

Figure 1 shows an overview of QARI, with Application Managers and Network Managers as key abstractions. Application Managers are the entities responsible for the management of one or more sensor network applications. Application Managers split their centralized deployment plans into smaller deployment specifications for decentralized processing by Network Managers [2]. These specifications include a description of the component to deploy as well as details of the desired quality level (expressed as an interval from the minimally required level to the preferred level). We therefore call them *quality aware deployment specifications*.

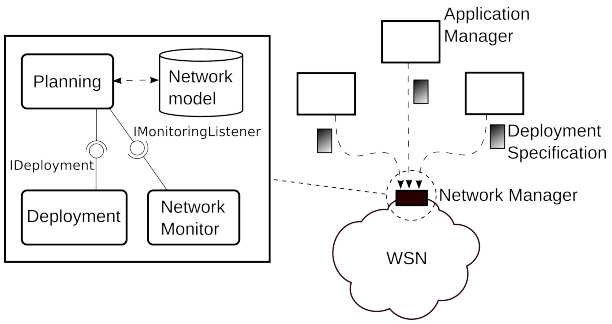

Figure 1: An overview of QARI, our approach for quality aware software deployment (right) and a high-level architecture of the Network Manager component (left).

The Network Manager, which is close to the WSN, is responsible for merging specifications into a single target specification for the software deployment of the WSN. It deploys and updates the software on the WSN to reach and maintain this target state.

This approach provides a clean separation of concerns. The definition of the quality requirements, which requires application specific knowledge, is handled by the Application Managers. The responsibility for achieving and maintaining these quality requirements, a task which requires local, network specific knowledge, is handled by the Network Manager.

A. Enacting and monitoring deployment

This section describes how network monitoring and software deployment functionality is realized by the *Network Monitor* and *Deployment* components. The *Planning* component is responsible for deployment planning and maintenance (see Section III-B). It uses the feedback from the *Network Monitor* and *Deployment* components to keep its *Network Model* up to date (see Figure 1).

The *Network Monitor* and *Deployment* components are the interface to the WSN and apart from their deployment and monitoring functionality, they perform two additional tasks: (1) they provide a point of virtual synchrony (i.e. they expose monitoring of and deployment to the WSN as operations on a synchronous system, although the WSN is asynchronous in nature) [17] and (2) they abstract WSN platform specific details. To perform their tasks they use platform specific knowledge, e.g. to select appropriate time-

outs for node failure detection, to select the appropriate deployment mechanism for a given deployment request, etc.

The *Network Monitor* uses platform specific knowledge and mechanisms for gathering information about the WSN. It implements a publish-subscribe interface over which this network information is disseminated to interested software entities (such as the *Planning* component). Remote entities that wish to be notified of changes in the WSN environment, should implement the *IMonitoringListener* interface and subscribe themselves at the *Network Monitor* which then pushes notifications of node failures, notifications of node arrivals and notifications of status changes of a deployed component.

The *Deployment* component provides an abstraction on top of one or more platform specific deployment mechanisms. The *Planning* component interacts with the *Deployment* component through the *IDeployment* interface that allows the *Planning* component to deploy, start, stop and undeploy a component—in the remainder of the paper, we use the term component for a piece of software that is independently deployable— on a given list of addresses. Each of the methods indicates in its return value for which addresses the action succeeded. The interface is called in synchronous fashion by the *Planning* component, thus it is up to the *Deployment* component to decide when an action must be considered unsuccessful (and thus provide virtual synchrony over the asynchronous WSN).

B. Deployment planning and maintenance

The *Planning* component of the Network Manager assigns components to sensor nodes based on the specifications it receives from the Application Managers. The calculation of these assignments uses local knowledge about the WSN. The *Planning* component also updates the assignments in reaction to significant changes in the WSN, which it is notified of through the *Network Monitor* component.

The specifications contain a description of the component that is requested—called component type in the remainder of this paper—and two quality levels: a minimal and a preferred level. These quality levels are expressed as a coverage metric. Future extensions might incorporate other quality metrics, such as communication delay or sensor data accuracy.

Initial deployment: When a component type is to be deployed for the first time, the *Planning* component first calculates an assignment that achieves the minimal coverage needed for the component type. After the component type has been deployed to this initial assignment, the assignment is updated to achieve the preferred coverage for the component type.

Combining specifications: The *Planning* component calculates only one assignment per component type. If there are multiple specifications for the same component type coming from different sources, the *Planning* component calculates an assignment for the component type that satisfies at least the minimal coverage of all specifications. If possible, the *Planning* component will try to satisfy all preferred coverages.

The possibility of sharing component deployments across multiple applications is a clear benefit of delegating deployment to the WSN edge. The individual Application Managers have no means to optimize deployments based on the requirements of other applications. However, since the Network Manager is responsible for enacting all deployments, it has the knowledge required to perform such optimization.

Failures: The *Planning* component reacts to node and component failures by calculating a new assignment. The new assignment is updated locally—i.e. only in the vicinity of the failed node—to repair the coverage while making as little changes as possible to the original assignment. This minimizes the disturbance to the network and the time needed to reach the targeted coverage again.

Node mobility: Node mobility is handled in a similar manner to node failure: the component assignment is updated locally around the area where the node left. If the node is not needed to ensure coverage in the area it moved to, the corresponding component will be removed from the node.

Network sharing: The *Planning* component uses local knowledge about the state of the WSN (gathered through the *Network Monitor*) to minimize the interference between applications. As discussed above, the *Planning* component combines multiple specifications for the same component type.

For different component types, two measures are taken to minimize the interference. First, the *Planning* component takes the state of a sensor node into account during the calculation of an assignment. One of the parameters which is accounted for is the number of components already deployed on a sensor node. This way, different components are spread as evenly as possible across the WSN.

Second, the *Planning* component schedules deployment actions so that at least the minimal coverage of other component types is maintained during the deployment. To realize this, the *Planning* component splits a deployment in multiple smaller deployments that have less effect on the network. That way, the minimal coverage of other component types is only broken if it is absolutely necessary.

IV. Evaluation

This section evaluates a prototype implementation of QARI. We evaluated the *Planning* component in two ways. First, we implemented a prototype of the *Network Monitor* and *Deployment* components for a WSN using the LooCI component model [1] and the Sun SPOT [3] WSN platform. We tested our approach using a test-bed of 9 standard Sun SPOT motes (180MHz ARM9 CPU, 512KB RAM,

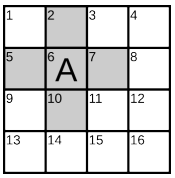

Figure 2: Coverage for Node A

SQUAWK VM 'BLUE' version).

The implementation of this prototype relies on the primitives offered by the LooCI component model. The *Network Monitor* component uses the LooCI introspection functionality. The *Deployment* component depends upon the LooCI deployment primitives, which themselves build upon the standard Sun SPOT deployment approach [3]. Currently this provides only unicast deployment, thus deployments to a group must be sequenced and executed in a one-by-one fashion. However, since the *IDeployment* interface is defined in terms of lists of nodes, the *Deployment* component could also be implemented for other platforms that support deployment to a group.

Second, we implemented a dummy version of the *Network Monitor* and *Deployment* components that simulate a grid of 10x10 sensor nodes. This allowed us to move beyond simple feasibility tests and conduct the analysis provided in the Sections IV-B to IV-D.

A. Case Study and Coverage Heuristic

We evaluated our approach in the context of a warehouse monitoring scenario. In this scenario, a storage company, STORAGE-CO, uses a WSN infrastructure to monitor the conditions in a temperature controlled warehouse. Nodes participating in the STORAGE-CO WSN are evenly distributed throughout the warehouse which is divided into 100 unique zones (9 zones for the physical testbed). Each node hosting a TEMP-SENSOR component has an effective radius for which it can provide temperature reading. For the purposes of our case study, we simplify the effective radius to the unique areas adjacent to the location of the node. Figure 2 illustrates our notion of coverage for Node A, which is deployed in unique area 6 and may be used to monitor the surrounding areas (2,5,7,10). Our simple coverage metric allows developers to specify a *minimal* and *preferred* number of nodes that should cover each unique area. For example, STORAGE-CO may wish to deploy TEMP-SENSOR components in such a way that every unique area is covered by at least one node and preferably two nodes.

The Network Manager *Planning* component uses a heuristic algorithm to calculate an assignment that satisfies the coverage requirements. The algorithm orders the areas based on the number of nodes that are able to cover the area. It then considers each of the areas in this order. For each area, the algorithm adds the *best* nodes to the assignment until the area is sufficiently covered. The *best* nodes are defined as the nodes that are not already in the assignment and have the best score associated. The score is calculated based on the number of currently uncovered areas the node covers, the number of components already running on the node and the reputation of the node. The reputation of a node is used to penalize nodes that have failed in the recent past.

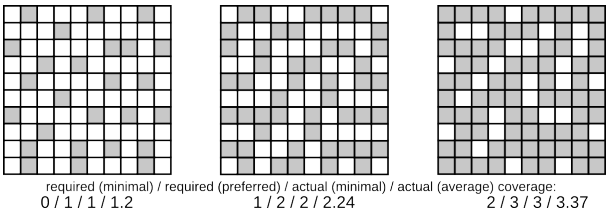

Figure 3: Calculated deployment patterns for a single component type.

If there is an assignment that satisfies the coverage requirements, the algorithm is always able to find it, since it keeps adding nodes until all areas are sufficiently covered. The algorithm doesn't always find the best assignment, but our experiments show that it always finds a good assignment with respect to the current state of the WSN.

B. Single application coverage

To demonstrate the feasibility and correctness of the coverage heuristic, we evaluated our approach by applying different coverage requirements for a single component type to our 10x10 simulated testbed. The experiments defined quality requirements by setting the following minimal/preferred coverage parameters: 0/1, 1/2 and 2/3. We executed each experiment 10 times, each of which yielded the same deployment pattern.

The realized coverage patterns for each experiment are shown in Figure 3. In the case of 3/4-coverage, there is no assignment that provides 4-coverage since the corner areas have only three neighboring nodes. In such cases, our system generates an "as good as possible" assignment: all other areas still maintain sufficient coverage to meet the requirements.

Once the areas in the grid are sorted, the prototype calculates the optimal deployment pattern in $O(n)$, where n is the number of areas. The calculation requires on average 8.3 ms for a 10x10 grid (Sun Java 1.6.0_16, Pentium D 3.2GHz, 1GB RAM). Transferring the TEMP-SENSOR component requires 1692 bytes to be sent to a selected node, installing and starting it takes 9 seconds, which considerably improves the throughput of manually updating each sensor node in the field (even in the case of this relatively small test case).

C. Multiple application coverage

QARI enables not only to deploy components for a single application (i.e. the same component type on every selected node), it also allows for sharing the WSN infrastructure by multiple applications. We have verified this by deploying both a TEMP-SENSOR and a HUMIDITY-SENSOR component.

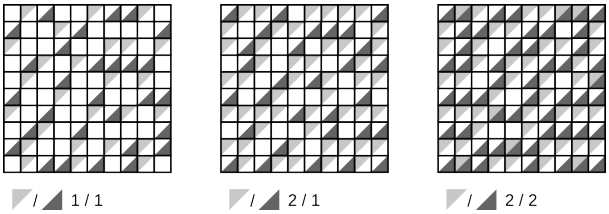

Figure 4: Calculated deployment patterns for a shared network: TEMP-SENSOR (light gray) and HUMIDITY-SENSOR (dark gray). The preferred coverage for each deployment is also indicated in the figure.

Figure 4 shows the deployment patterns of this test case; again, three experiments were defined with different minimal/preferred coverage requirements: (1) single coverage for both the TEMP-SENSOR and the HUMIDITY-SENSOR component, (2) double coverage for the TEMP-SENSOR component, and single for the HUMIDITY-SENSOR component, and (3) double coverage for both applications.

As illustrated, QARI is able to maximally spread different applications over the WSN, while not breaking the coverage requirements of other applications. Only when no alternative deployment patterns exist (Figure 4 right side), QARI will deploy multiple applications on a single node.

D. Resilience to node failure and mobility

Having sketched the feasibility of QARI to achieve coverage requirements in an automated and correct way, we finally show that QARI is able to maintain quality requirements in the context of unreliable network conditions, i.e. deployment failures, node failures and node mobility.

We have reevaluated the single application test case (cfr. Section IV-B), and randomly injected failures during component deployment. QARI succeeded in establishing a usable deployment pattern; obviously, in case of failures, the generated deployment patterns deviate from the patterns in Figure 3 (worse in terms of the number of nodes involved).

In addition, we evaluated the capability of QARI to maintain coverage requirements in case of node failures. We evaluated this by randomly removing nodes from and reinserting nodes into the testbed and verifying the system's reaction. As expected, QARI was able to (1) detect that coverage was no longer optimal and (2) reapply the heuristic to start a new deployment cycle. As long as enough nodes were alive, QARI was able to re-establish a coverage pattern that satisfies the requirements.

QARI minimizes the impact of a node failure by locally—in the vicinity of the failed node—repairing an assignment instead of globally re-assigning component types to nodes. For each of the 10 experiment runs, recovering from a failure that affected an existing assignment took at most 3 repair actions; on average only 1.69 repair actions were needed.

Figure 5 shows the minimal and average coverage over time for one of these runs. The coverage requirements for this experiment were 1 and 2 (minimal and preferred).

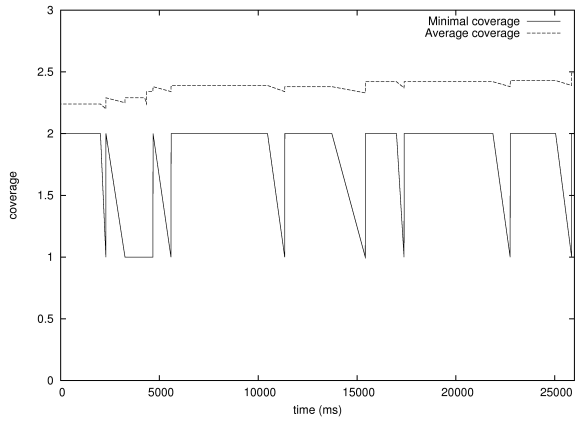

Figure 5: Minimal and average coverage over time in the presence of node failures.

QARI is able to keep the minimal coverage that is actually achieved between these bounds. It can be seen from the figure that at around 3s, the minimal coverage achieved is lower than the preferred coverage for about 1.5s. This is because not enough nodes were available to realize the preferred coverage for some areas. QARI recovers from this situation when enough nodes reappear. Even if it is not possible to meet the requirements—preferred coverage or in the worst case minimal coverage—, QARI tries to do "as good as possible", resulting in a good level of average coverage despite the fact that the coverage of some areas is lower than preferred.

Node mobility is handled in a similar way to node failure: a mobile node disappears in one area (cfr. node failure) and appears in another (cfr. node reappearing).

E. Discussion

In the previous sections we illustrated the feasibility of our approach and presented a quantitative evaluation of our prototype implementation. Given the limited simulation environment, we were not able to perform quantitative scalability tests. However, since the *Network Manager* runs on a less resource constrained device at the edge of the WSN and our measurements in Section IV-B indicate a reasonable performance, the scalability of QARI is expected to be bounded by the scalability of the underlying deployment mechanism.

The number of node failures that QARI can tolerate without breaking the coverage requirements will of course depend on the level of node redundancy in the network. Also the failure pattern will influence the failure rate that can be tolerated. If the failed nodes are concentrated in a certain area of the network, less failures can be tolerated than when the failed nodes are spread over the whole network. If the number of component types on the network is larger, the number of possible assignments will also be limited by the capabilities of the nodes to run multiple components simultaneously. This will also have an effect on the number

of failed nodes that can be tolerated.

V. Conclusions and future work

In this paper, we presented QARI, a decentralized, quality aware deployment service for inclusion in management middleware for interconnected, multi-purpose sensing infrastructure. QARI offers developers a simple, yet flexible specification of the desired quality. The realization of these specifications is delegated to an entity close to the sensor network. This entity uses locally gathered contextual data to achieve and maintain the required quality.

We have realized a prototype implementation of QARI for the LooCI component model and the SunSPOT platform. We evaluated QARI using this prototype and a simple simulation environment.

We see several opportunities for future work. First, we intend to extend the deployment specifications to include a more rich specification of the desired quality to allow QARI to optimize deployment in even more cases. Second, we consider it useful to further research the influence of the scoring used to select the *best* nodes in our heuristic. Finally, we plan to extend the simulation environment to facilitate more quantitative analysis and scalability testing.

VI. Acknowledgments

Wouter Horré is a PhD fellow of the Research Foundation - Flanders (FWO). This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, by the Research Fund K.U.Leuven, and by the Flemish agency for Innovation by Science and Technology (IWT) in the context of the IWT-SBO-STADiUM project No. 80037.

References

[1] D. Hughes, K. Thoelen, W. Horré, N. Matthys, J. Del Cid, S. Michiels, C. Huygens, and W. Joosen, "LooCI: a loosely-coupled component infrastructure for networked embedded systems," in *Proc. 7th Intl. Conf. on Advances in Mobile Computing and Multimedia (MoMM2009)*, Kuala Lumpur, Malaysia, Dec. 2009.

[2] W. Horré, K. Lee, D. Hughes, S. Michiels, and W. Joosen, "A graph based approach to supporting reconfiguration in wireless sensor networks," in *Proc. 1st Workshop on Appl. of Graph Theory in Wireless Ad hoc Networks and Sensor Networks*, Dec. 2009.

[3] Sun Microsystems, "Small Programmable Object Technology, "Inspiring Java developers to create a whole new breed of devices and technologies – and accelerating the growth of the Internet of Things"," Avail. online at: http://www.sunspotworld.com/vision.html, July 2009.

[4] C.-C. Han, R. Kumar, R. Shea, and M. Srivastava, "Sensor network software update management: a survey," *Int. J. Netw. Manag.*, vol. 15, no. 4, pp. 283–294, 2005.

[5] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *SenSys '04: Proc. 2nd intl. conf. on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2004, pp. 81–94.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *SIGPLAN Not.*, vol. 35, no. 11, pp. 93–104, 2000.

[7] P. Levis, D. Gay, and D. Culler, "Active sensor networks," in *Proc. 2nd USENIX/ACM Symp. on Network Systems Design and Implementation (NSDI)*, May 2005.

[8] R. Balani, C.-C. Han, R. K. Rengaswamy, I. Tsigkogiannis, and M. Srivastava, "Multi-level software reconfiguration for sensor networks," in *ACM Conf. on Embedded Systems Software (EMSOFT)*, October 2006.

[9] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes," in *MobiSys '05: Proc. 3rd intl. conf. on Mobile systems, applications, and services*. New York, NY, USA: ACM Press, 2005, pp. 163–176.

[10] Sentilla Corporation, "Perk platform frequently asked questions," Avail. online at: http://www.sentilla.com/perk_faq.html, July 2009.

[11] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, "Run-time dynamic linking for reprogramming wireless sensor networks," in *Sensys'06: Proc. 4th ACM Conf. on Embedded networked sensor systems*, Boulder, Colorado, USA, Nov. 2006.

[12] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan, "A generic component model for building systems software," *ACM Trans. Comput. Syst.*, vol. 26, no. 1, pp. 1–42, 2008.

[13] W. Horré, S. Michiels, W. Joosen, and P. Verbaeten, "Davim: Adaptable middleware for sensor networks," *IEEE Distrib. Syst. Online*, vol. 9, no. 1, January 2008.

[14] S. Kumar, T. H. Lai, and J. Balogh, "On k-coverage in a mostly sleeping sensor network," *Wireless Networks*, vol. 14, no. 3, pp. 277–294, 2008.

[15] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava, "Coverage problems in wireless ad-hoc sensor networks," in *INFOCOM 2001. Proc. 20th Ann. Joint Conf. of the IEEE Comput. and Comm. Soc.*, vol. 3, 2001, pp. 1380–1387 vol.3.

[16] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications." *IEEE/ACM Trans. Netw.*, vol. 18, no. 1, pp. 6–14, 2004.

[17] A. Schiper, K. Birman, and P. Stephenson, "Lightweight causal and atomic group multicast," *ACM Trans. Comput. Syst.*, vol. 9, no. 3, pp. 272–314, 1991.