Towards the Design of a Component-based Context-Aware Framework for Wireless Sensor Networks

Manik Gupta
School of Electronic Engineering and Computer
Queen Mary University of London
London, UK
e-mail: manik.gupta@eecs.qmul.ac.uk

Eliane Bodanese
School of Electronic Engineering and Computer
Queen Mary University of London
London, UK
e-mail: eliane.bodanese@eecs.qmul.ac.uk

***Abstract*—Context-awareness is one of the prerequisites in order to design adaptable applications and services. As Wireless Sensor Networks get more pervasive in nature and cater to diverse application they need to incorporate context-awareness. A lightweight sensor node level context processing framework is desirable. A component software programming-based approach has been proposed to design the context-aware architecture. A reference implementation has been provided and potential advantages in terms of low software reconfiguration overhead have been highlighted.**

Keywords-context-awareness; component-based software; middleware;adaptive sampling.

I. INTRODUCTION

Context-awareness is important for Wireless Sensor Networks (WSNs) since there are several WSN applications that need to make decisions on the basis of the prevailing contextual environment. For the current work, air pollution monitoring for urban streets application is being worked upon and the sensor nodes need to provide fine grained resolution pollution measurements for variables like carbon monoxide, etc. The sensor nodes can also provide support for measuring variables like temperature, air pressure, humidity, wind speed, location etc. These additional variables can be used as contexts for adapting the sensor application. An adaptive sampling application for changing the sampling rate of the sensor nodes is an example of such a context-aware application; wherein the sampling rate needs to be adapted as and when sensor nodes undergo certain environmental changes using an intelligent algorithm.

Hence, as the WSNs get more pervasive in nature and need to incorporate more intelligence in order to facilitate node level decision making, a generic lightweight context-aware framework design and implementation for the wireless sensor nodes becomes an imperative need to design smarter and adaptable applications. One of the challenges to be addressed is to make the framework customizable, allowing runtime reprogramming and dynamic reconfiguration, in order to address the application and environmental diversity, to which the WSNs generally cater.

Recently, component-based software development [1][2][3] has been advocated for WSN programming. Software componentization provides a high level programming abstraction through interface-based interactions between modules. A component represents a single unit of functionality and deployment. Components can be compiled separately and then composed into a system. A component can communicate with its outside through a well-defined interface and receptacle. An interface defines a set of operations provided by a component to others, while a receptacle specifies the set of operations a component requires from others. The system can be reconfigured by switching from an old component to a new one implementing the same interfaces.

Dynamically deployable and reconfigurable software components offer a lot of advantages for WSN since they are typically large in scale and operate for long periods of time in the face of dynamic environmental conditions and changing application requirements. Software development becomes easier since during the development cycle, changes to only one part of the system (single algorithm or function) are done at a time, so the rest of the software remains unaffected. Software updates become very convenient since only the updated components need to be transmitted through the network. It is possible to implement dynamic and reconfigurable applications since the functionality can be altered by reprogramming only parts of the system.

Hence, the main contribution of the paper is to propose a component-based context-aware architecture for sensor nodes. This framework bridges the gap between component-based software and context-awareness for WSNs by enabling a lightweight solution for context management. Though the use of component-based software development in the WSN field is not new at all, but the majority of the sensor nodes today are not capable of managing the contexts (other than merely acting as a context collector). This sensor node-based framework will aid not only in collection, but also in processing and use of the contexts for decision making. In a traditional context-aware application where all the contexts are collected and processed in a single monolithic application, a change or modification related to one of the contexts will lead to the complete application

 ISBN: 978-1-61208-207-3

change. On the contrary, in case of the component-based context-aware application design, only the context which needs to be changed can be updated by means of incorporating changes to that particular component. The specific requirement to make sensor nodes smarter and more practical by means of managing multiple contexts in an online manner is the main driver behind the proposed architecture.

Rest of the paper is structured as follows. Section II gives details about various context-aware frameworks and component-based middleware and software systems available for WSNs. Section III gives details about architectural design of the component-based context-aware framework. Section IV gives the reference implementation details and Section V draws a conclusion on the paper.

II. RELATED WORK

Context-awareness has been acknowledged as a very important step in ubiquitous computing and there are a lot of context-aware systems which exist in the literature [4][5][6]. Most of these context-aware systems typically cater to the needs of pervasive computing, but are not suitable for sensor node level context processing in WSN. Most of these systems are heavyweight and need back end processing. Given the nature of WSNs, a context-aware framework for WSNs cannot be heavyweight and processing/memory intensive. Therefore, new designs are required which are lightweight and are suitable for the needs of WSN applications.

Hence an investigation into component-based software paradigm for wireless sensor networks was carried out. Both component-based middleware solutions for WSN as well as component-based software reconfiguration and operating system exist for WSNs in the literature. RUNES [7], GridKit [8] and LooCI [9] are examples of component-based middleware's for WSN, but they do not address context-awareness specifically. RUNES and GridKit are more suitable for network level reconfiguration, while Java-based implementation of LooCI limits its usage. On the contrary, in this research work, the aim is to develop a context-aware framework built using a lightweight component middleware which can work on resource constrained nodes. Wisekit [10] is the only node level component-based middleware solution for WSNs found in literature that addresses context-awareness by providing application adaptation. It is a distributed component-based middleware solution which addresses context-awareness by making the adaptation and reconfiguration of WSN applications possible, but actual implementation details are not mentioned in the paper. Figaro [11] for fine grained software reconfiguration and Lorien [12], dynamic component-based operating system are examples of use of component software paradigm in WSNs, but again, they do not address context-awareness specifically.

Based on the literature survey on both the context-aware framework and component-based software for WSNs, it was found from the survey that a gap exists in these two domains in terms of integration by means of developing a component-based context-aware framework. Hence, in the current work, it has been proposed to adopt the component-based software programming paradigm for the development of a lightweight and reconfigurable context-aware framework. The architecture design has been explained in the following section.

III. DESIGN OF THE COMPONENT-BASED CONTEXT-AWARE FRAMEWORK

In this section, the design architecture for a component-based context-aware framework has been proposed. The context-aware framework needs to be built using the services of a component-based middleware. A component-based middleware called MIREA [13], developed at UCL, and has been used in this research work to build the context-aware framework. MIREA is specifically targeted at real-time embedded systems. MIREA is light-weight, component-based, and supports flexible reusability of software components. The aim of building the context-aware framework on top of the component-based middleware in this research work has been to lend adaptability at the middleware level that in turn will lead to adaptation at the application level.

A component can specify functionality that it requires and/or provides using a well-defined interface. The model as shown in Fig. 1 consists of *ComponentTypes*, *Components*, *Interfaces*, *Receptacles*, and *Connectors*. A *Component* is a runtime instance of a ComponentType. A *ComponentType* can export one or more *Interfaces*, through which a given component provides a set of functionalities to other components (i.e., in the form of a set of C/C++ functions in the middleware). A *Component* can have any number of *Receptacles*, through which a set of required functionalities are specified. A component can also have an associated component wide state that is only accessible from within the containing component. *Connectors* are a specialized form of component that performs intermediary actions if required, for instance, in order to monitor, log or encrypt data for security reasons.

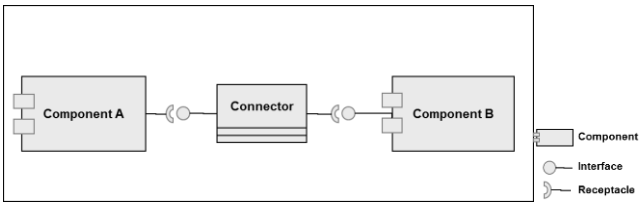

Figure 1. Elements of a component-based system.

MIREA provides the following categories of core services:

1. Loading and unloading of *ComponentTypes*
2. Instantiation and destruction of *Components*

3. Registration and acquisition of *Interfaces* and *Receptacles*
4. Connection between *Interfaces* and *Receptacles*
5. Registration and acquisition of Components' *States*
6. Destruction of *Connectors*

All of the services above are runtime activities whereas the process of defining a new *ComponentType*, *Interface*, *Receptacle*, *Connector* and *State* are static in nature - defined at an application design stage. Connections between components can be reset and reconfigured by first destroying an existing connector instance and then reconnecting them to a new type of interface and receptacle. After this, any invocations made on the given *Receptacle* will be redirected to the newly connected *Interface* instance, hence a new/different *Component* instance. More details about MIREA can be found in [13].

Most of the context-aware systems have a generic architecture consisting of context collector, context reasoner and a context database. In the component-based architecture, each of these tasks is going to be performed by an individual standalone component that will have a well-defined interface for interaction with the other components. These reusable components can be loaded and unloaded during runtime. Also, once loaded, the components can be instantiated at run time and their respective interfaces and receptacles can be registered and connected to each other using the MIREA API's. The architecture of the component-based context-aware framework has been shown in the Fig. 2.

The various components and their respective functionality in this architecture are explained as follows:

- Sensor Manager will be responsible for interacting with the physical sensors and will hide the hardware specific details from the application. The context collector will invoke the sensor manager for data from a particular sensor depending on a particular application and the sensor manager will provide the data to the context collector.
- Context Collector will be the main component responsible for collecting the application specific contexts and use interfaces provided by other components to gather data from sensors and map it into contextual information, store data in buffers and provide contextual data to the context reasoner.
- Context Database provides storage service for the various contexts and depending on the application requirement, the data can either be offloaded to the base station or be stored in external storage available on the sensor node.
- Context Reasoner is the component responsible for carrying out the reasoning over the various collected contexts and make adaptive decisions according to the application requirements. This component will also be responsible for performing data processing tasks like prediction or clustering analysis of the gathered contexts in order to facilitate the decision making process.

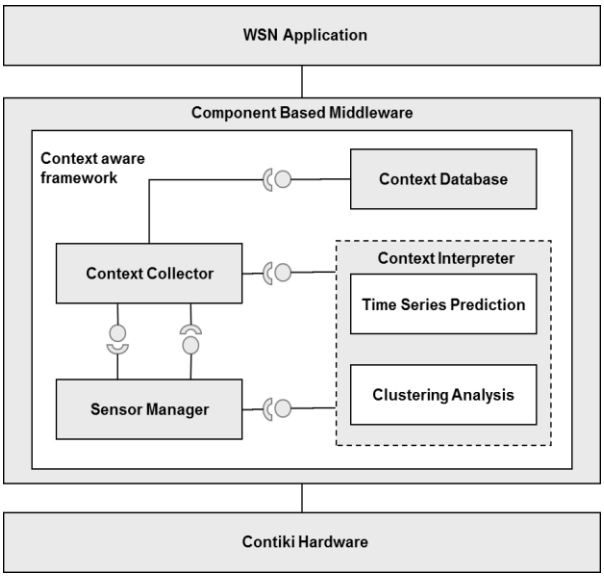

Figure 2. Architecture of component-based context-aware framework.

IV. REFERENCE IMPLEMENTATION

The reference implementation for an adaptive sampling based data collection application is shown in the Fig. 3, which defines the various interfaces/receptacles between the various components. The adaptive sampling technique used in the reference implementation has been proposed as a part of on-going research work and more details can be found in [14]. The adaptive sampling technique based upon time series forecasting can adapt the sampling rate of a sensor node according to the prevailing contextual environment.

The implementation has been done using the Contiki [15] operating system on the T-mote sky platform. Contiki was chosen as the operating system of choice because of its support for dynamic loading and linking of loadable software modules [16]. Each loadable module in Contiki is in Compact Executable and Linkable format (CELF) format which is a modification of the common object code format , Executable and Linkable format (ELF). The dynamic loader/linker (elfloader) in Contiki parses the ELF format and is able to perform dynamic loading, linking and relocation of ELF object code files. Initially the components can be stored in the external EEPROM by programming either using the serial interface or over the air programming. The various steps involved in implementing the context-aware framework on the T-mote Sky are as follows:

- The MIREA middleware was ported onto Contiki.
- The components were implemented and built using the platform specific compiler.

 ISBN: 978-1-61208-207-3

- The components are loaded on the external flash of the T-mote sky using the serial interface.
- The MIREA application driver program loads, connects and unloads the components using the MIREA API's.

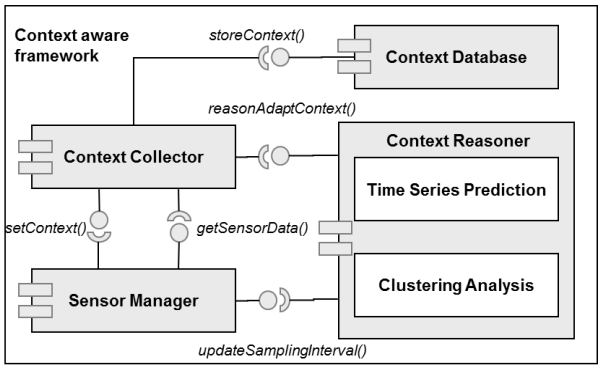

Figure 3. Interface/receptacle definition for an adaptive sampling application.

The memory footprints for each of the components (w.r.t. both program and data memory) for T-mote sky are shown in Table I. The memory footprints have been evaluated by looking at the size of binary images after compilation. The *text* section represents the code size, *data* section is the size of initialized memory, and *bss* section is the size of uninitialized memory.

TABLE I. MEMORY FOOTPRINTS FOR VARIOUS COMPONENTS

Component Name	Memory Footprints		
	Text(bytes)	***Data(bytes)***	***BSS(bytes)***
sensorManager	554	10	12
contextCollector	632	10	0
contextDatabase	256	10	0
contextReasoner	984	22	14

Another experiment was carried out to compare the Contiki image size with context-aware framework implementation as a single software module (monolithic context-aware framework) vs. Contiki image with MIREA implementation and the results are shown in Table II In case of monolithic implementation, once the context-aware framework is programmed on the node, it is difficult to reconfigure and update the software program. Any software reconfiguration would require communicating full application image of ~23K bytes to the node. On the contrary, in case of the component-based implementation, once the node is programmed with MIREA middleware that occupies ~33K bytes, less than ~1K bytes (refer to Table I) need to be transmitted to the node to enable component runtime reconfiguration or reprogramming.

TABLE II. COMPARISON OF MEMORY FOOTPRINTS FOR DIFFERENT IMPLEMENTATIONS

Different Implementations	Memory Footprints		
	Text(bytes)	***Data(bytes)***	***BSS(bytes)***
Contiki image with monolithic context-aware framework	23054	178	5164
Contiki image with MIREA middleware	32936	170	7658

V. CONCLUSIONS

In this paper, a design of a component-based context-aware framework has been proposed for WSN. This framework provides integration between the component software and context-awareness technologies for the WSNs. This design architecture has several advantages in terms of ease of programming, software updates and reconfiguration, dynamic application development etc. This architecture is lightweight in nature and suitable for sensor node level context-aware processing. Most of the context-aware systems existing in the literature do not cater to the needs of sensor nodes and are architecturally and functionally more sophisticated and heavyweight in nature. The component-based software technology is suitable for programming context-aware WSN applications and a proof of concept reference implementation using the Contiki OS that enables dynamic linking/loading of software components has been carried out.

ACKNOWLEDGMENT

The authors would specially like to thank our partners, Prof. Steve Hailes and Dr. Jagun Kwon from University College London for providing their full support and help on the MIREA middleware. This project was funded by India-UK Advanced Technology Centre (IUATC).

REFERENCES

[1] Karl H. Johnasson, John Lygeros, Anthony Tzes, Karl-Erik Arzen, Antonio Bicchi, Gianluca Dini, Stephen Hailes, A component-based approach to the design of networked control systems. *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*.

[2] Barry Porter, Utz Roedig, Francois Taiani, Geoff Coulson, A comparison of static and dynamic component models for Wireless Sensor Networks. *Computing, 224460.*

[3] Geoff Coulson, Gordon Blair, Paul Grace, Francois Taiani, Ackbar Joolia, Kevin Lee, Jo Ueyama, Thirunavukkarasu Sivaharan, A generic component model for building systems software. *ACM Trans. Comput. Syst.* 26, 1, Article 1 (March 2008), 42 pages.

[4] Daniel Salber, Anind K. Dey, Gregory D. Abowd, The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit* (CHI '99). ACM, New York, NY, USA, 434-441.

[5] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, Klara Nahrstedt, A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing* 1, 4 (October 2002), 74-83.

 ISBN: 978-1-61208-207-3

[6] Tao Gu, Hung Keng Pung, Da Qing Zhang, A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl.* 28, 1 (January 2005), 1-18.

[7] Paolo Costa, Geoff Coulson, Cecilia Mascolo, Gian Pietro Picco, Stefano Zachariadis, The RUNES middleware: a reconfigurable component-based approach to networked embedded systems, In *Proceedings of the 16th Annual IEEE International Symposium Personal, Indoor and Mobile Radio Communications, 2005*.

[8] Paul Grace, Geoff Coulson, Gordon Blair, Barry Porter, Danny Hughes, Dynamic reconfiguration in sensor middleware. In *Proceedings of the international workshop on Middleware for sensor networks* (MidSens '06). ACM, New York, NY, USA, 1-6.

[9] Danny Hughes, Klaas Thoelen, Wouter Horr, Nelson Matthys, Javier Del Cid, Sam Michiels, Christophe Huygens, Wouter Joosen, LooCI: a loosely-coupled component infrastructure for networked embedded systems. In *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia* (MoMM '09). ACM, New York, NY, USA, 195-203.

[10] Amirhosein Taherkordi, Quan Le-Trung, Romain Rouvoy, Frank Eliassen, WiSeKit: A Distributed Middleware to Support Application-Level Adaptation in Sensor Networks. In *Proceedings of the 9th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems* (DAIS '09), Twittie Senivongse and Rui Oliveira (Eds.). Springer-Verlag, Berlin, Heidelberg, 44-58.

[11] Luca Mottola, Gian Pietro Picco, and Adil Amjad Sheikh. 2008. FiGaRo: fine-grained software reconfiguration for wireless sensor networks. In *Proceedings of the 5th European conference on Wireless sensor networks* (EWSN'08), Roberto Verdone (Ed.). Springer-Verlag, Berlin, Heidelberg, 286-304.

[12] Barry Porter, 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). ACM, New York, NY, USA, 7-12.

[13] Jagun Kwon, Stephen Hailes, MIREA: Component-based middleware for reconfigurable, embedded control applications, In *Proceedings of the IEEE International Symposium on Intelligent Control* (ISIC,2010),

[14] Manik Gupta, Lamling Venus Shum, Eliane Bodanese, Stephen Hailes, Design and evaluation of an adaptive sampling strategy for a wireless air pollution sensor network, In *Proceedings of the IEEE 36th Conference on Local Computer Networks*, (LCN, 2011).

[15] Adam Dunkels, Bjorn Gronvall, Thiemo Voigt, Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks* (LCN '04). IEEE Computer Society, Washington, DC, USA, 455-462.

[16] Adam Dunkels, Niclas Finne, Joakim Eriksson, and Thiemo Voigt. 2006. Run-time dynamic linking for reprogramming wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems* (SenSys '06). ACM, New York, NY, USA, 15-28.

 ISBN: 978-1-61208-207-3