Reference Architecture for Self-adaptive Management in Wireless Sensor Networks

Jesús M.T. Portocarrero[1], Flavia C. Delicato[1], Paulo F. Pires[1], and Thais V. Batista[2]

[1] PPGI-iNCE/DCC-IM/Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
{jesus140,fdelicato,paulo.f.pires}@gmail.com
[2] DIMAp, Federal University of Rio Grande do Norte, Natal, Brazil
thaisbatista@gmail.com

Abstract. Self-adaptive component-based architectures facilitate the building of systems able of dynamically adapting to varying execution contexts. Such a dynamic adaptation is particularly relevant in the domain of wireless sensor networks (WSNs), where numerous and unexpected changes of the execution context prevail. In this paper, we introduce a reference architecture for WSNs in order to contribute to middleware development for enabling self-adaptive behavior in service-oriented WSNs. This reference architecture follows the autonomic computing model MAPE-K, for making decisions aiming to attend self-adaptive WSN requirements. At the end of this paper, we present a case study to explain how instantiate our reference architecture in order to create a specific concrete middleware for WSN.

Keywords: Autonomic computing, sensor network, reference architecture.

1 Introduction

Wireless sensor networks (WSN) consist of networks composed of devices equipped with sensing, processing, storage, and wireless communication capabilities. Each node of the network can have several sensing unit, which is able to perform measurements of physical variables. The nodes in a WSN have limited computing resources, and are usually powered by batteries; thus energy saving is a key issue in these networks in order to prolong their operational lifetime. WSN nodes operate collaboratively, extracting environmental data, performing same simple processing and transmitting them to one or more exit points of the network called sink nodes, to be analyzed and further processed.

There is currently a wide range of applications for WSN, ranging from environmental monitoring to structural damage detection. The first WSN applications had simple requirements that did not demand complex software infrastructures. Typically WSN were designed to attend requirements of a unique target application. However, the rapid evolution in this area and the increasing of complexity of sensors and applications involved the need of specific middleware platforms for these networks. Furthermore, typically WSN are used in highly dynamic and hostile environments, without human participation, and therefore, they should have an autonomous behavior, able to be fault-tolerant coverage and connectivity. Sensor nodes must be smart to recover

A. Bouchachia (Ed.): ICAIS 2014, LNAI 8779, pp. 110–120, 2014.

autonomously from failures with minimal human intervention; in other words, WSN should be able to self-manage and to adapt itself to the context dynamically.

According to [1], autonomic computing, also known as self-adaptive computing, is a capacity of an infrastructure for adapting itself according to policies and business goals. Autonomic computing just tries to help IT professionals to focus in higher value tasks, turning technological work more intelligent, with business rules oriented to self-management. These rules also known as self-* properties are: (i) Self-Configuration, it is the ability to adapt itself to the environment changing according to high-level policies, aligned with business goals and defined by system administrators; (ii) Self-Healing, it is the ability to recover after a system disturbance and to minimize interruptions to maintain the software available for the user, even in the presence of individual failure of components; (iii) Self-Optimization, it is the system ability to improve its operation continuously and (iv) Self-Protection, it is the ability to predict, detect, recognize and protect from malicious attacks and unplanned cascade failures.

A highlight approach to develop autonomic systems is the architecture to autonomic computing proposed by IBM [2] that defines an abstract framework for self-managing IT systems. In this framework an autonomic system is a collection of autonomic elements. Each element consists of an autonomic manager and a managed resource. In the context of WSN, an autonomic manager can be a middleware system and the own sensor network represents the managed resource. The autonomic manager allows adaptation through four activities: monitoring, analyzing, planning and executing, with support from a knowledge base. In monitoring activity, elements collect relevant data via sensors to reflect the current state of the system – the managed resource (and thus, grant it context awareness). In analyzing activity, the collected data are analyzed in search of symptoms relating the current and desired behavior. The planning activity decides whether is necessary to adapt the system to attend the goals defined previously. In execution activity are instrumented the desired adaptation acts by actuators or effectors. In order to implement these activities to allow self-adaptation of software feedback loops are required, with explicit functional elements and interactions between them for managing the dynamic adaptation. These elements are known as MAPE-K model (Monitor, Analyze, Plan, Execute and Knowledge Base). Feedback control loops are considered a key issue in pursuing self-adaption for any system, because they support the four above-mentioned activities. They play an integral role in adaptation decisions. Thus, key decisions about a self-adaptive system's control depend on the structure of the system and the complexity of the adaptation goals. Control loops can be composed in series, parallel, multi-level (hierarchical), nested, or independent patterns. We refer the interested readers to [3] which have further discussed the choices and impact of control loops on the design of self-adaptive systems. In this context, Autonomic Computing is presented as an interesting option to meet basic requirements in WSN design. Thus, autonomic computing principles can be applied to WSN in order to optimize network resources, facilitate their operations and achieve desired functionality in the wide field of sensing-based applications and providing conditions for this type of network manage itself without involve human operators. So, a WSN becomes an autonomous WSN. The MAPE-K model described above provides conceptual guidelines about the autonomic systems conception; in practice, this information model needs to be mapped to an implementable architecture for managing and control of autonomic WSNs.

Hence, this work proposes a middleware reference architecture for self-adaptive management of WSN. Middleware for WSN [4] assists the development of WSN applications, providing services and abstractions that hide details about underlying hardware devices and low-level software mechanisms. Reference architectures are created based on reference models and architectural patterns [5]. Our reference architecture adopts the MAPE-K model and a component-based and service-oriented approach. The main purpose of a reference architecture is to facilitate and guide [6] (i) the design of concrete architectures for new systems; (ii) the extensions of systems of neighbor domains of a reference architecture (iii) the evolution of systems that were derived from the reference architecture and (iv) the improvement in the standardization and interoperability of different systems. These play a dual role in relation to specific software architectures, the first role generalizes and extracts common functions and configurations, and the second role provides a base for instantiating target systems. In other words, reference architectures can be seen as a repository of a given knowledge area, contributing towards software development, since the reuse of knowledge and improvements of productivity are promoted. Thus, the proposed middleware reference architecture aims to satisfy this dual role in WSN domain. This middleware reference architecture has been designed applying a service-based approach [7], in which the WSN is seen as a service provider for user applications. The service provided by the WSN is data collection and delivery. The services provided by the middleware are the interpretation of the application requirements and the selection of the best initial network configuration and network reconfiguration based on those requirements.

The rest of the paper is organized as follows: Section 2 introduces the self-adaptive WSN requirements addressed in the work. Section 3 details the proposed approach. An instance of our reference architecture is described and analyzed in Section 4. Section 5 draws conclusions and related work.

2 Self-adaptive WSN Requirements

Considering WSN singularities, especially with regard to resource constraints, there are some requirements of design in WSN applications that also must be considered in the middleware design for WSN:

- **Hardware resources:** the advent in microelectronics technology made it possible to design miniaturized devices on the order of one cubic centimeter. These tiny devices could be deployed in hundred or even thousands in harsh and hostile environments, where in some situations a physical contact to maintain or replace these devices is impossible and wireless media is the only way for remote accessibility.
- **Scalability and dynamic network topology:** the network topology is subject to frequent changes due to diverse factors as devices failures, mobile obstacles, mobility and interferences. If an application grows, the network should be flexible enough to include other nodes anytime without impacting network performance. A WSN middleware should support mechanisms for fault tolerance and sensor nodes self-maintenance. In order to attend these requirements, **Topology Control** and **Fault Tolerance** mechanisms are required.

- **Dynamic network organization:** Unlike traditional networks, sensor networks must deal with resources that are dynamic, such as energy, bandwidth, and processing power. An important issue is to support applications in the efficient design of routing protocols and providing ad hoc network resource discovery, because knowledge of the networks is essential for it to operate properly.
- **Application knowledge:** An autonomic middleware for WSN must include mechanisms for injecting application knowledge of WSN infrastructure. This allows mapping the application requirements with the network parameters, and adjusts the process of network monitoring.
- **Focused on data:** WSN applications generally are not interested in node identity, but the data it produces, especially when the same types of nodes are deployed to produce the same type of data. A WSN middleware should support the centrality of data, providing mechanisms for routing and centralized query inside the network. Mechanisms for **Sensing and Data Delivery** are appropriate.
- **Quality of service (QoS):** Traditional networks only move data from one place to another, however, nodes in WSN work collaboratively to move data, monitor and control an environment. For this type of networks, data confidence determines that an event that should be detected was in fact detected.

3 Reference Architecture for Self-adaptive Management of WSN

This Section details the proposed approach. First is presented the architectural styles and design patterns used in the reference architecture designing process and after that the components of the architecture are detailed. We consider the interactions among the different activities of control loops realized by the MAPE-K components.

3.1 Architectural Styles and Design Patterns

Software architectures is almost never limited to a single architectural style, is often a combination of architectural styles that make up the complete software. To built our reference architecture we used combinations of the following architectural styles.

- **Layer Architectural Style:** Focuses on the grouping of related functionality within an application into distinct layers that is stacked vertically on top each other. The main benefits are abstraction, isolation, manageability, performance, reusability and testability. Our reference architecture contains three 3 layers (Figure 1a): Sensor MAPE-K Layer (SML), Network MAPE-K Layer (NML) and Goal Management Layer (GML). SML concerns the autonomic management inside sensor devices; NML concerns the autonomic management in the whole network and GML aims to set adaptation policies used by underlying layers in order to perform adaptations. Also, this layer allows to get the current network status. At this level of abstraction the NML acts as the autonomic a manager and the SML acts as a managed resource. The communication between GML and NML is based on SOA services, and the communication between NML and SML uses the Message Bus Architectural Style. Here, the communication is based on messages that use known schemas.

- **Component-based and Service-Oriented Architectural Style.** Component architectural style focuses on the decomposition of the design into individual functional or logical components that expose well-defined communication interfaces containing methods, events, and properties. The main benefits of this approach are: easy of deployment, reduced cost, easy to development, reusable and mitigation of technical complexity. Service-oriented architectural style enables application functionality to be provided as a set of service. At the component level we applied the pattern proposed by [8] that describes the structure of service components for using in self-adaptive systems. The interfaces are (see Figure 1b): Input, used to receive information; Output, used to send information; Sensor, that makes the component able to achieve information from the external; Effector, that makes the component be able to manage the external; Emitter, used to emit status information to an external manager.
- **Decorator Pattern.** For the general implementation of the components we adopted the design pattern Decorator that allows behavior to be added to an individual object, either statically or dynamically, without affecting the behavior of other objects from the same class. This pattern, depicted in Figure 1c, is based on three types of entities: (i) Interfaces that define services provided by components, (ii) Abstract classes with the definition of basic methods, services and references to other components, and (iii) Implementation classes that define the specific required behavior.

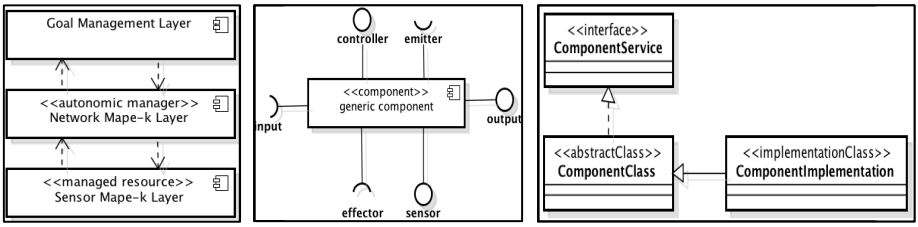

Fig. 1. (a) Layers of Reference Architecture (b) Structure of service components (c) General implementation of components

3.2 Reference Architecture Components

Our reference architecture assumes a network as a set of heterogeneous nodes (node manager and managed nodes) and sink nodes. The sink node acts as the gateway connecting WSN nodes and external networks and applications. Node manager acts as autonomic managers of managed nodes; this type of nodes manages a group of nodes organized in clusters. Node managers are cluster heads of clusters. On the other hand, Managed nodes receive adaptation messages from Node managers.

The overall system is controlled by a hierarchical control structure where complete MAPE-K loops are present at all architecture layers of the hierarchy. MAPE-K loops at different levels interact with each other by exchanging information. The MAPE-K loop at a given level may pass to the level above information it has collected, possibly filtered or aggregated, together with information about locally planned actions, and may issue to the level below directives about adaptation plans that should be refined

into corresponding actions. Our proposed architecture depicted in Figure 2, contains three levels of layers following the architectural style presented in Section 3.1.

The SML allows adapting a node configuration according to nodes context information and adaptation policies. At this point, our reference architecture considers as context information: battery level of sensor nodes, data delivery model/send rate of sensing data, state of nodes (active/inactive/idle), ID of nodes, type of node (node manager/managed node) function (routing/sensing/storing), power of signal, localization. Node managers may create new configurations for managed nodes localized inside the cluster. Each node manager is responsible for managing its own cluster. In order to determine all adaptation actions needed to reconfigure managed nodes of a cluster, a node manager receives continuously context information from its managed nodes and a MAPE-K process is performed to verify the need of an adaptation, if needed, a node configuration message is created. A node configuration message contains information able to modify the topology of the cluster (activating and deactivating managed nodes), to adjust tasks performed by cluster members (such as, data delivery model) and node functions (routing, sensing storing).

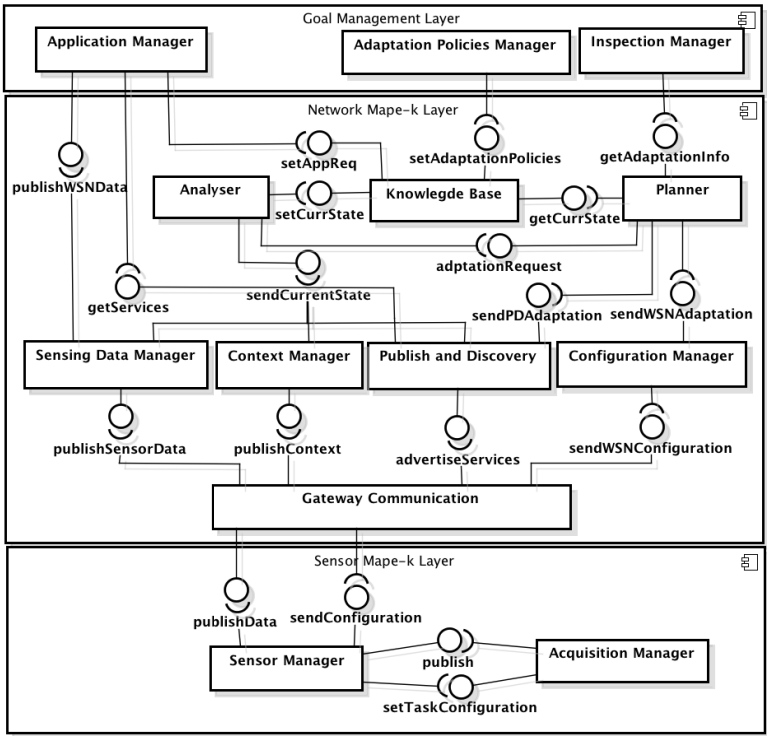

Fig. 2. Reference architecture for self-adaptive management of WSN

In addition, this layer is able to receive a cluster configuration message sent by NML. This message contains a set of task to be executed by sensor nodes and the adaptation policies used by the underlying layer in order to support adaptation decisions. SML consists in two service components: SensorManager and AcquisitionMa-

nager. All components of our architecture follow the pattern for autonomic computing components proposed by [8], presented in Section 3.1.

- **SensorManager** component manages the nodes behavior and determines all adaptation actions needed to reconfigure: (i) a cluster, if the node is configured as a manager, (ii) itself, if the node is configured as a managed node. This component is responsible for executing the MAPE-K process.
- **AcquisitionManager** component collects measures of physical phenomena monitored by sensors and executes the data delivery. If an adaptation request defines changes in the data delivery model, this component will be notified through the interface called setTaskConfiguration.

The NML performs adaptation actions to the network configuration. Thus, the contextual information used for this activity is provided by whole network. This layer contains service components that consists in:

- **GatewayCommunication**: This component provides to Analyser component, contextual information collected through the SML. This information is collected using the interface publishData. This interface collects: sensing data (measures of physical phenomena), context information (such as battery status) and the services provided by a node (such as temperature, humidity).
- **Analyser** component uses the sendCurrentState interface to collect contextual information of network and detects symptoms to determine a network adaptation need. This interface collects: sensing data provided by Sensing Data Manager component, context information of nodes provided by Context Manager component and the current services of network provided by Publish and Discovery component. Also, the Analyser uses other sources of contextual information, from application requirements, such as data delivery model, desired services and QoS requirements. With this information the Analyser component supports the implementation of methods for verifying if application requirements, coverage and connectivity are been guaranteed, and verify the energy state of network. If an adaptation need is detected must be performed an adaptation request using the adaptationRequest interface.
- **Planner** component plans a network configuration once an adaptation request is sent by Analyser component. The Planner component considers adaptation policies in order to generate a network configuration. A policy specifies a set of actions that should be taken by the middleware upon the occurrence of adaptation requests. An adaptation policy can be defined using XML or JSON files.
- **ConfigurationManager** receives configuration parameters through the sendWSNAdaptation interface, translates these parameters in a configuration message and disseminates this configuration to the network nodes, through the Gateway Communication component.
- **SensingDataManager** receives sensing data from Gateway component and publishes these data to application monitor component. Also, this component publishes the sensing data to Analyser component in order to analyze the context of the network.
- **ContextManager** receives sensor context information from Gateway Monitor and publishes this information to Analyser component.

- **PublishandDiscovery**, receives an advertise of service from Gateway Monitor, provided by sensor nodes. This data is published to Application Monitor and Analyser component.
- **KnowlodgeBase** store all context information and support the MAPE-K process.

The proposed reference architecture is based on self-adaptation principles and in order to perform this autonomic behavior a minimal human intervention is required. The components that allow human interaction to define the policies and configurations of network adaptation mechanisms are in the GML and its components are ApplicationManager, AdaptationPoliciesManger and Inspection Manager.

- **ApplicationManager** component is used to create applications, present to the end-user network provided services and monitoring sensing data.
- **AdaptationPoliciesManager** is used to define adaptation policies. This component uses the setAdaptationPolicies interface to offer these policies to the Network MAPE-K layer.
- **Inspection Manager** is used to inspect adaptation information of middleware that is accessible to external environment.

4 Case Study

In this section we defined a case study in order to instantiate a concrete architecture derived from our reference architecture. The concrete architecture consists in the definition of a specific WSN middleware to perform self-adaptation of network configuration in order to preserve energy consumption of the network. Energy management was the main adaptation requirement of this concrete middleware whereas WSN nodes have limited computing resources, thus energy saving is a key issue in these networks in order to prolong their operational lifetime. For such, this energy-aware middleware must guarantee: (i) application requirements (ii) the network has sufficient residual energy to attend all running applications, (iii) each sensor node becomes active whether it has residual energy to guarantee its work until the end of task allocated to it, and (iv) the network must be fully connected in terms of radio communication. Table 1 shows a mapping between the reference architecture layers and the case study requirements and depicts activities that must be executed in every feedback cycle. The NML and the GML of the concrete architecture were implemented in Java programming language. The SML was implemented in NesC (network embedded systems C) programming language, an extension of C programming language [9]. This layer was deployed in MicaZ sensor platform, which runs on TinyOS operating system.

Figure 3 shows the NML class diagram. Each component was implemented following the general structure of components (Decorator pattern). It is important to note that these abstract classes are connected to other service interfaces. This approach allows to clarify the component relationship and to define the behavior of components. Thus, we can see a clear and complete separation among mechanisms that handle adaptation issues of the reference architecture and the specificity of the instantiated middleware. In this case, seven implementation classes (gray classes) extend our reference architecture (white classes) in order to create the specific energy-aware middleware detailed in this case study: GatewayImpl, AnalyserImpl, PlannerImpl, ConfigurationImpl and

GatewayExecutorImpl. SML, implemented on MicaZ/TinyOS platform, publish context messages with residual energy information to the GatewayMonitor class, in order to process context message and select a list of best nodes.

Table 1. Specific actions of every concrete architecture layer

MAPE-K Components	SML	NML
Monitor tasks	-To monitor residual energy of nodes	-To monitor data delivery rate and residual energy -To monitor state of nodes
Analyzer tasks	-Residual energy must last by the allocated task time.	-Residual energy must last by runtime application time. -Network fully connected, coverage redundancy
Planner tasks	-When an undesirable energy level is detected: send an alert message, reduce data delivery interval.	-When an undesirable energy level is detected: reduce the percentage of active nodes, remove node redundancy. -When an undesirable state linked to QoS is detected: increase the percentage of active nodes, increase node redundancy.
Executor tasks	-Update the data delivery interval -Node only with router/sensing functionalities, Turn off the node.	-Translate the adaptation plan in a readable format by the MicaZ/TinyOS platform, and disseminate the new network configuration.

Thus, the GatewayImpl class contains specific methods to communicate the SML with the NML, receive these TinyOS context messages and translate them in Java objects. AnalyzerImpl and PlannerImpl class contains methods to manage the requirements of the concrete middleware, detailed in Table 1, and adjust choose which nodes are able, in terms of energy, to publish services and attend application requirements.

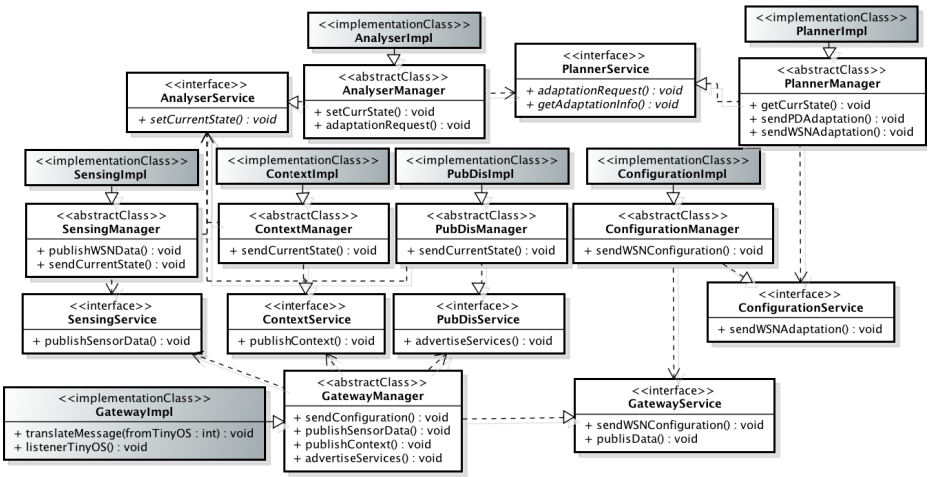

Fig. 3. Network MAPE-K Layer Instance

5 Final Remarks and Related Work

In this paper was proposed a reference architecture for self-adaptive service-oriented WSN in order to facilitate the building of middlewares capable of dynamically adapting to varying execution context. This reference architecture follows an autonomic computing model for making decisions aiming to attend self-adaptive WSN requirements.

Our reference architecture consists in two levels of autonomic management, one level of management inside sensor devices and the second level of management considers whole network. Both of them are based in the autonomic computing model MAPE-K. A case study was described to instantiate our proposed reference architecture in a specific energy-aware middleware, and showed how it supports designers of WSN middleware. In the current literature we found four different paradigms that allow building middleware systems for self-adaptive WSNs, namely component-based [10], service-oriented [11], multi-agent [12] and Software Product Line [13]. Our reference architecture adopts a component-based and service-oriented approach specific for self-adaptive WSNs.

Acknowledgments. This work is partially supported by FAPERJ, CNPq and CAPES.

References

1. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst. 4(2), 1–42 (2009)
2. IBM: An Architectural blueprint for autonomic computing, Autonomic Computing (2005)
3. Brun, Y., et al.: Engineering Self-Adaptive Systems through Feedback Loops. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Self-Adaptive Systems. LNCS, vol. 5525, pp. 48–70. Springer, Heidelberg (2009)
4. Wang, M., Cao, J., Li, J., et al.: Middleware for wireless sensor networks: A survey. Journal of Computer Science and Technology 23(3), 305–326 (2008)
5. Angelov, S., Grefen, P., Greefhorst, D.: A framework for analysis and design of software reference architectures. Information and Software Technology 54(4), 417–431 (2012)
6. Nakagawa, E.Y., Oquendo, F., Becker, M.: RAModel: A reference model of reference architectures. In: ECSA/WICSA 2012, Helsinki, Finland, pp. 297–301 (2012)
7. Delicato, F.C., Pires, P.F., Rezende, J., Pirmez, L.: Service Oriented Middleware for Wireless Sensor Networks. In: Editor (Ed.)^(Eds.): Book Service Oriented Middleware for Wireless Sensor Networks (2004)
8. Puviani, M., Cabri, G., Zambonelli, F.: A taxonomy of architectural patterns for self-adaptive systems. In: Proceedings of the International C* Conference on Computer Science and Software Engineering (C3S2E 2013), pp. 77–85. ACM, New York (2013)
9. nesC: A Programming Language for Deeply Networked Systems, `http://nescc.sourceforge.net/` (last access: April 2014)

10. Conan, D., Rouvoy, R., Seinturier, L.: Scalable processing of context information with COSMOS. In: Indulska, J., Raymond, K. (eds.) DAIS 2007. LNCS, vol. 4531, pp. 210–224. Springer, Heidelberg (2007)
11. Ruiz, L.B., Nogueira, J.M.S., Loureiro, A.A.F.: MANNA: A Management Architecture for Wireless Sensor Network. IEEE Commun Mag. 41(2), 116–125 (2003)
12. Fok, C.-L., Roman, G.-C., Lu, C.: Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. TAAS 4(3) (2009)
13. Gamez, N., Fuentes, L., Araguez, M.: Autonomic computing driven by feature models and architecture in FamiWare. Springer (2011)