Serviceware - A Service Based Management Approach for WSN Cloud Infrastructures

Susan Rea, Muhammad Sohaib Aslam, Dirk Pesch
Nimbus Centre for Embedded System Research
Cork Institute of Technology
Cork, Ireland
e-mail: {susan.rea, dirk.pesch}@cit.ie, Muhammad.S.ASLAM@gmail.com

***Abstract*—With the push for smart environments and the advent of concepts like *Systems of Systems* and *Internet of Things*, the need for underlying large-scale wireless sensor networks (WSNs) infrastructures is evident, however it is likely that no single private user could justify the costs to be incurred for a large-scale WSN deployment and the subsequent management and maintenance costs. In order to drive down costs and maximize the WSN utility a shared infrastructure approach makes large-scale multipurpose WSNs viable. The shared infrastructure paradigm where multiple applications and end users act on a single physical WSN infrastructure in parallel requires a fundamental change in the way WSN resources are managed are, specifically at the WSN device level. This paper draws on the cloud computing infrastructure as a service (IaaS) model and presents *Serviceware* - a middleware approach for infrastructure virtualization in next generation WSNs, where the WSN resources are exposed as services. Virtualization enables the *slicing* of the physical infrastructure into unique segments or virtual sensor networks that can be configured for individual end users according to their specific requirements.**

Keywords-WSN; Serviceware; WSN-SOrA; Cloud Computing;

I. INTRODUCTION

The relentless demand for pervasive computing in our everyday lives has led to and will continue to drive the roll-out of large-scale embedded systems in the form of Wireless Sensor Networks or WSNs in our environment. This in turn drives the need for reusable, flexible and manageable WSN infrastructures. A WSN can consist of a multitude of devices and can be considered as a large-scale infrastructure because of the sheer number of resources they can potentially hold, which can span many thousands of devices with heterogeneous functionalities and capabilities. To simplify system operation and maintenance as well as to reduce costs, WSNs must become infrastructures that are capable of providing services to multiple end users concurrently rather than having to roll-out individual infrastructures for specific purposes as is typical nowadays [1, 2].

WSNs are a rapidly evolving technology that in their current form lack sustainability due to cost associated with the WSN devices themselves, deployment, operation and maintenance of expansive embedded systems infrastructures. WSNs are seen as intrinsic elements of future ICT infrastructures and concepts such as Internet of Things (IoT) [3] and Systems of Systems (SoS) [4], where novel service offerings are based on the digital integration of physical infrastructures through computing systems that enable on-demand service delivery. IoT and SoS are driving applications spaces such as Smart Cities [5] and the like, which will rely on large-scale deployments of WSNs [6]. WSN technologies are evolving but in their current form they will not be able to fully support these concepts due to a lack of support for infrastructure reusability [1, 2, 7]. Network virtualization has been proposed in an effort to cope with exponentially expanding network populations and demands. This is a relatively new phenomenon which allows reusability of hardware infrastructure in order to increase productivity. Virtualization is the essence of cloud computing theory where the end-user is provided resource access in the form of services based on three paradigms i.e. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). New software architectures have arisen to accommodate changes in the networking world and slowly the emphasis has fallen onto providing network infrastructure in the form of services to the end-user systems.

This paper presents a service based infrastructure slicing approach for virtualization in embedded networked devices for cloud based WSNs, which we refer to as *WSN Cloud* in this paper. The WSN Cloud model follows the cloud computing principle of delivering IaaS to the end-user, where a subset of IaaS called Network as a Service (NaaS) is used in the context of this paper to deliver the WSN as a service in cloud environments. A key enabler in delivering virtualization and NaaS is the *Serviceware* middleware, an Service Orientated Architecture (SOA) based middleware that runs over the embedded networked devices and provides virtualization of the underlying hardware in the form of services, which itself is an architectural element of the wireless sensor network service orchestration architecture (WSN-SOrA) introduced in [8]. WSN-SOrA is a service based management approach that orchestrates service provisioning for large-scale embedded networked systems and enables WSNs to act as cloud-ready infrastructure that facilitate on-demand provisioning for potentially multiple individual backend systems. A key enabler in delivering virtualization and NaaS in WSN-SOrA is the Serviceware, an SOA based middleware that runs over the embedded networked devices and provides virtualization of the underlying hardware in the form of services. The reminder of this paper is structured as follows: Section II discusses related work in the areas of WSN middleware and

978-1-4673-5077-8/13/$31.00 ©2013 IEEE

virtualization; Section III presents the proposed Serviceware middleware with Section IV discussing a proof of concept evaluation of the ServiceWare middleware and finally Section V summarizes the contributions of this work and indicates aspects to be considered as future extensions to this work.

II. Related Work

WSNs find application in a variety of pervasive computing environments and continue to be an active area of research for industry and academia alike. However, challenges still persist in areas relating to the design, deployment and management of WSNs together with application development. WSN middleware has forged advancement in these areas limitations still persist with present day approaches being rigid solutions, designed for WSNs that are task specific where the WSN is considered to be a monolithic network with devices being used for a specific purpose only. However, some work has been done in the direction of WSN virtualization and is briefly described next.

A. WSN Virtual Machine based Middlewares

VM based methods abstract the device or node network into an execution layer that executes multiple instances of application programs or scripts, approaches include [9]: Mate, VM*, DAViM, SwissQM, Smart Messages and Agilla. While the concept of a VM is appealing from a hardware utilization perspective and support for multiple application instances, WSN application and domain characteristics impose significant challenges on the nature and scope of the VM that can be realized. The long term management of applications must be considered after the WSN has been deployed, with sensing devices often being embedded in physical settings that are difficult to reach. Consequently the ability to dynamically update applications and system software is hindered. In addition WSN applications must also be energy efficient. VM based applications tend to be slower, as they use an interpreter based execution engine. For VMs to be viable as a WSN system software, they need to be made sufficiently efficient in operation.

B. Cloud Computing & WSNs

The advent of cloud computing pushed the WSN scope further and provides an opportunity to abstract disparate physical WSN sensory data and to make it available in the virtual world in the form of data virtualization where data brokerage services are developed on top of the WSN (i.e. the WSN and the cloud infrastructure are seen as separate entities) and accessed by data clients using a single data access layer with Sensor Cloud [1] being an example of data virtualization for WSNs. Moving beyond data virtualization the WSN Cloud scope looks to next generation networks and supports the provisioning of a single WSN infrastructure on demand and simultaneously by multiple users, here the scope targets infrastructure virtualization where the WSN physical resources are made available in the form of services that can be provisioned on demand by multiple end users concurrently, the WSN Cloud scope operates across all tiers of the WSN infrastructure. In comparison to infrastructure virtualization, data virtualization is limited as the end user has access to the WSN data only and cannot effect changes on the WSN infrastructure itself.

In [10] clouds are defined as being ”*a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization*”. Cloud computing is a mechanism that offers end users a platform where dynamically scalable resources are accessible virtually as services over the Internet. With WSNs being seen a promising enabler of smart cities applications and the like, the widely held belief is that the synergies between WSNs and cloud computing will offer a potential solution to managing next generation networks [11] in the form of both data and infrastructure virtualization. In the context of the work presented in this thesis, infrastructure virtualization is used to define a new scope for WSNs in the form of The WSN Cloud to support service provisioning in next generation WSNs. Cloud computing [12], follows a business model of delivering host services to users and is based on the following principles: Virtualization, Multiplicity, Resource Sharing, SOA, and X as a Service.

Virtualization allows the reuse of a hardware or software resource, in such a way that each instance of the virtualization is unique to a specific user. For instance, virtualization in computer systems allows the user to run their operating system as if they were the sole user of the system, although it may be used by a number of users at the same time. Multiplicity and resource sharing are intrinsically linked where multiplicity looks to provide access to and share the resources of a single infrastructure among multiple users concurrently. SOA is a seminal component of cloud computing, [13], provides an architectural principle which transforms the way in which integration and communication occur between systems. SOA itself is not a standard but an architectural principle which views a system as being comprised of services where these services are asynchronous programs providing functionality through well-defined interfaces. SOA is well established architectural approach in high-end systems; however it is a relatively new concept for embedded network devices. Traditionally, embedded wireless sensor devices have had low computational power; therefore they were not capable of supporting SOA concepts. However, recently these devices have become powerful enough to support SOA [14, 15, 16] along with the required underlying operating systems such as SOS [17], Lorien [18] and Squawk [19] to support SOA principles at the embedded device level. Cloud computing is typically described in terms of providing an ”X as a Service” (XaaS) business model of delivering host resources to the end user. XaaS defines a general philosophy of assuming everything as a service. However in computing the most common translation of XaaS is through the Software, Platform, and Infrastructure (SPI) model.

C. WSN Virtualization

Recently virtualization for WSNs has become a hot topic. For example [20] proposes sensor virtualization in the form of a sensor cloud, however the scope of the idea presented focuses on the collection of data from physical sensors (data virtualization) using predefined virtual groups of sensors where data is collected from the sensory infrastructure and virtualized for the end-user, where users can then select different filters to extract desired data. It is interesting to note that at this stage the focus is still on using WSNs as data collectors and on developing an overlaying IT management infrastructure for rendering it as a Cloud. In other research studies [21, 22] the focus is on the synergy between cloud computing and WSNs and they concentrate on delivering software as service (SaaS) for data virtualizations, similar to [20]. Additional works include [23] which looks at the integration of WSNs and cloud computing by providing a platform for data collection that can then be shared by a number of users. Similarly in [24], which is designed for health applications, the emphasis is on collecting data from traditional WSNs and creating an enterprise IT management network which follows a cloud computing model, SaaS in this particular case, to provide multiple users with subsets of the collected data depending on their requirements. All these works focus on the integration of WSNs and cloud computing and view them as separate entities where data brokerage services are supplied to run on top of the WSN rather than virtualizing the WSN resources. The aim of the Serviceware middleware presented here is to enable WSN infrastructure resources to be virtualized - not by developing an IT infrastructure on top of WSN conforming to a cloud computing model (data virtualizations), but rather by virtualizing the embedded WSN physical infrastructure itself and exposing the WSN device resources as services (this is referred to as infrastructure virtualizations). In contrast to VM based virtualizations is the less resource intensive SOA based Serviceware middleware where services are used to virtualize the resources and provide it as a service to the user and so direct access to the underlying infrastructure is not needed.

III. SERVICEWARE

The Serviceware middleware is an experimental system design for implementation on embedded devices, such as the SunSPOTs [26] and TelosB [25] platforms which are supported by the Squawk and Lorien operating systems respectively, which virtualizes the underlying functionality of WSN devices. The Serviceware consists of a series of components, see Figure 1, namely the service chain, service, threader and encoder/decoder components. Provisioning in embedded devices is supported by the concept of a service chain. A service chain is a collection of services which are executed in a sequence and the output of each service is passed to the next service in the chain, the last service in the chain is usually a radio communication service to send data to the required destination. Consider a service chain for a temperature reading would involve the service to read the value recorded by the temperature sensor, a service to create a radio packet and a service to send the data over the physical interface.

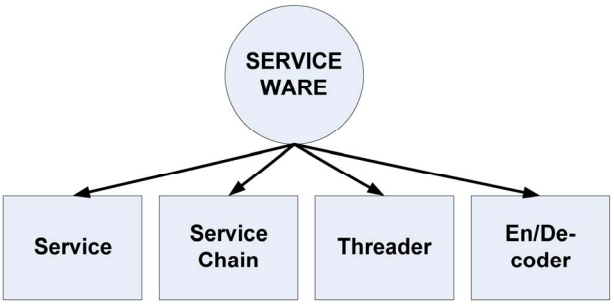

Figure 1. Serviceware Components

In the Serviceware each service chain execution is controlled by a separate thread which is a part of the threader component and the frequency of execution or duty cycle of the chain depends upon the type of data collection requirement i.e. periodic or event based. Threads are defined as ”sequential processes that share memory”. In a single system with a single processing unit multi-threading allows a parallel execution model. It provides the illusion of tasks running in parallel on a single processor which in realty are sharing the processor time. This is managed by the scheduler where each thread is provided with a window for execution while other threads wait for their turn in queue. Multi-threading enables the Serviceware to associate a thread with the sequential execution of objects. The objects represent service components (classes) that contain the implementation to access the hardware resource i.e. sensory components such as a light sensor for example. This approach allows multiple threads to run on an embedded device in parallel, executing the objects that are instances of the service classes in a sequence that forms a service chain. The sampling rate specified by the end-user defines the waiting time for the thread. While the threads are executed in order multithreading gives the illusion that the threads were execute at the same time. The multi-threading parallel execution model is used by the Serviceware to virtualize the underlying hardware resources among multiple end-users concurrently.

The Serviceware builds upon Object Oriented Design (OOD) that can be translated for any device that support operating systems which are compatible with the OOD approach. Shown in Figure 2 is an overview of the Serviceware implementation on an embedded device. The Serviceware middleware acts as an encapsulating layer over the operating system (OS) in order to virtualize the hardware resources. The services in the Serviceware use the API provided by the OS layer to access the hardware resources. Each provisioning requirement for an user is encapsulated as service chain, this is an object which executes the services as specified by the end users. Each service chain object is controlled by a thread and OS environments such as as Lorien and Squawk provide a multithreading mechanism that allows more than one of these threads to run in parallel. For example consider the SunSPOT platform which runs the

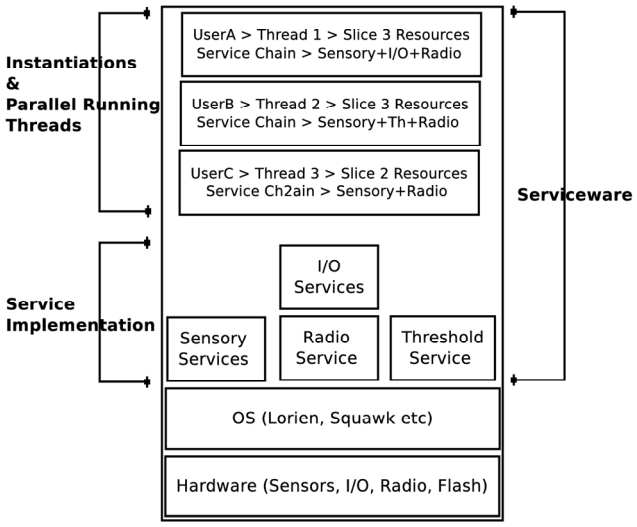

Figure 2. Sensorware Design

Squawk OS, this OS provides APIs that can access hardware in a managed or unmanaged way. Unmanaged access provides direct access to the MAC and physical layers (the SunSPOT uses the IEEE802.15.4 compliant CC2420 radio) whereas managed access gives access to standard network layer routing protocols that include AODV and LQRP. This presents the Serviceware middleware with two options, it can use managed access and in this case the Serviceware radio service (used to send a packet) is a simple service that provides a parameter to select any of the available routing protocols provided by the OS and it uses the available MAC and physical layers. Now consider a scenario where a custom routing protocol is required for example, in this case the custom routing protocol must be designed as a service that is compatible with the Serviceware. Unmanaged access is then used to bypass the routing protocol provided by the OS.

A. Commissioning & Provisioning

The initial commissioning of devices includes installing an appropriate OS on the embedded device hardware, for example in an embedded network device such as SunSPOT, this will include loading the Squawk operating system. Configuring embedded WSN devices involves porting of the Serviceware code onto the device, this can be done using Over the Air (OTA) Transfer or using a wired link. The size of the Serviceware middleware code is directly affected by the program memory available on the embedded network devices, where the Serviceware code size reflects the number of services that an implementation can hold. For example the Serviceware for SunSPOTs can hold services for temperature, light, humidity, multiple routing protocols and other aggregation services with the TelosB device offering temperature, light and humidity. Once devices have been commissioned the WSN can be deployed.

WSN gateways manage the wireless network and aggregate data from distributed WSN devices that can be pushed toward backend cloud based systems. The gateway can also be used to configure and reconfigure the WSN devices; we refer to this as provisioning in the context of the Serviceware middleware a provisioning packet is transmitted by the gateway to the WSN devices. The size of the provisioning packet can vary based on the protocol used for communication and the number of simultaneous services an embedded device can support. Generally, WSN communication conforms to the IEEE 802.15.4 standard and the radio packet has an upper size limit of 127 bytes out of which the payload is a maximum of 102 bytes. The provisioning packet data is stored in the payload of the IEEE 802.15.4 radio packet and must be constructed in a sequence that is readable and compatible with the Serviceware running on the embedded device.

B. Serviceware Call

Shown in Figure 3 is the sequence of operations in a Serviceware proof of concept implementation. The provisioner module in the gateway program sends the provisioning packet to the Serviceware on the embedded device. These packets can be routed using commonly available WSN routing protocols such as Link Quality Routing Protocol (LQRP) or Ad hoc On-Demand Distance Vector (AODV), or indeed any underlying routing protocol will suffice. The Service module receives the provisioning call and passes it to the Protocols module to decode the provisioning packet. The Primaries module is the module that creates a thread for the provisioning and associates it with the provisioning id. This module then uses the Service base which contains the implementation of the services for the device and activates the service objects that are requested in provisioning call to form a service chain. The contextual data (sensory data) is transferred by communication services as part of service chain execution.

C. Provisioning Request Retry Management

Provisioning Request retires are linked to the MAC layer retries for the implementation presented here. The IEEE802.15.4 MAC layer retry limit is set to three. if after three attempts the provisioning packet fails to send, it is dropped at the MAC layer and this generates an exception which informs the program running on the gateway to regenerate the provisioning packet and to attempt the provisioning packet dissemination again. The limit for this ”upper layer” retry is configurable and for the implementation here values between 2-5 were used. If this retry limit is reached the provisioning operation is cancelled, provisioning rollback is performed where all devices are deprovisioned. In deprovisioning the thread running in Serviceware associated with servicechain that was previously provisioned is terminated and the cloud user is informed. Likewise for multihop scenarios if provisioning fails for devices that are two or more hops away from the gateway, the provisioning operation is again cancelled and a provisioning rollback is performed for all devices concerned and again the the cloud user is informed.

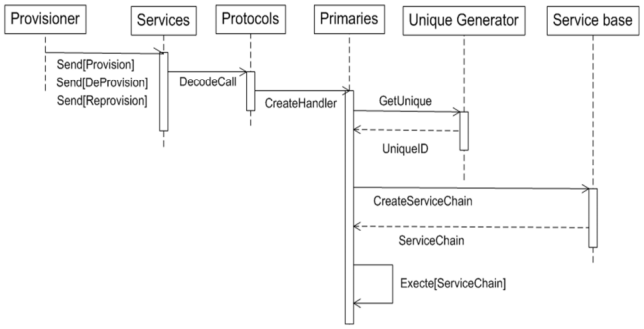

Figure 3. Serviceware Operation

D. *Serviceware To Data Collection End Point Setup*

Once the embedded network devices are provisioned, ports on the gateway are opened to enable the collection of data from the embedded devices. The data at the collection point is can be either pushed to the cloud user or processed at the gateway.

E. *Serviceware: Advantages & Disadvantages*

The Serviceware middleware has been designed to support virtualizations on embedded WSN devices where the hardware resources are exposed as services. Providing access to the hardware via services prevents the deployment of end user application code directly on to devices (NaaS model), this prevents the exploitation of the underlying hardware resources and OS by imprudent application code practices and malicious users. The Serviceware middleware requires the underlying operating system (OS) to support multithreading as a fundamental requirement. Virtualization supports multiple services (and subsequently multiple end users) running over a single device, in comparison to traditional WSNs this shared infrastructure approach does not exist. While virtualization maximizes the WSN device utility it will increase the energy expenditure per device when multiple services and users are acting on it in comparison to monolithic or data virtualization approaches. Consequently energy management and the number of services that can be provisioned on a single device must be carefully managed in order not to over utilize the available resources.

IV. SERVICEWARE: PROOF OF CONCEPT IMPLEMENTATION

A prototype of the Servieware middleware implemented using a heterogeneous mix of sensor device platforms consisting of SunSPOTs (4MB Flash) running the Squawk OS and TelosB (48KB Flash) devices running TinyOS [27] or the Lorien OS. TinyOS running over the TelosB platform was used a baseline in this experiment to compare against the Serviceware middleware requirements and capabilities. TinyOS is a monolithic OS and as such a complete OS image must be deployed for sensor configuration and re-tasking. Lorien and Squawk on the other hand are component based operating systems that support multi-threading and OOD making them compatible with the Serviceware middleware. The size of NaaS endpoint Serviceware middleware code was limited by the program memory available on the embedded network devices, where the Serviceware code size reflects the number of services that a specific implementation can hold for a target hardware platfom. The telosB devices running tinyOS were configured to support a temperature service and the Serviceware code size (in this case the complete OS image) was 8KB. For the telosB devices running Lorien, temperature and humidity services were configured with an initial code size of 38KB and finally for the SunSPOT devices temperature, light, humidity, routing and acclerometer services were configured with an initial code size of 1.6MB. Loading of the initial Serviceware on the WSN devices depends on the serial connection with the device and for the implementation discussed here deployment of the Serviceware took 1-2s and is dependent on the Serviceware code size. When the WSN is live and is required to be reconfigured the provisioning packet size depends upon the communication standard in use, for example IEEE 802.15.4 was used in this implementation and supports a maximum frame size of 127 bytes and transmission rate of 250kbps and as such the provisioning packet maximum size is 127B. Provisioning is exclusive to systems that can support SOA and in this case this means that the devices running Lorien and squawk are capable of being provisioned using a single packet that is 127B in size. TinyOS follows a monolithic image deployment methodology and so a provisioning operation is not applicable in this case as deployment of the whole OS image is required each time (this is equivalent to redeploying the Serviceware middleware repeatedly). The provisioning time depends upon the provisioning packet size, bandwidth and any disruption or interference in the radio communication between the wireless devices. Using over the air programming (OTA) the provisioning time was measured as being approximately 4ms. Cost is an important factor in any infrastructure deployment. The NaaS approach allows a single infrastructure to be reused simultaneously. This approach reduces the infrastructure cost significantly. If for example two users have different requirements the typical approach is to configure and schedule individual use of the network for each user independently. Using Serviceware the cost in terms of time and infrastructure is reduced by 50% as it allows both users to use the network simultaneously. As technology advances WSN network devices will have increasing functionality and Serviceware provides an efficient mechanism for infrastructure reusability in the form of NaaS.

The Serviceware middleware delivers on the cloud computing business model principles (virtualization, multiplicity, resource sharing, SOA, X as a Service) with the Servicewarwe relying on Naas, a subset of IaaS, to virtualize the WSN resources. Serviceware is an SOA based middleware that takes advantage of multi-threading to slice the WSN hardware which supports resource sharing and multiple end user concurrently.

V. CONCLUSION & FUTURE WORK

This paper describes the concept of service based infrastructure slicing. The Serviceware middleware

presented in this paper is a key component of a comprehensive architecture designed specifically for provisioning in next generation cloud based WSN infrastructures. The Serviceware middleware slices the infrastructure using services, where a fundamental requirement is that the underlying operating system must support multi-threading and compliance with SOA approaches. The service chain module in the Serviceware allows the services to be bundled together to support a sequence of processes to deliver the functionality of the hardware resources, such as sensory components in a WSN network device. This approach lays the foundation for infrastructure slicing that is inspired by the high-end network devices where quasi services such as VLAN, VRouter provide the same functionality. In order to provide more grained control over the WSN device resources the Serviceware virtualization should be extended to include the memory and CPU components. While virtualization will increase the energy use per device, a tradeoff between exploiting the benefits of virtualization against the need to replace batteries frequently must be further investigated. In addition, energy harvesting devices can be used to replenish energy sources which can lead to infinite device lifetimes and should be considered.

ACKNOWLEDGMENT

The authors wish to acknowledge the support of Science Foundation Ireland under the 06-SRC-I1091 ITOBO project in part funding the work reported in this paper.

REFERENCES

[1] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure – physical sensor management with virtualized sensors on cloud computing," in Network-Based Information Systems (NBiS), 2010 13th International Conference on, sept. 2010, pp. 1 –8.

[2] W. Kurschl and W. Beer, "Combining cloud computing and wireless sensor networks," in Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services, ser. iiWAS '09. New York, NY, USA: ACM, 2009, pp. 512–518. [Online]. Available: http://doi.acm.org/10.1145/1806338.1806435

[3] C. Pfister, Getting Started with the Internet of Things, B. Jepson, Ed. O'Reilly, 2011.

[4] C. Dagli and N. Kilicay, "Understanding behavior of system of systems through computational intelligence techniques," in Systems Conference, 2007 1st Annual IEEE, april 2007, pp. 1 –7.

[5] H. Waer and M. Deakin, From Intelligent to Smart Cities. Taylor & Francis, 2012.

[6] Y. Lu, N. Huansheng, Y. Laurence T., and Y. Zhang, THE INTERNET OF THINGS: From RFID to the Next-Generation Pervasive Networked Systems. Auerbach Publications, 2008.

[7] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," Comput. Netw., vol. 54, pp. 862–876, April 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2009.10.017

[8] M. Aslam, S. Rea, and D. Pesch, "Service provisioning for the wsn cloud," in Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, june 2012, pp. 962 –969.

[9] I. Chatzigiannakis, G. Mylonas, and S. Nikoletseas. 50 ways to build your application: A survey of middleware and systems for wireless sensor networks. In Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on, pages 466 –473, sept. 2007.

[10] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," SIGCOMM Comput. Commun. Rev., vol. 39, no. 1, pp. 50–55, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496100

[11] R. Liu and I. J. Wassell, "Opportunities and challenges of wireless sensor networks using cloud services," in Proceedings of the workshop on Internet of Things and Service Platforms, ser. IoTSP '11. New York, NY, USA: ACM, 2011, pp. 4:1–4:7. [Online]. Available: http://doi.acm.org/10.1145/2079353.2079357.

[12] G. Reese, Cloud Application Architectures: Building Applications and Infrastructure in the Cloud. OReilly, 2009.

[13] Michael Bell. Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture. Wiley, 2008.

[14] A. Rezgui and M. Eltoweissy. Service-oriented sensor-actuator networks [ad hoc and sensor networks]. Communications Magazine, IEEE, 45(12):92 –100, december 2007.

[15] Edgardo Avils-Lpez and J Antonio Garca-Macas. Tinysoa: a service oriented architecture for wireless sensor networks. Service Oriented Computing and Applications, 3(2):99–108, 2009.

[16] E. Meshkova, J. Riihijarvi, F. Oldewurtel, C. Jardak, and P. Mahonen. Service-oriented design methodology for wireless sensor networks: A view through case studies. In Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC '08. IEEE International Conference on, pages 146 –153, june 2008.

[17] Chih chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. Sos: A dynamic operating system for sensor networks. In Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys). ACM Press, 2005.

[18] Barry Porter and Geoff Coulson. Lorien: a pure dynamic component-based operating system for wireless sensor networks. In Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks, MidSens '09, pages 7–12, New York, NY, USA, 2009. ACM.

[19] Oracle. Squawk for SunSPOT. http://labs.oracle.com/projects/squawk/squawk-sunspot.html.

[20] M. Yuriyama and T. Kushida. Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing. In Network-Based Information Systems (NBiS), 2010 13th International Conference on, pages 1 –8, sept. 2010.

[21] Werner Kurschl and Wolfgang Beer. Combining cloud computing and wireless sensor networks. In Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services, iiWAS '09, pages 512–518, New York, NY, USA, 2009. ACM.

[22] K. Lee and D. Hughes. System architecture directions for tangible cloud computing. In Cryptography and Network Security, Data Mining and Knowledge Discovery, E-Commerce Its Applications and Embedded Systems (CDEE), 2010 First ACIS International Symposium on, pages 258–262, oct. 2010.

[23] K. Ahmed and M. Gregory. Integrating wireless sensor networks with cloud computing. In Mobile Ad-hoc and Sensor Networks (MSN), 2011 Seventh International Conference on, pages 364 –366, dec. 2011.

[24] Xuan Hung Le, Sungyoung Lee, Phan Truc, La The Vinh, A.M. Khattak, Manhyung Han, Dang Viet Hung, M.M. Hassan, M. Kim, Kyo-Ho Koo, Young-Koo Lee, and Eui-Nam Huh. Secured wsn-integrated cloud computing for u-life care. In Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE, pages 1 –2, jan. 2010.

[25] Crossbow. http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=252.

[26] Sun Small Programmable Object Technology (Sun SPOT). https://labs.oracle.com/projects/dashboard.php?id=145.

[27] TinyOS. http://www.tinyos.net/.