A Self-Adaptive Context Processing Framework for Wireless Sensor Networks

Amirhosein Taherkordi, Romain Rouvoy, Quan Le-Trung, and Frank Eliassen
University of Oslo, Department of Informatics,
P.O.Box 1080 Blindern, 0314 Oslo, Norway
{amirhost, rouvoy, quanle, frank}@ifi.uio.no

ABSTRACT
Wireless sensor networks are increasingly being exploited in ubiquitous computing environments as one of the main platforms for gathering context data. In order to continuously observe the environment context during a long period, the sensor node should be considered itself as a context-aware device having particular contextual parameters, such as residual energy or sample rate. Existing work in the field of context-aware computing mostly considers the sensor node as a context data collector agent, regardless of the concern of the node's context elements. In this paper, we first propose an approach for modeling sensor network context information, and then, we introduce a middleware framework that maps our context model to software components, processes the context data, and implements the context model. For this purpose, we propose the notion of *context node*, which is the building block of our context processing framework. The proposed solution is exemplified in the shape of a home monitoring application. Using the proposed framework, the sensor application can adapt itself to the current situation in the environment through executing a high-level context model describing both the context information to process and the adaptation actions to perform.

Categories and Subject Descriptors
C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Distributed Applications*

General Terms
Performance, Design

Keywords
Wireless Sensor Networks, Context-awareness, Middleware, Adaptation

1. INTRODUCTION
Gradually, applications for *Wireless Sensor Networks* (WSNs) move beyond "sense and send" to pervasive computing environments, where a sensor node has tight interactions with actuators in the environment and behave depending on the context information surrounding it [1,11]. Applications for such environments must observe continuously their execution context in order to detect the conditions under which some behavioral adaptations are required. This execution context includes various categories of observable entities, such as sensing parameters, residual energy of node, sample rate, or user preferences. Interpretation of context data coming from these entities can be used for improving the execution performance, and adapting application behaviors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MidSens'08, December 1-5, 2008 Leuven, Belgium.
Copyright 2008 ACM 978-1-60558-366-2/08/12... $5.00.

The notion of *context* has been recognized as an important characteristic of ubiquitous computing environments, where a large number of autonomous agents work together to collect environmental information for smart and interactive devices [15]. Fundamentally, context is defined as "any information about the circumstances, objects, or conditions by which a user is surrounded and that is considered as relevant to the interaction between the user and the ubiquitous computing environment" [12].

According to what has been reported in the literature [3,8,13], in most of the context-aware systems, *sensor* is recognized as a context information provider agent. Depending on the type of possible activities in an environment, various sensor devices should be exploited. The c*ontext manager*, as a main part of such systems, is in charge of analyzing sensor data and identifying situations where application needs to be adapted. As an example, in a context-aware mobile application, the setting of a mobile device display depends on what is reported from the light detector sensors in the environment. However, the question that arises is whether the sensor node itself should be considered as an adaptable device having its own contextual parameters.

To make this issue more concrete, we consider a monitoring application for which sensor nodes with different capabilities may be deployed in an environment with changing parameters. In this case, we need to extract the different circumstances under which the application is running, process them, and deduce what should be performed in a particular situation. The problem becomes more complex when a lot of conditions come into the play; thereby inserting context management code in the application logic becomes an impractical solution. On the other side, most of the contextual logics are specified by the application user and may change over the application lifespan. Consequently, we face the challenges of *i)* how to model the flow of context information in a WSN application, and *ii)* how this application can be dynamically adapted based on the context model.

In this paper, we propose a context management framework in the middleware layer of WSNs to process context information and provide the necessary analyzed data for adaptation and reconfiguration tasks. Our work is inspired partially from the COSMOS framework—a comprehensive model for processing context information in ubiquitous computing environments [4,14].

Specifically in our proposal, each piece of context information is defined as a *context node*. Context nodes can be considered as virtual sensor nodes that are distributed over the physical sensor nodes in the network with respect to the type of information provided by the context node and its role in the context model. Besides, during their deployment, context nodes are mapped to the software components proposed specifically for context data processing in the WSN application. Therefore, the reification of context nodes as components at run-time provides support for the dynamic reconfiguration of sensor nodes whenever their execution context changes.

The rest of paper is organized as follows. In Section 2, we demonstrate a motivating application illustrating why context consideration is important in WSNs. The basic concepts of the context middleware are introduced in Section 3. The infrastructure of the proposed middleware and implementation issues are presented in Section 4. Next, in Section 5, the proposal is exemplified based on the motivation scenario described in Section 2. We discuss some related work in Section 6. Finally, Section 7 concludes this paper and identifies some future work.

2. MOTIVATING SCENARIO

As a motivating scenario, let us consider a *home-monitoring application*. Most of the earlier efforts in this field employed a high-cost wired platform for making the home a *smart environment* [9,10]. The emergence of wireless technologies and miniaturized devices make it possible to realize the same functionalities for a smart home more efficient in terms of deployment cost and deployment time. In particular, *sensor nodes* together with *actuators* are relevant technologies capable of monitoring various parameters, reasoning about the situation, and reacting to the processed information accordingly [1]. Thus, future home-monitoring applications are expected to be filled with different types of sensors that can provide various context information related to themselves, the inhabitants, and other appliances, such as occupancy, activity, and resource availability. These row of context data are fed to the context-aware home automation system.

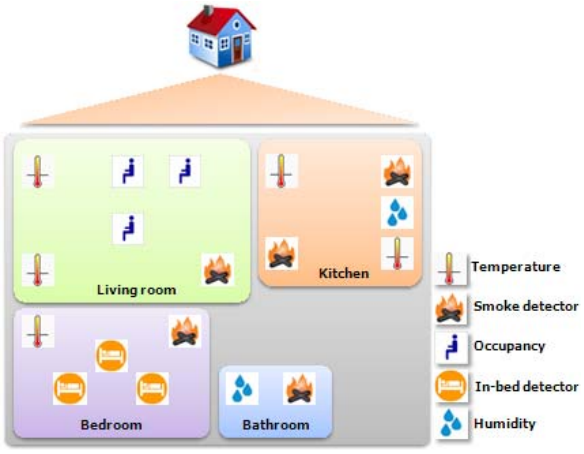

Figure 1. Description of the home monitoring system.

Figure 1 illustrates an hypothetical context-aware home. Rooms in the home are equipped with various sensor nodes. As shown, depending on the activities that can happen the relevant sensors are deployed, *e.g.*, in the living room three "occupancy" sensors detect movement, two sensors sense the temperature of the room, and one smoke detector sensor determines if a fire is present in the room. Obviously, sensor types and their placement in each room are determined according to the inhabitants' requests, potential home disasters, and technical issues.

The tight interaction of the sensor nodes and the environment on one side, and the inherent resource limitations of WSNs on the other side, complicate requirements analysis and their relations. In this way, adding contextual conditions to the application software in a hardcoded manner becomes an impractical solution. An important question arises on how we can model such information and build a reconfigurable application based on the model.

For this purpose, at first, we describe the basic concepts used in building a context model; next the approach for modeling context data will be explained.

3. CONCEPTS OF A CONTEXT MIDDLEWARE

Inspired by COSMOS [4], we introduce the architecture of context management in this section. Subsequently, the related concepts are explained. Figure 2 illustrates the overall architecture of a context information management framework. This architecture is divided into three layers: the Context Collector, Context Processing, and Context Adaptation layers. Similarly to COSMOS, each layer is organized into a 3-steps cycle of data collection, data interpretation, and situation identification.

The lower layer defines the notion of a Context Collector. Context collectors are software entities that provide raw data about the environment and sensor resources status. The Context collector also encompasses information coming from user preferences. The rationale for this choice is that context collectors should provide all the inputs needed to reason about the execution context. In our hierarchical architecture for WSN, the responsibility of context collectors is assigned to the sensors.

The middle layer defines the notion of Context Processing. Context processors filter and aggregate raw data coming from context collectors. The purpose is to compute some high-level, numerical or discrete, information about the execution environment. Data provided by context processors are fed into the adaptation layer. Intuitively, context processing tasks should be performed by the intermediate more powerful nodes in the hierarchical WSN architecture, namely, cluster heads and sink node.

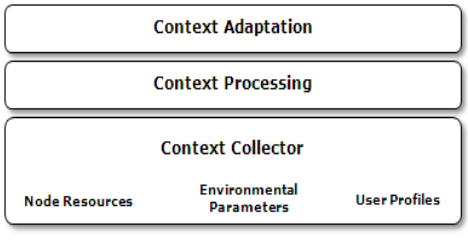

Figure 2. Architecture of a context management framework.

The upper layer is concerned with the process of decision making. The purpose is to be able to make a decision on whether or not an adaptation action should be performed. The Context Adaptation layer is thus a service that is provided to applications and that

encapsulates the situations identified by context nodes and processors. For example, in the motivating scenario, changing the behavior of temperature sensor in case of room occupancy is a decision made within this layer.

3.1 Architecture of a Context Node

The above architectural model defines the main aspects of a typical context management system. To exploit this model for WSN applications, each layer needs to be tailored according to the limitations and specifications of WSNs. In particular, for each layer it is necessary to extract the tasks and find the appropriate node in the network for performing these tasks.

We define the notion of *context node* as a representative of functionality in the context management architecture. In fact, a context node is the basic structuring of the architecture. A context node reifies particular context information. Context nodes are organized into hierarchies, which are compatible with hierarchies defined in WSN architectures. The graph of context nodes represents the set of context management policies associated to the application logic.

Thus, as illustrated in Figure 3, a *context node* interacts with other context nodes by exchanging *messages*, which encapsulate context information reports and are handled by the *message manager*. The context node can be either *active* or *passive*. A passive node obtains messages upon demand, while an active node gathers periodically messages via the *activity manager*. The *context processor* is responsible for processing the received messages into context information of higher-level of abstraction and can, eventually, operate some functional or non-functional *actions* on the enclosed context nodes. These actions are planned and executed by a *context reasoner* and a *context configurator*, respectively. The context model we define supports the sharing of *Context Reasoner* and *Configurator* as well as *Activity* and *Message Managers* across collocated context nodes in order to reduce the memory footprint and the resource consumption of the sensor nodes.

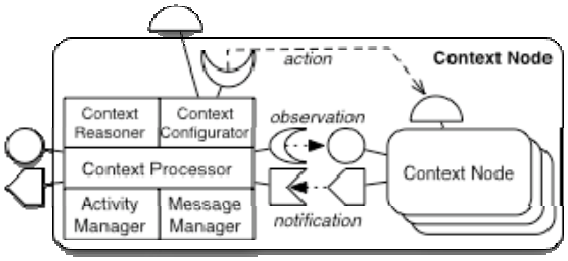

Figure 3. Architecture of a context node.

As an example from the motivating scenario, the temperature sensors deployed in the room enclose four context nodes (cf. Figure 4). At the leaf level, a context node encapsulates the hardware sensor and converts raw data into the context information reports. This report covers not only temperature reports, but also clock and energy information. The current temperature and the energy left are extracted by two other context nodes (meaning that a context node can be shared among other context nodes). The energy status is then processed by a fourth context node in order to be compared to a given threshold that determines if the action of reconfiguring the sample rate should be taken or not. The current temperature is notified to the context node of the parent domain (the room is this case) in order to be processed.

3.2 Composition of Context Nodes

Each context node is placed in one of the layers of the context processing architecture based on its responsibility. The context model we define organizes the collaboration between context nodes.

Figure 4 illustrates the context model of a dynamic home-monitoring system. Hardware sensors deployed in the rooms produce *context information* that is continuously processed to identify potential *actions* to execute. These actions can be functional (*e.g.*, activation of the alarm bell) or non-functional (*e.g.*, reconfiguration of a sensor sample rate). The context information is propagated within the monitoring system proactively or reactively via *observation* and *notification* mechanisms, respectively. *Context domains* identify context nodes that can be collocated in order to reduce the cost of communication and thus react autonomously (without relying on a centralized architecture).

A context node in the context model can be shared between other nodes. The sharing of a context node corresponds to the sharing of a context management policy. Nodes in the lower level of the hierarchy are more likely to be shared. Such nodes are more expected to collect context data, rather than performing any action. For instance, the context node providing raw information about the temperature—*i.e.*, **Temperature Sensor**—falls in this category.

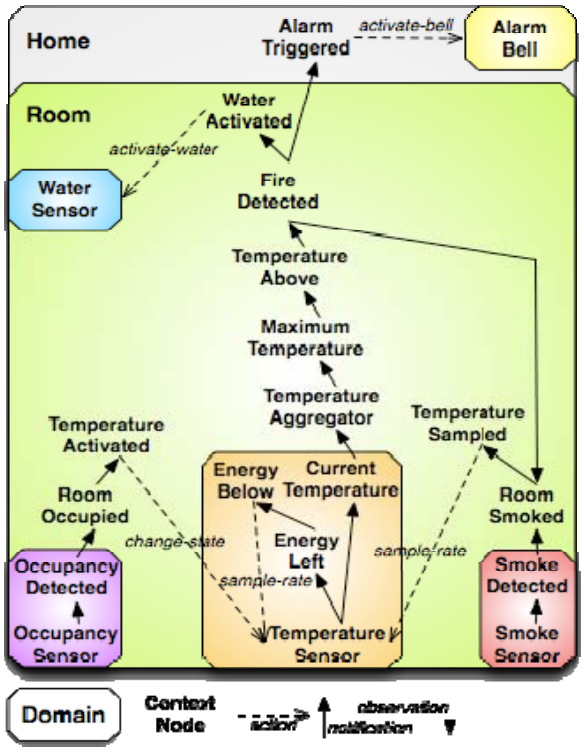

Figure 4. Context model of the monitoring system.

Our proposal for context model is more in accordance with hierarchical architectures for WSNs. The frequent reference to

this model in WSN challenges makes the proposed context model a reference model for addressing context management issues for WSNs. Note that like hierarchical WSNs architectures, our context model does not impose any limitation on the depth of the hierarchy.

In addition to the propagation of context information, the context nodes can embed some autonomic behavior to react to context changes. In fact, context nodes basically provide the raw data for identifying the type of adaptation required. Like the design model of the context node, the adaptation model is expected to follow the component model in order to have a unified approach for addressing self-* issues. This feature enables the context model to be self-adaptive and thus adapts the context information retrieval characteristics depending on the context currently manipulated.

4. IMPLEMENTATION OF A CONTEXT MIDDLEWARE

In this section, we put the concepts mentioned in the previous section all together and describe the cycle of context management first. Next, the infrastructure of a middleware facilitating the context management tasks will be presented.

Context management encompasses all activities required to reach a context-aware application. The major phases of management can be categorized as *i)* classification of context information for a particular computing environment, *ii)* finding the relation between the classified information, *iii)* extraction of context elements (context nodes in our scope), *iv)* putting context elements together in an interaction model, *v)* mapping the context model to the platform supporting context elements definition and interaction, *vi)* iterating these steps for the possible future needs.

Context management frameworks have tried to span as much as possible of the above tasks, although the main focus has been on the modeling. Apart from concerns directly related to each step, the underlying platform brings its own challenges and concerns. The limitations and characteristics of a particular platform specialize some steps of context management. Particularly, context modeling and execution need to be considered precisely with respect to the target platform.

Moreover, frameworks for context execution have been a challenging issue in terms of the degree of generality, implementation techniques, and non-functional features, such as scalability. Proposals target middleware layer solutions to address such concerns. A middleware solution can simplify the task of maintaining context-aware systems and provide common services for context management. It supports a common model for context data definition and also it is able to maintain the context model dynamically.

Our proposal relies on a component-based middleware for providing context information in WSN applications. Thus, we have investigated several component models [2,6,7] to identify features suitable for our context model. Based on this we extract a fundamental characteristic of state-of-the-art component model: A *lightweight hierarchical* component model that stresses on *modularity* and *extensibility*. It allows the definition, configuration, dynamic reconfiguration, and clear separation of functional and non-functional concerns. The central role is played by interfaces, which can be either *functional* or *control*. Whereas functional interfaces provide external access point to components, control interfaces are in charge of non-functional properties of the component (*e.g.*, life-cycle management or binding management). Components are sometimes divided into *passive* and *active*. Whereas passive components generally represent services, active components contain their own thread of control.

Based on these assumptions, a context node is implemented as a composite component that contains at least five primitive components: the context processor, the context reasoner, the context configurator, the activity manager, and the message manager. The context node's dependencies are enclosed as components as well. The observation and notification mechanisms are implemented by functional interfaces, while the support for reconfiguration is provided by the control interfaces.

The middleware run-time system is composed of two major parts: core services and core context components. Core services are dedicated for maintaining the context model, *e.g., communication service* for making the interaction of context nodes located in the different nodes. There are also some context nodes that are frequently used in the context models, *e.g,* ResidualEnergy. The middleware is equipped with such core context components as well.

Moreover, the middleware is in charge of maintaining the context model. Particularly, the middleware run-time system takes care of managing context nodes and their interactions according to the context model description. Figure 5 illustrates how the context model is mapped to the context components. Each context component represents a portion of context model (one context node or more context nodes and their interactions). The logic behind the context model specifies which portion of model should be mapped to a particular context component. Based on this knowledge, the middleware will be able to deploy context components over the sensor nodes, handle local and remote interactions of context components, and provide the run-time system for context-aware application execution.

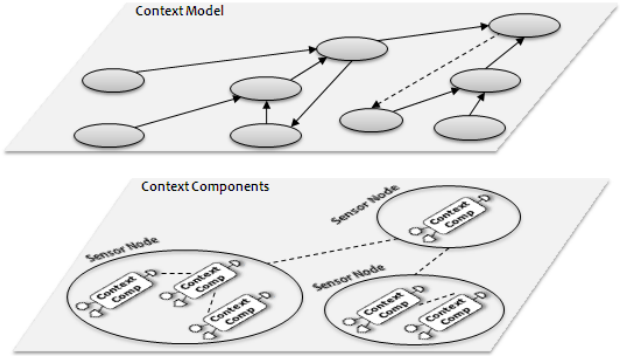

Figure 5. Mapping of the context model to the context components.

5. SAMPLE SCENARIO EXECUTION

Let us again refer to the sample home-monitoring application and consider how the context components and their interactions can be obtained from the context model. As mentioned before, we propose the context domain for identifying collocated context nodes in order to process information locally. In fact, each context domain represents a physical node in the system. The most inner domains represent the sensor nodes and domains encompassing them are cluster heads, while the outer domain surrounds all clusters context information.

Figure 6 presents the context components and their interactions for a temperature sensor, occupancy sensor, and cluster head. The `TemperatureSensor` component provides two kinds of contextual information: *current temperature* and *residual energy* of the sensor node (see Figure 4). For the former one, a context component reading the current environment temperature is deployed. All temperature sensors inside the room send the output of this component to the temperature aggregator component located in the cluster head. The latter information is provided by the `EnergyLeft` component. In case of reaching a threshold of residual energy, the `EnergyBelow` component calls the `sampleRateControl` function from `TemperatureSensor` to reduce the sample rate. `OccupancySensor` component running on the occupancy sensors notifies the `RoomOccupied` component in the cluster head in case of detecting any movement. `TemperatureActivation` component detects room occupancy. Upon receiving any notification from `RoomOccupied` component it calls the `changeState` function of `TemperatureSensors` and makes them silent while receiving occupancy notification.

The interactions of components located in a node are established via local message passing. As of context node architecture, the message manager in each component receives the context information, processes it, and delivers the result to the context processor part. The context components interacting directly to each other are expected to know the format of message passed among them. Likewise, components running in different nodes interact via message passing, however, in this case messages are sent over the data distribution protocol of network.

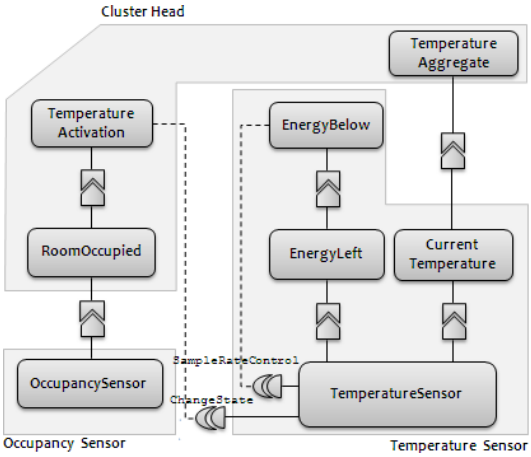

Figure 6. Context components composition for home monitoring.

The middleware run-time system is in charge of connecting context components via message passing according to the context model description. The middleware *communication service* also encapsulates the remote message and sends it to the relevant node within the network. Similarly, in the target node, this service reads the message and delivers it to the relevant context component for further processing.

Figure 6 presents just a small part of managing the home context information. In fact, this model is part of a larger model that encompasses all context elements. The middleware supporting such model should be able to manage the context model in a scalable fashion. Maintaining the context model in a particular sensor node needs just the *meta-context information* related to that node including *i)* the context components that should be deployed within that node, and *ii)* the details of local and remote interactions of those components. In our proposal, the middleware in each sensor node is equipped with such metadata. In this way, the large amount of meta-context information is split among sensor nodes according to their role in the context model execution.

6. RELATED WORK

As mentioned at the beginning of this paper, most of the context processing work employs the sensor node just as a context data collector, whereas the concern of context awareness for WSN application is a separate topic indicating how the WSN itself can be managed based on its own context data.

The first relevant work has been reported by Huebscher et al [8]. They propose an adaptive middleware for context-aware application that abstract the applications from the sensors that provide context data. The proposed middleware is a layered architecture for context provision. In this architecture sensors are placed at the bottom as raw context data provider. In the one upper level, a network of context services is located. The main contribution of this work is to choose the best set of available context services in order to satisfy the context-aware application requirements. The motivation for this work is a home automation application for smart appliances in the home equipped with sensors. The sensors in such a smart-home are not necessarily networked, but play as an isolated agent for gathering the context data. However, in our proposal a sensor is characterized as an autonomous node of a network capable to adapt itself against the environmental and sensory context data.

Some work uses the context-aware concept as a technique for conserving energy in WSNs. [3] proposes a framework for supporting the use of context to trigger power saving functionalities in the sensors—*i.e.*, controlling the behavior of the sensors. For this purpose, they propose *context discovery* to discover useful contexts from sensor data, and *context-trigger engine* to use the discovered context as a trigger. In this work, two centralized databases are proposed for indentifying context information and context actions. This work is suitable for applications with simple isolated context information. However, the overhead of centralized observation method in this framework may compromise the improvement claimed in the context-based energy conservation.

TeenyLime is the recent reported middleware solution for dealing with the complexity of specifying how multiple tasks coordinate to accomplish a functionality in the wireless sensor and actor networks [5]. TeenyLIME operates by distributing the tuple space among the devices, transiently sharing the tuple spaces contents as connectivity allows, and describing reactive operations that fire when data matching a template appears in the tuple space. In fact, TeenyLIME system provides a single unified abstraction for representing both the application and system context. TeenyLime can takes part in our model as a context processor if the message format, adopted for context nodes interactions, conforms to the tuple structure.

7. CONCLUSION AND FUTURE WORK

Using WSNs in autonomous environments makes the application design process more challenging. This process is based on

gathering, analyzing and treating large amount of context data produced by the environment and sensor nodes, and then adapting and configuring application based on this data.

In this paper, we presented an approach for addressing context related concerns in the WSNs. Mainly, our approach is composed of two parts *i)* a context information processing architecture for modeling the sensor context elements and their interactions, and *ii)* a middleware framework for executing the context model. The basic structuring concept of our proposal is the *context node*. In fact, a context node is context information modeled by a context component performing context execution tasks (context processing, context reasoning, and context configuration). Context components are distributed over the network according to the context model description. The underlying middleware run-time system maintains the model and provides a container for context execution. Using this middleware, a WSN application can adapt itself to the current situation in the environment through executing a high-level context model describing the context information and the adaptation actions corresponding to the context.

We are currently focusing on the home-monitoring application as a motivating scenario. In this paper, we described this application briefly and discussed how its context model can be obtained and also the equivalent context components composition for a part of context model was illustrated at the end of paper.

This work will be continued along two main axes. First, the initial version of middleware framework will be developed based on the requirements mentioned in this paper. The run-time middleware system should be equipped with services for facilitating context data processing, reasoning and configuration, besides the services for supporting context model execution. Second, the middleware system will be evaluated by developing the context-aware home monitoring application. Also, the performance of the middleware will be evaluated in terms of memory occupation, context model execution processing overhead, and energy consumption.

8. REFERENCES

[1] Akyildiz, I., Kasimoglu, I., 2004. Wireless sensor and actor networks: Research challenges", Ad Hoc Networks 2 (4), pp. 351-367.

[2] Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.-B., 2006. The FRACTAL component model and its support in Java. Softw., Pract. Exper. 36(11-12): 1257-1284, http://fractal.objectweb.org

[3] Chong, S, Krishnaswamy, S., and Loke, W., 2005. A context-aware approach to conserving energy in wireless sensor networks, In Proceedings of the 3rd Int'l Conf. on Pervasive Computing and Communications Workshops (PerCom'05), Kauai Island, HI, USA.

[4] Conan, D., Rouvoy, R., and Seinturier, L., 2007. Scalable processing of context information with COSMOS. In 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'07). p. 210–224 of LNCS 4531 (Springer). Paphos, Cyprus. June 5–8.

[5] Costa, P., Mottola, L., Murphy, A., Picco, G., 2007. Programming wireless sensor networks with the TeenyLime middleware. In Proceedings of the Middleware Conference pp.429-449. Vol. 4834 of LNCS. Springer.

[6] Coulson, G. et al., 2008. A generic component model for building systems software. ACM Trans. Computer Systems, 1-42.

[7] Crnkovic, I., Larsson, M., 2002. Building reliable component-based software systems**.** Artech House.

[8] Huebscher, M. C., and McCann, J. A. 2004. Adaptive middleware for context-aware applications in smart homes, 2nd Workshop on Middleware for Pervasive and AdHoc Computing.

[9] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J., 2002. Wireless sensor networks for habitat monitoring. In: Proc. of First ACM Workshop on Wireless Sensor Networks and Applications (WSNA).

[10] Mozer, M., 2004. Lessons from an adaptive home. In: D.J. Cook and S.K. Das, Editors, Smart Environments: Technology, Protocols, and Applications, Wiley, 273-298.

[11] Puccinelli, D., and Haenggi, M. 2005. Wireless sensor networks: applications and challenges of ubiquitous sensing. IEEE Circuits and Systems Magazine, vol. 5, no. 3, 19-31.

[12] Ranganathan, A., and Campbell, R, 2003. A Middleware for context-aware agents in ubiquitous computing environments. In: CM/IFIP/USENIX International Middleware Conference, Brazil.

[13] Rocha, R., and Endler, M., 2006. Context management in heterogeneous, evolving ubiquitous environments, IEEE Distributed Systems Online.

[14] Rouvoy, R., Conan, D., Seinturier, L., 2008. Software architecture patterns for a context-processing middleware framework. In IEEE Distributed Systems Online (DSO), vol. 9, no. 6.

[15] Yau, S., Karim, F., Wang, Y., Wang, B., Gupta, S., 2002. Reconfigurable context-sensitive middleware for pervasive Computing. IEEE Pervasive Computing 1(3), 33–40.