Edwin Rajesh.A¹, Dr. Ponmuthuralingam, P²  
Assistant Professor, Department of Computer Science, Bishop Appasamy College of Arts & Science, Race Course Road, Coimbatore, India¹  
Research Supervisor, Professor and COE, Govt. Arts College (Autonomous), Govt. Arts College Road, Coimbatore, India²  

Abstract: Wireless sensor networks are widely being explored in pervasive computing environments used as source of gathering context data. The continuous observation of 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 elements or sample rate. Existing method of context-aware computing mostly considers the sensor node as a context data collector agent without the concern of the node’s context elements. In this paper, we propose an execution approach for modeling sensor network context information, and we introduce a middleware framework that maps the context model to software elements, processes the context data, and implements the context model. For this purpose, we propose the concept of context node, which is the building block of our context processing framework. By 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.

Keywords: WSN, Context Data, Context Node  

1. INTRODUCTION: Wireless Sensor Networks (WSNs) move beyond “sense and send” to pervasive computing environments, where a sensor node has tight coupling with actuators in the environment and responds depending on the context information surrounding it. Applications for such environments must monitored continuously for their execution context in order to detect the conditions under which some behavioral inheritance are required. This execution context includes various categories of monitoring entities, such as sensing range, un consumed energy of node, sample rate, or user preferences. Interpretation of context data coming from these units can be used for improving the execution performance, and adapting application behaviors. The concept of context has been recognized as an important characteristic of universal computing environments, where a large number of self-determining agents work together to collect environmental information for smart and interactive devices. Basically, context is defined as “any information about the surroundings, objects, or conditions by which a user is pretended and that is considered as relevant to the interaction between the user and the universal computing environment.” The context manager, as a core part of such systems, is in charge of verifying sensor data and identifying situations where application needs to be adapted. To make this issue more strong, we reflect a monitoring application for which sensor nodes with different possibilities may be implemented in an environment with changing limitation. In this case, we need to extract the different conditions under which the application is running, process them, and figure out what should be performed in a particular situation. The problem becomes more complex when a lot of conditions come into the perform; thereby inserting context management code in the application logic becomes an impractical solution. Consequently, we face the challenges of how to model the flow of context information in a WSN application, and 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.

2. Concepts of a Context Middleware  
To exemplify the overall architecture of a context management and information management framework. This architecture is divided into three layers: the Context Collector, Context Processing, and Context Adaptation layers.

![Content Management Framework Architecture](image)

Fig.2 Content Management Framework Architecture  
The lower layer defines the preception of a Context Collector. Context collectors are software entities that
provide unpracticed data about the environment and sensor resources status. The context collector also incorporate information coming from user preferences. In our hierarchical architecture for WSN, the responsibility of context collectors is assigned to the sensor. The middle layer defines the preception 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 passed into the adaptation layer. Inherently, context processing tasks should be performed by the intermediate more powerful nodes in the hierarchical WSN architecture, namely, cluster heads and sink node. The upper layer is concerned with the process of decision making. The purpose is to be able to make a decision on adaptation action should be performed or not. The Context Adaptation layer provides a service which is been fed in to the applications and that encapsulates the conditions identified by context nodes and processors.

3. Architecture of a Context Node
The above architectural model defines the main aspects of a typical context management system. To utilize this model for WSN applications, each layer needs to be personalized according to the parameters and specifications of WSNs. In particular, for each layer it is necessary to root out the tasks and find the appropriate node in the network for performing these tasks. We define the conception 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 receives a particular context information. Context nodes are organized into hierarchies, which are compatible with hierarchies defined in WSN architectures. Thus, as illustrated in below figure (Dig.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 reasoned and a context configuration, respectively. The context model we define supports the sharing of Context Reasoned and Configuration 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.

At the leaf level, a context node encapsulates the hardware sensor and converts raw data into the context information reports.

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) Extracting context elements
ii) Putting context elements together in an interaction model.
iii) Mapping the context model to the platform supporting context elements definition and interaction.
iv) 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. It allows the definition, configuration, dynamic reconfiguration, and clear separation of functional and non-functional concerns.

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., Residual Energy. 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 3: Context Node Architecture](image)

![Figure 4: Mapping of the context model to the context components](image)
(Fig:4) 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.

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 lumped 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 encircling them are cluster heads, while the outer domain surrounds all clusters context information. (Fig.4) presents the context components and their interactions for a temperature sensor, occupancy sensor, and cluster head. The Temperature Sensor component provides two kinds of contextual information: current temperature and residual energy of the sensor node. 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 Energy Left component. In case of reaching a threshold of residual energy, the Energy Below component calls the sample Rate Control function from Temperature Sensor to reduce the sample rate. Occupancy Sensor component running on the occupancy sensors notifies the Room Occupied component in the cluster head in case of detecting any movement. Temperature Activation component detects room occupancy. Upon receiving any notification from Room Occupied component it calls the change State function of Temperature Sensors and makes them silent while receiving occupancy notification.

![Fig. 4: Context components composition for home monitoring.](image)

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. 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.

6. Conclusions 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 runtime 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.
References


