50 ways to build your application: A Survey of Middleware and Systems for Wireless Sensor Networks

Ioannis Chatzigiannakis, Georgios Mylonas and Sotiris Nikoletseas
Research Academic Computer Technology Institute (CTI) and University of Patras, Greece
{ichatz, mylonasg, nikole}@cti.gr

Abstract

In this paper, we survey the current state-of-the-art in middleware and systems for Wireless Sensor Networks (WSN). We provide a discussion on the definition of WSN middleware, design issues associated with it, and the taxonomies commonly used to categorize it. We also present a categorization of a number of such middleware platforms, using middleware functionalities and challenges which we think will play a crucial role in developing software for WSN in the near future. Finally, we provide a short discussion on WSN middleware trends.

1. Introduction

As wireless sensor networks are becoming increasingly popular in the research community and their costs are steadily going down the last few years, allowing more researchers to build their own sensor network testbeds, the number of available software environments specifically for these networks also keeps increasing at a steady pace. Although we haven't yet reached the point of each sensor node being available for only a few euros/dollars, things have changed greatly for the better in that respect since the appearance of the first sensor nodes.

This came along with the proposal of using WSN for a multitude of applications, that was constantly updated in the recent years. Although many of these applications have quite unique characteristics, they still share many common features, and it is also interesting to see how the vision for such applications has progressed through time. Generally, we can divide WSN applications proposed and developed in three "waves":

- The first applications that appeared (target tracking, room surveillance, environmental tracking, etc.) regarded a number of sensor nodes belonging to a single sensor network. The emphasis was mostly on providing algorithms and protocols for efficient power management, routing, localization, etc.
- The (recent) second wave of applications provides added capabilities and services, and to some extent utilizes a number of sensor gateways (rather than a single one), thus setting new standards in terms of size and heterogeneity, also allowing multiple sensor networks to be administered as one.
- The third wave, that currently is starting to take place, relates to Internet-scale sensing applications. Unlike previous WSN applications, interfacing between/with different networks and applications is gaining importance. This kind of applications is related to what some call "the Internet of things".

This progression of WSN applications, along with other important factors, such as the lack of standardization in hardware and software, led to:

i) Multitude of programming paradigms to fill in the gap: A large number of different approaches concerning middleware and network management environments for sensor networks has been proposed the last few years. Between choosing any of these different approaches there is a certain trade off between ease of use, expressiveness (regarding available network functionality) and usage of the nodes' resources (energy, memory, etc.).
ii) Multitude of functionalities and services these approaches provide: gradually the number of services WSN software provide has increased, due to additional application requirements. It is becoming more common to talk about heterogeneity, quality of service, security, trust, standardized interfaces, apart from efficient resource management, data delivery success rates, etc., nowadays. Also, actuators and wired sensor networks are gaining importance.
iii) Multitude of levels in which these software environments function: More recent WSN applications may require different architectures than the ones used in the first implementations of such applications. For example, tiered networks, that use different kinds of sensor nodes, in terms of available resources creating additional hierarchical levels, are proposed to be used in a number of cases. Also, Internet-scale sensing implies software running also on top of WSN nodes and gateways.

The fact is that, still, WSN remain relatively difficult to program and integrate into a commercial platform, and have some road ahead of them. Naturally, it is our belief

that this is a crucial factor for the eventual success of WSN in general.

As the scope of WSN applications continues to grow, in order to harvest their potential benefits, and ultimately become a part of everyday life, a vision needs to be realized which goes well beyond incremental and disconnected improvements of diverse (and often incompatible) implementations. It is our belief that they will have an important role in what is called *global computing*. To accomplish this, software for WSN applications must adapt to the new requirements posed. Global Computing is about computational infrastructures available globally, able to provide uniform services with variable guarantees for communication, co-operation and mobility, resource usage, security policies and mechanisms, etc., with particular regard to exploiting their universal scale and the programmability of their services. The seeming integration of the Internet and sensor domains and its benefits can be better described e.g., by the concept of a *Sensor-oogle* or a *Google Earth Sense*. A whole new layer of information can be used for offering new services to existing or new systems. Imagine a version of Google offering search capabilities for sensing services or a Google Earth with live sensing data coverage per square mile.

Future systems that will exploit WSN infrastructures could be powerful distributed search engines that operate without any central database, but instead use geographically-separate networked computers to index documents locally within each node, using whichever processing resources are available. Of course, on this level, what becomes the most important is the data itself from all the available sensor networks. Managing, analyzing and understanding such data, through the use of suitable tools, poses new unique challenges. A recent discussion on properties of such systems can be found in [12].

There is some previous related work surveying middleware for wireless sensor networks, namely [22],[21],[39],[28]. A more detailed comparison of a number of WSN middleware systems is presented in [37]. The present paper in some ways complements these works and provides another perspective to the categorization of such software, along with an extended bibliography containing recent developments in middleware for WSN.

2. What is WSN middleware?

Before going into further detail in describing the taxonomies and the state-of-the-art in middleware for wireless sensor networks, it is reasonable that we provide a description of what we refer to as middleware in the case of WSN. This is not a simple issue, since, as stated in the introduction, the scope of WSN is changing and growing constantly the last few years.

The term middleware itself can sometimes be a buzzword, meaning different things, depending on the computing field its used for and the scope of the person referring to it. In distributed systems, middleware is usually defined as software that lies between the operating system and applications running on each node of the system. Generally, middleware is expected to hide the internal workings and the heterogeneity of the system, providing standard interfaces, abstractions and a set of services, that depends largely on the application. Regarding abstractions offered by WSN middleware, in [28] it is stated that "the middleware layer is equivalent to the presentation layer in the OSI model", although this description does not cover all cases.

An interesting definition of WSN middleware is given in [39]. In short, middleware has to provide support for the development, maintenance, deployment and execution of WSN application. It is also stated that "the scope of middleware for WSN is not restricted to the sensor network alone, but also covers devices and networks connected to the WSN". This perspective of WSN middleware is neglected in most of the work surveying such software, if not even taken into consideration. Essentially, someone could argue that since WSN are application-specific systems all software interfacing such networks with more traditional systems can be classified as middleware, since it is a means of developing sensing applications.

For this reason, since we agree with such a definition of WSN middleware, we chose to include in this survey and classify it as middleware even software that does not act on a sensor network level, but rather as a means e.g., to provide transparency between a number of discrete WSN. Also, when we refer to WSN systems, we include software that can act as a way for developers to build a complete WSN application, top-down, regardless of the overall capabilities such systems provide to the final user.

3. Middleware design issues and requirements

Regarding the requirements and other issues that govern the design and implementation of WSN middleware, it should be noted that the situation is still rather fluid, as stated in the introduction. There is still no wide consensus over things such as hardware platforms and operating systems design, and there is also the constantly expanding WSN application field. Also, the vision of WSN itself is changing, as applied research in the field progresses. From the initial vision of "hundreds or tens of thousands of sensor nodes in each sensor network", we are moving to more realistic sensor node numbers (in the scale of thousands currently), or to a number of sensor networks with a rather small number of sensors in each such network. This is not irrelevant to the practical difficulties programmers face while developing WSN applications.

Having these parameters in mind, there are some issues and requirements that are common in most of the WSN middleware platforms proposed so far, and the things that change are the abstractions provided to the developers, as a result of the different approaches taken. *Efficient resource management* is the definitive issue in designing

software for WSN – restrictions in processing capabilities, energy and communication bandwidth, simply put, cannot be easily ignored in WSN. Then, *fault tolerance* and *adaptability* are the other factors traditionally used as design issues for WSN software. *Data-centric tasking* is another important factor, but as tiered and heterogeneous sensor networks become more important, so do the capabilities of specific network nodes, which somehow comes in contrast to the data-centric paradigm. Data aggregation was evaluated as an important factor, but in recent years there have been proposals of moving complexity to upper network tiers (i.e., nodes with more processing capabilities). *Security* and *trust*, *heterogeneity* and *scalability* are four very important factors in developing future WSN applications, even more so in the vision of massive-scale sensing applications.

We have selected a number of these issues and requirements as the basis of a categorization of current WSN middleware platforms, in Section 5. A more detailed description of these selected criteria, along with a number of examples, is provided in that section.

4. Taxonomies for WSN middleware

A popular taxonomy of WSN platforms is the following:

- *sensor databases*,
- *virtual machines*,
- *agent-based* approaches,
- *network management tools*,
- other approaches.

The central idea in *sensor databases* (e.g., [30],[6]) is to provide an SQL-like interface to the programmer that makes the sensor network *look like* a *DBMS*. *Virtual machines* (e.g., [31]), on the other hand, provide sensor programming primitives in an assembly-like language. Users compose scripts that are uploaded to the network nodes and are executed in a virtual machine running in each of these nodes. In *agent-based* approaches ([15]) the software provides abstractions for computing tasks to the programmer (like in virtual machines), such as events firing, complex sensing tasks, etc., that are used to program mobile agents, that can move from node to node. In this way, complex tasks can be realized in situations where each node may have to run different software, depending on what tasks it has to carry out. Finally, *network management tools* (e.g., [32],[41]) provide the user capabilities to visualize results from a sensor network, combined with a data logger that runs on a gateway. There is not much programming flexibility provided in these tools, i.e., there is only a number of certain functionalities available. Motes in the network poll their sensors at a sampling rate specified by the user and send them to the gateway using a multi-hop protocol; readings from the network are usually stored in a relational database for further processing. All these environments can be combined or come with tools for remote reprogramming of the network nodes to ease development and deployment. There are also some other approaches, like publish/subscribe ([11]) or tuple-based ([8]), but, currently, these are not as popular as the aforementioned ones, or cover specific application cases.

Another, more general, taxonomy, included in [19], is categorizing WSN middleware into two classes, based on their programming scope, i.e., the way they aid developers in implementing WSN applications:

- *Programming abstractions*: they provide programmers with abstractions for viewing sensor nodes and sensor data, or the network as a whole.
- *Programming support*: these provide mechanisms and services for making programming WSN simpler, e.g., high-level composition of applications, remote code update, remote debugging, etc.

Each of these two classes is further subdivided into a number of subclasses, according to the specifics of the approach followed in each case. Programming abstractions, e.g., depending on whether they are used to abstract the behavior of the whole sensor network or only node neighborhoods, are characterized as global and local behavior programming abstractions, respectively. Further subdivisions are possible, based on a number of other criteria, such as consideration of each node separately, etc.

A mixed approach of the previous two taxonomies is presented in [21], where the categories of the first taxonomy (sensor databases, etc.) are all characterized as programming support middleware. Also, virtual machines in [22] are dismissed as being not middleware, since current implementations do not provide enough capabilities, but it is our belief that they do fit in our description of WSN middleware. In other words, there is an active debate regarding the categorization of WSN software.

A problem with these taxonomies is that they do not cater for middleware that is actually running, partly or entirely, on top of the sensor network level (i.e., not in the sensor network nodes). Also, in some cases, it is not easy to place certain WSN middleware in just one of the above taxonomy classes. Lastly, one other aspect that these taxonomies do not cover, albeit not as important, is whether the middleware in question follows a completely distributed, hierarchical or more centralized software architecture model. Therefore, it is our belief that these taxonomies, though quite general and successful, are not entirely adequate for describing the current trends in WSN middleware.

5. A categorization of WSN middleware and systems based on present and future challenges

In this section, we make an attempt to provide a categorization of the current systems, based on a number of other criteria than those discussed above. It is important to note that lately there is a shift from the single sensor network domain concept used in the earlier approaches, to a multiple sensor network application domain, so we must choose criteria that reflect this philosophy. We include a number of the most representative platforms for WSNs as examples for each criterion. Although not exhaustive, it our belief that such a list is quite indicative of the current situation in WSN software. We include only a short reference to each of these platforms.

5.1. Purpose – Application development characteristics and extendability

The question here is whether to provide a number of implemented features and functionalities to the final users, through which a certain sensor network application is implemented (i.e., have *limited extendability*), or to provide a set of tools and APIs to implement such an application and interface it to other environments (*application development*). In many cases a mixed approach is followed.

It could be said that platforms like MoteView [32] and ScatterViewer [41] are on the side of tools with rather limited extendability, providing a set of functionalities that cannot be easily extended or their architecture is too tightly coupled, while platforms like SenseWeb [40], P2PBridge [24], Hourglass [47], GSN [1], are on the side of application development tools, providing environments general enough to help developing large-scale WSNs, each one in its own way. SNACK [18] and DSN [4] provide users with logical abstractions-languages in order to compose new software application components, as an alternative to the widely used nesC programming language. In short, they provide higher-level programming abstractions and mechanisms, but not as high-level as, e.g., TinyDB. Most of the other platforms fall somewhere in the middle, since most provide some degree of flexibility into building custom WSN applications. There are also some solutions like Abstract Regions [49], Hood [46], and Neidas [10] that provide logical abstractions of network neighborhoods to ease local node interaction and data sharing. Kairos [19] provides programming abstractions, but on a global network behavior scope (regarding specific nodes not necessarily in the same neighborhood), implemented as an extension to Python. Tenet [13] defines a sensor tasking language, along with providing a task library and a tiered architecture, where more demanding tasks are executed at master nodes, which are more capable than the rest of the nodes.

To some extent, a complete WSN middleware solution should be able to provide an application development environment, simplifying the development of custom WSN applications, without sacrificing generality of use. This means that network administrators will be able to leverage existing solutions for sensor networks along with creating new ones. However, there is of course room for solutions providing limited services to the programmer.

5.2. Adaptability – Network management or self-organizing

The next question we consider is whether the software in question provides self-organizing features or if the network parameters can or have to be specified by the network administrator. E.g., do the network nodes organize in multiple cluster levels and adjust to the network conditions automatically, does the network administrator have any control over this situation, does the administrator have to tune specific network parameters instead of relying on the network node software, etc.

Most existing sensor network applications provide some self-organizing and network management features. Most self-organizing features revolve around providing services like time synchronization (e.g., TinyDB [30]), routing protocols (e.g., the ZigBee stack), but in general the network administrator has little control over such features. On the other hand, some applications (e.g., MoteView [32]) require the intervention of the network administrator to carry out even simple tasks. There is also the issue of whether the policies providing these dynamic behavior, can be dynamically updated themselves. RUNES [5] explicitly caters for such issues allowing software components to be updated on-the-fly. Impala [29] is also able to update on run-time and adapt to changing network conditions and hardware failures.

We believe that a sensor network application environment should provide both self-organizing features and the flexibility to tune them in according to the needs of the administrator and the application itself.

5.3. Fault tolerance

Fault tolerance in WSN is a concept whose semantics vary greatly depending on the level of the sensor network discussed. On a middleware level, it also has multiple ways to be applied. Regarding the tasking operations inside the network, what happens if a node faces a temporary hardware or software failure and reboots? There should be a way to learn if this node has been assigned with specific sensing tasks, some sort of a reverse service discovery procedure. As an example approach, TASK [2] deals with this situation using a watchdog timer to detect failures and then retains state from non-failed nodes surrounding failed ones using a query sharing scheme.

There is also the issue of separate sensor hardware failures. Given that some sensor hardware has gone faulty (and not the node as a whole) and starts transmitting mistaken sensed values, is there a way to compare these values to the ones produced by neighboring nodes, in order to test the validity of the sensed values? This aspect of fault tolerance is closely tied to data aggregation, but it can also

be done in a more centralized manner. There is finally the issue of the reliable transmission of data and binary code across the sensor network.

5.4. Software runs on top of the sensor application or runs also on a sensor node level

This criterion refers to whether the software in question runs only on top of an existing sensor network application (and apart from the sensor network itself), or if part of the software environment also runs on a sensor node level. Most of the current implementations feature custom software running on the sensor node level. On the other hand, SenseWeb and GSN use some abstractions to communicate with the the software running on the sensor network gateways, thus they do not depend on the use of some custom software of their own (of course software drivers for each application case have to be implemented).

We believe that both approaches are useful: one should implement functionality on the sensor node level, in order to provide novel services, but also define abstractions, general enough so as to use other types of software, too (in the form of defined APIs and respective drivers). It is an important issue how to extend already implemented platforms and provide application development environments (i.e., software runs above sensor gateway level) to help in the implementation of truly distributed WSN applications.

5.5. Number of wireless sensor networks and heterogeneity

Another important feature is the number of *distinct* sensor networks in the system and whether these networks comprise of different hardware and software subcomponents. The majority of the platforms that appeared in recent years did not take such matters into much consideration. Multiple sources of information, possibly coming from entirely different sensor networks, will be a key feature in implementing very large WSNs. Regarding single sensor networks, some representative approaches are TinyDB [30], Cougar [6], Mote-View [32], Maté [31], ScatterViewer [41] and Agilla [15]. These approaches are also limited in terms of heterogeneity, partly because they assume use of specific OS platforms and hardware, although some provide mechanisms for defining e.g., new sensors. Impala, on the other hand, assumes almost completely homogeneous systems.

Multiple sensor network management is a concept discussed in jWebdust [3]. jWebdust proposes the notion of a virtual sensor network that is comprised of a number of a discrete sensor networks that can be managed as a single one. Extending this multiple sensor network management concept, and introducing a peer-to-peer notion, Hourglass [47] and Global Sensor Networks [1] build on these single sensor network environments to support geographically separated networks and connect them with applications providing a number of services. The key idea is to interface sensor networks and end-user applications to publish locally-generated data streams or request streams of interest, without worrying about the underlying sensor infrastructure. SenseWeb, GSN, MetroSense [9], CarTel [23], furthermore, seem to be supporting heterogeneity to a larger extent, at least theoretically. Another interesting recent environment that enables the use of multiple sensor networks is Sensor Web Enablement [36].

One of future WSN applications objectives should be the support of multiple WSNs in a similar fashion to the way clients connect to a P2P file sharing network, so as to simplify the overall process, extend the functionalities provided in current platforms, and truly support Internet-scale sensing.

5.6. Mobility of nodes and gateways

Another important feature is whether the system supports mobility of nodes and gateways. At the present time, there seem to be only a few systems that explicitly support such features. As the vision of future WSNs grows wider to include new applications, mobility is expected to play a crucial role in the future. The architecture and implementation of a WSN platform will be heavily influenced in the case of adopting such concepts. The idea of using mobile devices, such as mobile phones, to act as sensing devices or intermediates is an example of such a concept. In this case, we believe that the use of both mobility cases is justified. CarTel, Skylark [27] and MetroSense use this idea (of utilizing mobile phones) to a certain extent, while SenseWeb can probably use it as well. TinyLIME [8] is specifically designed to meet mobile gateways needs, although its architecture somehow limits its general appeal, since it uses a single-hop communication scheme. Impala is also designed to function in high mobility scenarios, in fact, to some extent, it bases its architecture in mobility.

We think that the support of both mobile nodes and gateways will be a critical feature in tomorrow's WSN applications. Mobility constraints in present sensor network applicationslimit their applicability and usability. If WSN are to play a part in our everyday life, they should somehow deal e.g., with mobile sensor nodes participating to different sensor network from time to time.

5.7. Single or multiple instances (transparency)

If there exist distinct subnetworks in the system, do they communicate with each other, and if so, how do they interact? System-wise we are interested in transparency [7]. In most cases, even if there are multiple WSNs supported, no direct or indirect communication is available between them, and all results are stored in a central database (like in TinyDB).

IrisNet [16] was a shift from this approach using a distributed database and allowing multiple instances. Hourglass, GSN, Skylark and P2PBridge allow for inter-WSN-operability through P2P communication schemes. Also, Agimone [20] is an example of mobile agents combined with some bridging software to allow the migration of mobile agents from one sensor network to another (thus allowing both data and functionality to move from network

to network). We envision multiple sensor networks interconnected through a P2P network substrate, and interacting by exchanging both data and resources (i.e., access to information and functionality).

5.8. Interfacing to the rest of the world

The first platforms for WSNs provided interfaces to the final user and the rest of the world through custom application interfaces (like TinyDB) or through a web page. This trend seems to be shifting toward providing also interfaces through XML, SOAP, etc., in order to integrate them with other environments, like the case in ArchRock [38], Octavex [35]. Similar approaches are followed in Sensilink [42], SynapSense OneClick [48] and XServe [33] (part of the MoteWorks platform). Of course, in general, when you have a database in your system you can always define interfaces to integrate it somehow with other applications. The issue here is how well do these interfaces work in a WSN scenario. An interesting approach, a spreadsheet interface, was used in [34].

Another approach of providing interfaces to the rest of the world, is through the use of P2P protocols, like in HourGlass, P2PBridge, Skylark. Usually this is done using a P2P software substrate, like JXTA [25]. Also, there is the approach of sensor network gateways functioning as bridges, as in ArchRock and TinyREST [14], mapping internal WSN node IDs to IPv4 and IPv6 addresses. An emerging standard for interconnecting IEEE 802.15.4 and IPv6 networks is *6loWPAN*, an open source implementation of which is featured in [43].

We believe that the use of interfaces like those provided by Google Maps and Google Earth, much like the approach in SenseWeb [40] and SensorScope [45], is an important step toward implementing Internet-scale sensing applications.

5.9. Security and trust issues

By security and trust, we refer to encrypted communication and access authorization, respectively. Apart from using, e.g., TinySec [26], on the sensor node level there isn't really much work implemented in these fields, when referring both to in-sensor-network trust and to inter-WSN application communication. For example, the problem of accepting a newly added sensor node as a trustworthy member of the network is of high importance. Furthermore, it is not clear how a sensor or a user from one WSN or an application client could or should have access to the functionalities and resources of another WSN. A number of these issues are dealt with in Sensor Web Enablement, where end-users may have different roles and privileges accessing data and tasking different sensor networks through the system.

We believe that developers should start taking these two matters seriously into consideration, since their necessity will become more evident in the emerging global scale sensor networks, and incorporate them into their own work. In general, these characteristics should be, to some extent, part of any serious large-scale application.

6. Discussion – Conclusions

The last few years we have seen a large number of WSN hardware, test-beds and application proposals sparkle out. A diversity of approaches, regarding WSN middleware, has been proposed to deal with the multitude of requirements that have arisen from all this activity. As the vision of wireless sensor networking grows to encompass new challenges, new requirements are added to the present ones. In this paper we have presented a fair sample of current WSN middleware solutions along with a simple categorization. The overview is that there is a number of software platforms that offer many services to developers, but current systems do not support all of the requirements of the envisioned future applications. Some important issues rising are the support of security and trust, transparency, standardization, ontologies and mobility support.

Since WSN are currently a very active research field, we have seen a multitude of different proposals and approaches, but still few widely accepted hardware and software standards. Regarding inter-WSN interfacing and service discovery, there are some proposals like SensorML [44] or the concept of virtual sensors in GSN [1], that are specifically tailored to fill this gap.

Another benefit, apart from the introduction of standards itself, is that this could lead also to easier comparisons between WSN middleware solutions. Up till now, there has not really been much work comparing performance between different approaches in important issues like efficiency, scalability, etc. A large part of the published results focuses on describing software components size, use of memory resources, total network lifetime, etc. Though useful, these properties do not necessarily provide insight to realistic scenarios. Perhaps, some benchmark cases could be proposed to promote such comparisons. Although proposals differ in their scope, requirements and complexity, there are common factors that can be evaluated. An evaluation of a number of WSN software solutions based on factors such as power awareness, openness, scalability, mobility, heterogeneity, ease of use, is included in [21].

Also, as stressed in [22], there is the issue of utilizing results from neighboring research fields, like context-aware computing, "smart spaces", etc., that have been using specific *ontologies* to face application requirements. In some ways, this is a top-down approach, contrary to the bottom-up approach used in many WSN middleware proposals, that were built to face specific problems like energy efficiency, etc. Furthermore, the issue of integrating WSN with expert systems is important as well. An example of such an approach is found in GoodFood project [17].

One last question, after seeing so many different design and implementation approaches, is whether there exists a least common denominator in all these approaches. If we

could agree on a small or large set of common services and interfaces, we could easily settle on a number of standards to further simplify WSN application development.

Acknowledgements

This work has been partially supported by the IST Programme of the European Union under contract number IST-2005-15964 (AEOLUS). Also, by the Programme PENED under contract number 03ED568, co-funded 75% by European Union – European Social Fund (ESF), 25% by Greek Government – Ministry of Development – General Secretariat of Research and Technology (GSRT), and by Private Sector, under Measure 8.3 of O.P. Competitiveness – 3rd Community Support Framework (CSF).

References

[1] K. Aberer, M. Hauswirth, and A. Salehi, *The global sensor networks middleware for efficient and flexible deployment and interconnection of sensor networks*, Tech. report, Ecole Polytechnique Federale de Lausanne (EPFL), 2006, Technical Report.

[2] P. Buonadonna, D. Gay, J. Hellerstein, W. Hong, and S. Madden, *TASK: Sensor network in a box*, In the Proceedings of the 2nd European Workshop on Sensor Networks, 2005, pp. 133–144.

[3] I. Chatzigiannakis, G. Mylonas, and S. Nikoletseas, *jWebDust : A java-based generic application environment for wireless sensor networks*, In the proceedings of the first International Conference on Distributed Computing in Sensor Systems (DCOSS '05), 2005, pp. 376–386.

[4] D. Chu, L. Popa, A. Tavakoli, J. Hellerstein, P. Levis, S. Shenker, and I. Stoica, *The design and implementation of a declarative sensor network system*, Tech. report, University of California, 2006, Tech. Report.

[5] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G.P. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis, *The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario*, Proceedings of the 5^{th} IEEE International Conference on Pervasive Computing and Communications (PERCOM07) (New York, USA), IEEE Press., March 2007.

[6] *The Cougar Sensor Database Project homepage*, http://www.cs.cornell.edu/database/cougar.

[7] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, 3rd ed., Addison-Wesley, Boston, USA, 2001.

[8] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. Murphy, and G. Picco, *TinyLIME: Bridging Mobile and Sensor Networks through Middleware*, Third IEEE International Conference on Pervasive Computing and Communications, PerCom 2005, 2005, pp. 61–72.

[9] S. Eisenman, N. Lane, E. Miluzzo, R. Peterson, G. Ahn, and A. Campbell, *MetroSense project: People-centric sensing at scale*, In Workshop on World-Sensor-Web (WSW 2006), Boulder, October 31, 2006.

[10] A. Lachenmann et al., *Versatile support for efficient neighborhood data sharing*, In the Proc. of the European Workshop on Sensor Networks (EWSN 2007), 2007.

[11] E. Souto et al., *Mires: A publish/subscribe middleware for sensor networks*, In the Journal of Personal and Ubiquitous Computing **10** (2005).

[12] M. Balazinska et al., *Data management in the worldwide sensor web*, IEEE Pervasive Computing (2007).

[13] O. Gnawali et al., *The Tenet architecture for tiered sensor networks*, In the Proc. of the 4th international conference on Embedded networked sensor systems (SenSys'06), 2006, pp. 153–166.

[14] T. Luckenbach et al., *TinyREST - A protocol for Integrating Sensor Networks into the Internet*, In the Proceedings of the First REALWSN 2005 Workshop on Real-World Wireless Sensor Networks, 2005.

[15] C. Fok, G. Roman, and C. Lu, *Rapid development and flexible deployment of adaptive wireless sensor network applications*, In Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'05), 2005, pp. 653–662.

[16] P. Gibbons, B. Karp, and S. Seshan Y. Ke, S. Nath, *IrisNet: An Architecture for a World-Wide Sensor Web*, IEEE Pervasive Computing **2** (2003), no. 4.

[17] *GoodFood Integrated Project homepage*, http://www.goodfood-project.org/.

[18] B. Greenstein, E. Kohler, and D. Estrin, *A Sensor Network Application Construction Kit (SNACK)*, In the Proc. of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), 2004, pp. 69–80.

[19] R. Gummadi, O. Gnawali, and R. Govindan, *Macro-programming wireless sensor networks using Kairos*, In the Proc. of the International Conference on Distributed Computing in Sensor Systems (DCOSS'05), Springer, 2005, pp. 126–140.

[20] G. Hackmann, C. Fok, G. Roman, and C. Lu, *Agimone: Middleware Support for Seamless Integration of Sensor and IP Networks*, In the Proceedings of 2006 International Conference on Distributed Computing in Sensor Systems (DCOSS '06).

[21] S. Hadim and N. Mohamed, *Middleware challenges and approaches for wireless sensor networks*, IEEE Distributed Systems Online **7(3)** (2006).

[22] K. Henricksen and R. Robinson, *A survey of middleware for sensor networks: State-of-the-art and future directions*, In the Proc. of MidSens '06, 2006.

[23] B. Hull, V. Bychkovsky, K. Chen, M. Goraczko, A. Miu, E. Shih, Y. Zhang, H. Balakrishnan, and S. Madden, *CarTel: A distributed mobile sensor computing system*, In the Proceedings of SenSys '06, 2006.

[24] M. Isomura, T. Riedel, C. Decker, M. Beigl, and H. Horiuchi, *Sharing sensor networks*, Sixth International Workshop on Smart Appliances and Wearable Computing (IWSAWC), Lisbon, Portugal, Proceedings of the ICDCS 2006, 2006.

[25] *Project JXTA*, http://www.jxta.org/.

[26] C. Karlof, N. Sastry, and D. Wagner, *Tinysec: A link layer security architecture for wireless sensor networks*, In the Proc. of the 2nd ACM Conference on Embedded Networked Sensor Systems (SensSys 2004), 2004.

[27] S. Krco, D. Cleary, and D. Parker, *Enabling ubiquitous sensor networking over mobile networks through peer-to-peer overlay networking*, Computer communications **28** (2005), 1586–1601.

[28] M. Kuorilehto, M. Hannikainen, and T. Hamalainen, *A survey of Application Distribution in Wireless Sensor Networks*, EURASIP Journal on Wireless Communication and Networking **5** (2005), 774–788.

[29] T. Liu and M. Martonosi, *Impala: A middleware system for managing autonomic, parallel sensor systems*, In the proc. of the 9th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP'03), 2003, pp. 107–118.

[30] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, *TinyDB: An Acquisitional Query Processing System for Sensor Networks*, Journal of ACM TODS **30** (2005), 122–173.

[31] *Application specific virtual machines for TinyOS*, http://www.cs.berkeley.edu/ pal/mate-web/.

[32] *Mote-VIEW monitoring software, Crossbow Technology Inc.*, http://www.xbow.com/.

[33] *Moteworks software platform*, http://www.xbow.com.

[34] *MSR Networked Embedded Sensing Toolkit*, http://research.microsoft.com/nec/msrsense/.

[35] *The Octavex platform, Octave Technology Inc.*, http://www.octavetech.com/solutions/octavex.html.

[36] *OGC Sensor Web Enablement, Overview and high-level architecture*, OpenGIS white paper, OGC 06-050r2.

[37] RUNES Project, *Survey of middleware for networked embedded systems*, 2005, Deliverable 5.1.

[38] Arch Rock, *A new embedded web services experience for wireless sensor networks*, In the Proc. of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06), 2006.

[39] K. Romer, O. Kasten, and F. Mattern, *Middleware challenges for wireless sensor networks*, ACM Mobile Computing and Communications Review **6** (2002), 59–61.

[40] A. Santanche, S. Nath, J. Liu, B. Priyantha, and F. Zhao, *SenseWeb: Browsing the physical world in real time*, Demo Abstract, ACM/IEEE IPSN06, Nashville, TN, 2006.

[41] *The Scatterweb wireless sensor network platform*, http://www.scatterweb.de.

[42] *Sensilink WSN Middleware Platform, Meshnetics*, http://www.meshnetics.com.

[43] *NanoStack 6loWPAN open-source implementation*, http://www.sensinode.com.

[44] *Introduction to SensorML, webpage*, http://vast.uah.edu/SensorML/.

[45] *Web page of the Sensorscope project*, http://sensorscope.epfl.ch.

[46] C. Sharp, E. Brewer, and D. Culler, *Hood: A neighborhood abstraction for sensor networks*, In the Proc. of MobiSYS'04, 2004.

[47] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh, *Hourglass: An infrastructure for connecting sensor networks and applications*, Tech. report, Harvard TR-21-04, 2004.

[48] *SynapSense OneClick WSN software architecture*, http://www.synapsense.com.

[49] M. Welsh and G. Mainland, *Programming sensor networks using abstract regions*, In the Proc. of 1st Usenix/ACM Symposium on Networked Systems Design and Implementation (NSDI'04), 2004, pp. 29–42.