ROLE OF MIDDLEWARE TECHNOLOGIES FOR DIFFERENT APPLICATIONS: A STUDY

[1]Velmurugan, [2]Naol Bakala, [3]Lami Garoma
[1]Professor, [2]Head of the Department, [3]Lecturer
[123]Computer Science,
[123]Ambo University, Ambo, Ethiopia

Abstract: In distributed systems, software architectures support development motivated on modular functional building blocks (components), their interconnections (configurations), and their interactions (connectors). Since architecture-level components often contain compound functionality, it is realistic to expect that their interactions will be complex as well. Middleware technologies such as CORBA, COM, and RMI provide a set of predefined services for enabling component composition and interaction. However, the potential role of such services in the implementations of software architectures is not well understood. In practice, middleware can resolve various types of component heterogeneity — across platform and language boundaries, for instance — but also can induce unwanted architectural constraints on application development. This paper reviews the application areas and methods through which middleware components communicate through architecture-level software connectors.

Index Terms - ***Internet of Things, middleware, semantic model, context-awareness, ubiquitous computing***.

I. Introduction

Middleware is a software layer exist in on top of the operating system that links different software components or applications. It provides interoperability and other services like the distribution of functionality, scalability, load balancing and fault tolerance [1]. The functionalities of middleware tumble within three general categories which are application-specific, information-exchange, management and support. The application-specific middleware delivers services for different ranges of applications such as distributed-database services, distributed transaction processing, and specialized services for mobile computing and multimedia, while the information-exchange middleware deals mainly with information management. The management and support middleware is used to communicate with servers, manage security, handle failures, and monitor performance [2]. Middleware can be classified according to the way each middleware either assists an application or helps with the integration of multiple applications or application components. Each middleware has different communication protocols and modes of operation. It includes different types such as Procedure Oriented Middleware, Object Oriented Middleware, Message Oriented Middleware, Component Based or Reflective Middleware and Agents [3]. Moreover, middleware provides security mechanisms for many applications. Security became an important issue because most transactions and operations occur online and need data transmission. The applications and data involved need to be protected from malicious and unintentional attacks as well as from any possible risks of exposure. Well defined access polices, encryption mechanisms and authentication models can help in providing security. Pervasive computing refers to the ubiquitous presence of computing in both mobile and embedded environments, with the ability to access and update information anywhere, anyplace and anytime [4].

Middleware is used to support integration and interoperability among ubiquitous computing elements, and recently it has also become important in providing security for ubiquitous computing. This paper explores different security middleware projects and approaches devised to provide different security measures for ubiquitous applications and pervasive environments.

II. BACKGROUND AND RELATED WORKS

Since middleware grips all the communications between a client and the target application, it can also mask all the fundamental complexities including the security aspects. Various kinds of middlewares based on their supported functionalities like adaptability, context awareness and application domains like Wireless Sensor Network (WSN), Radio Frequency
Identification (RFID) are studied. The surveys performed in [8] and [9] have studied the middleware based on context-awareness feature. The survey in [8] is based on the architectural aspects and provides taxonomy of the features of a generic context-aware middleware. Survey reported in [9] evaluates several context-aware architectures based on some relevant criteria from ubiquitous or pervasive computing perspective.

A survey of middleware paradigms for mobile computing [7] presented several research projects that have been initiated to address the dynamic aspects of mobile distributed systems. The authors argue that traditional middleware approaches such as CORBA and Java RMI have worked properly for a long time in dealing with heterogeneity and interoperability but they fail to provide the appropriate support for modern mobile applications. One of the several mentioned new technologies is the Event-Based Paradigm. This paradigm supports the development of large scale distributed systems by facilitating a highly decoupled approach and many-to-many interaction style between client and server. Another research that has been conducted and took a similar approach surveys middleware for mobile ad hoc networks [10]. Here the authors present middleware approaches and solutions for mobile ad hoc networks (MANETs) by making a comparison between their features, and ends the paper with a list of required functionalities for MANET middleware, and the ones that haven't been addressed efficiently in the middleware approaches that were surveyed. One of these middleware approaches is Selma, a mobile agent based middleware that provides functionalities such as positioning, neighborhood discovery and wireless communication.

III. CHARACTERISTICS OF SECURITY MIDDLEWARE

To clearly understand the unique requirements of security middleware for ubiquitous computing, it is important to first highlight the main differentiating characteristics of ubiquitous computing and pervasive environments. It is important to support several needs such as mobility, heterogeneity, seamless interfaces, efficient operations, anywhere/anytime access to information and applications, and secure access and utilization. The user of such applications should not be bothered by how services are designed and where they reside. The user should be able to access any service anywhere, anytime and at the same time be assured of the security and integrity of the system and data being used. Several operational issues have been addressed and efficiently resolved, yet the security issues remain a big obstacle hindering large-scale acceptance and utilization of ubiquitous applications. The main issues that must be addressed in terms of security are:

1. Authentication mechanisms and credential management
2. Authorization and access control management
3. Shared data security and integrity
4. Secure one-to-one and group communication
5. Heterogeneous security/environment requirements support
6. Secure mobility management
7. Capability to operate in devices with low resources
8. Automatic configuration and management of these facilities.

These issues can be addressed independently with each ubiquitous application being developed. However, this approach creates several problems: (1) Developers will need to be aware of these issues and skilled enough to design and implement suitable solutions; (2) each application might repeat many similar functionalities as other applications, which would be a waste of time and effort; (3) applications will increase in size and functionality and most likely will have a larger footprint which makes them hard to port onto small devices.

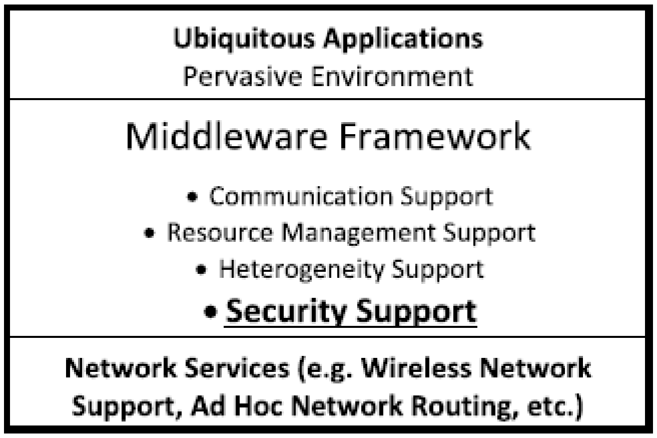

Figure 1 : Middleware support for ubiquitous applications.

As shown in Figure.1, the use of middleware can solve these problems by providing an independent layer to support the security aspects. In this case, the common functions and application independent features can all be part of the middleware. Any application requiring their use can be implemented using the proper API, and would be able to use those functions directly. As a result, such new applications would be more efficient. Additionally, less time and effort will be spent trying to secure each application separately. Several researchers tried to address these issues and provide security middleware to support these functionalities. However, most proposed solutions tackle only a few of these issues at a time.

IV. ROLE OF MIDDLEWARE IN INTERNET OF THINGS

Middleware for IoT is required for various reasons. The summary of reasons is as follows:

1. Difficult to define and enforce a common standard among all the diverse devices belonging to diverse domain in IoT.
2. Middleware acts as a bond joining the heterogeneous components together.
3. Applications of diverse domains demand abstraction /adaptation layer.

4. Middleware provides API (application programming interfacing) for physical layer communications, and required services to the applications, hiding all the details of diversity.

The above stated reasons generate the need for various functional components the IoT middleware must support. The functional components of the middleware as proposed here are portrayed in the current section. The functional component of an IoT-middleware is depicted in Figure. 2.

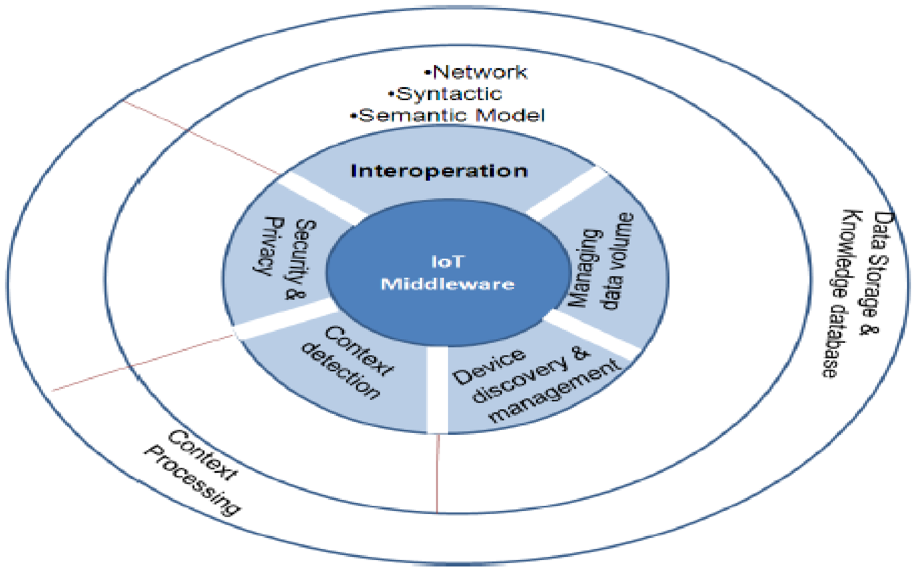

Figure 2: Functional Components of IoT-Middleware

The inner most circle shows the required functional blocks. The second circle encompasses the further division of the functional blocks, and the outermost circle shows the important modules interacting with the various functional components, but not part of the middleware – example context processing, data storage and knowledge database. The functional components are as follows:

• Interoperation
• Context detection
• Device discovery and management
• Security and privacy
• Managing data volume

The system architecture is depicted in Figure.3. It presents a layered view of the IoT middleware architecture.

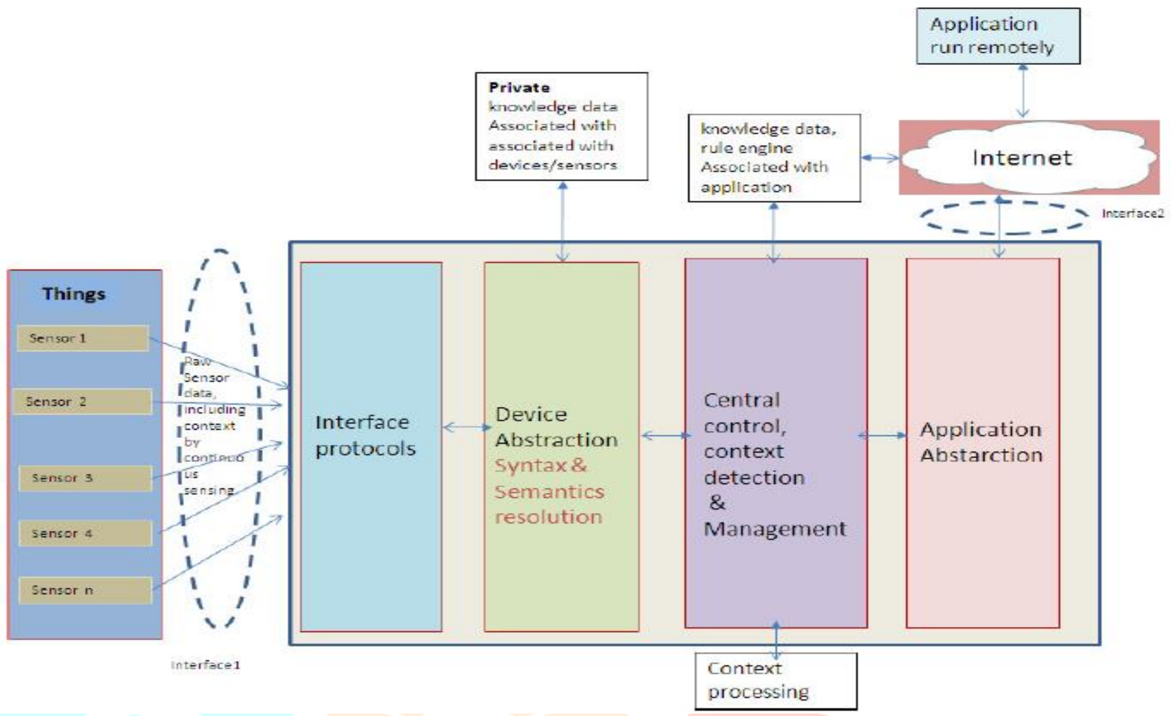
Application run remotely
Private knowledge data Associated with associated with devices/sensors
knowledge data, rule engine Associated with application
Internet
Interface2
Things
Sensor 1
Sensor 2
Sensor 3
Sensor 4
Sensor n
Raw Sensor data, including context by continuous sensing
Interface protocols
Device Abstraction Syntax & Semantics resolution
Central control, context detection & Management
Application Abstarction
Interface1
Context processing

Figure 3: Functional Components of IoT-Middleware

The fundamental layers are interface protocols, device abstraction responsible for providing interoperation, resolving the syntax and semantics associated with devices, central and management module is the core component, which performs the device discovery, management and context detection. Application abstraction module provides the interface with local and remote application. Local applications mainly run as event driven services. The other components like context analysis, knowledge database may reside in remote system essentially generating a need of distributed architecture.

V. MIDDLEWARE-BASED DISTRIBUTED SYSTEMS SOFTWARE PROCESS

Non-functional requirements address important issues of quality and restrictions for software systems, although some of their particular characteristics make their specification and analysis difficult:

(1)Non-functional requirements can be subjective, since they can be interpreted and evaluated differently by different people;
(2)Non-functional requirements can be relative, since their importance and description may vary depending on the particular domain being considered;
(3)Non-functional requirements can be interacting, since the satisfaction of a particular non-functional requirement can hurt or help the achievement of other non-functional requirement.
A set of ISO/IEC standards are related to software quality, being standards number 9126, 14598-1 and 14598-4 the more relevant ones. The main idea behind these standards is the definition of a quality model and its use as a framework for software evaluation. A quality model is defined by means of general characteristics of software, which are further defined into sub-characteristics in a multilevel hierarchy; at the bottom of the hierarchy appear measurable software attributes. Quality requirements may be defined as restrictions over the quality model. The ISO/IEC 9126 standard fixes 6 top level characteristics: functionality, reliability, usability, efficiency, maintainability and portability. Furthermore, an informative annex of this standard provides an illustrative quality model that refines the characteristics as shown in Figure. 4.

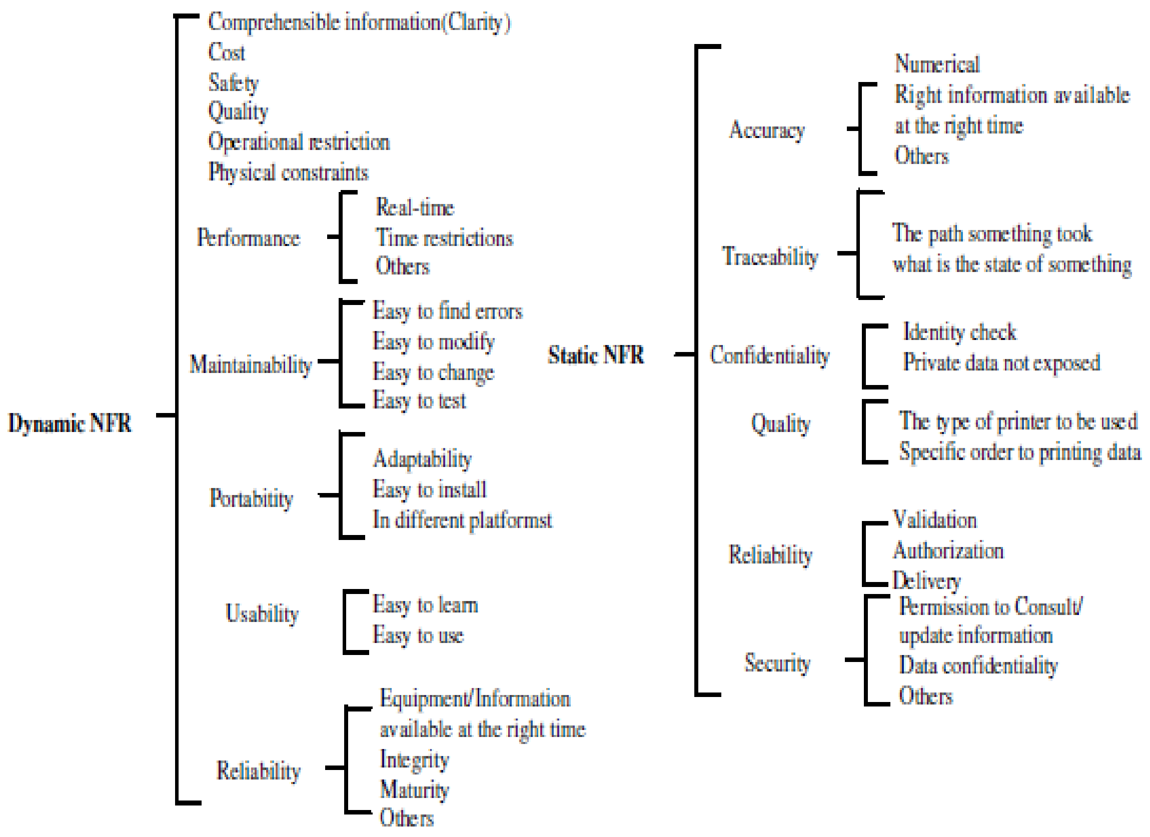

Figure 4: ISO 9126 standard

VI. CHARIOT: CONTEXT-DRIVEN ORCHESTRATION MIDDLEWARE

CHARIOT [14] is context-driven orchestration middleware that supports (1) model-based definition of goals that specify applications required by an IoT system, (2) the composition of—and the requirements imposed by—the applications, and (3) the constraints governing the deployment and (re)configuration of applications. CHARIOT uses these services to support initial application deployment, failure avoidance, fault management, and operations management of distributed IoT systems. Goal-based system description model. CHARIOT use the concept of goal-based system description to describe the IoT system being managed. IoT system goals are de- fined as high-level capabilities that can be decomposed into smaller sub-capabilities using the concept of capability decomposition, as shown in Figure 3. Next, these capabilities are mapped onto logical components that provide the capabilities. These components can have inter-dependencies that are also captured with CHARIOT's goal-based model.

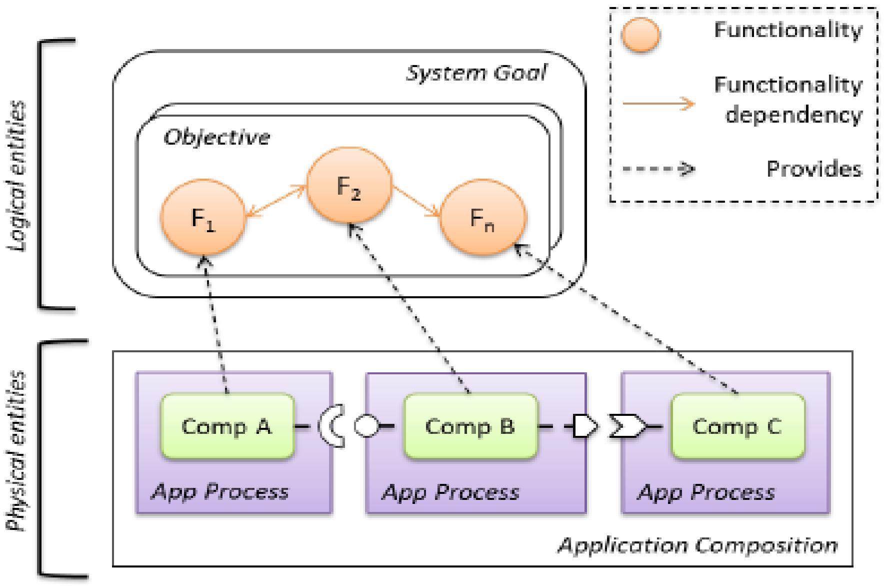

Figure 5: Example of a Goal-based Description of an Application Runtime architecture

CHARIOT provides a monitoring and deployment infrastructure, as well as a novel management engine that uses IoT system information stored persistently to formulate Satisfiability Modulo Theories (SMT) constraints that encode system properties and application requirements. CHARIOT can therefore use SMT solvers to dynamically compute optimal system (re)configuration at run-time. CHARIOT's distributed architecture allows any node in the system to compute the required solution and then automate the resulting deployment to available system resources. Failure detection, group management, and leader election in CHARIOT is provided by Zookeeper [15].
CHARIOT's approach to IoT system management is based on the concepts of configuration space and configuration points. If a system's state is represented by a configuration point in a configuration space, then any change that invalidates a configuration point requires moving the IoT system from one configuration point to another in the same configuration space. As such, a configuration space includes (1) a goal-based description of di_erent IoT systems, (2) replication constraints corresponding to redundancy patterns associated with different systems, (3) constraints on valid component-to-node deployment mappings, and (4) available resources, including different nodes and their corresponding resources, such as memory, storage, and computing elements.
Using Context and Goal Information for Runtime Management and Reconfiguration. As shown in Figure 6, CHARIOT's Monitoring Infrastructure is responsible for detecting changes in the sensing phase.

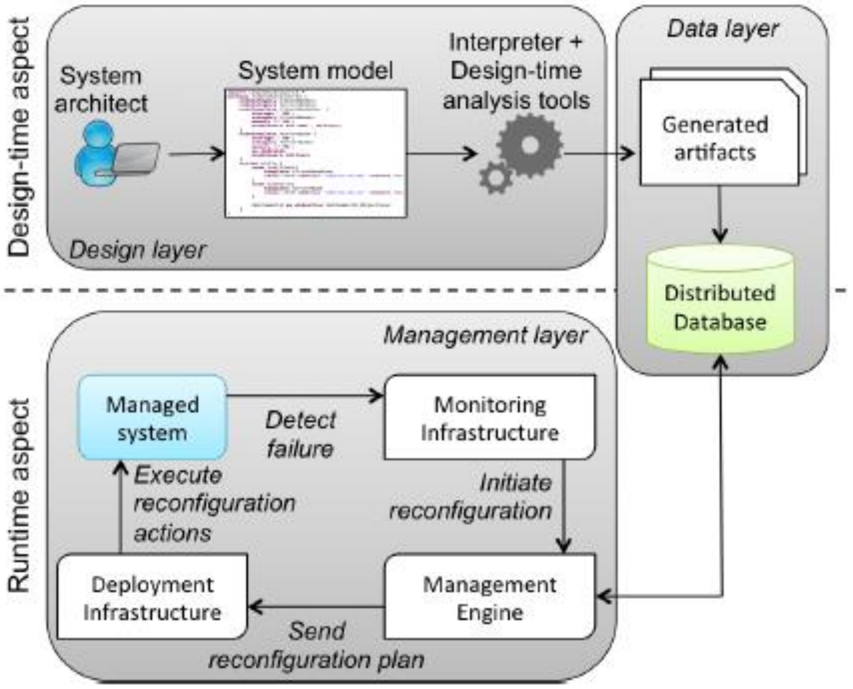

Figure 6: CHARIOT's Self-reconfiguration Mechanisms

After detection and diagnosis, its Management Engine determines the goals that are required based on the current context of an IoT system. This context is derived from known global system states. Thereafter, actions needed to reconfigure the system such that changes are handled are calculated in the planning phase. After reconfiguration actions are computed, CHARIOT's Deployment Infrastructure is responsible for taking those actions to reconfigure the system in the acting phase.

VII. CONCLUDING REMARKS

Modeling and managing IoT systems with dynamic orchestration middleware like CHARIOT helps digitally-enhance smart grid operations to more effectively manage risks arising from volatility in the power system domain by flexibly determining where to deploy functionality throughout a grid hierarchy. Context aware functionality is supported by HYDRA, UBIWARE, UBIROAD and SMEPP. On the other hand, SOCRADES, SMEPP, GSN, UBIROAD and HYDRA are some examples of middleware implementing security and user privacy in their architecture. Based on platform portability, syntactic resolution, HYDRA, SMEPP and ASPIRE are OSGi compliant, UBIROAD uses JAVA and XML, UBISOAP uses J2SE and J2ME, GSN uses XML and SQL, SIRENA and SOCRADES use DPWS while SOCRADES also uses SAP NetWeaver platform and ISMB uses any JAVA compliant platform. WhereX is developed using J2EE architecture and is integrated with Oracle Application Server 10g. It also uses Rhino rule engine which is implementation of Java Script.

REFERENCES

[1] Middleware, Wikipedia, retrieved March 1, 2009, from: http://en.wikipedia.org/wiki/Middleware.
[2] Q. Chen, J. Yao, R. Xing, Middleware components for E-commerce infrastructure: An analytical review, Journal of Issues in Informing Science and Information Technology 3 (2006) 137_146.
[3] T.A. Bishop, R.K. Karne, A survey of middleware, in: Proc. of Computers and their Applications, 2003, pp. 254_258.
[4] Pervasive Computing, the Parliamentary Office of Science and Technology, retrieved March 1, 2009, from: http://www.parliament.uk/documents/ upload/postpn263.pdf.
[5] R. Jason, Weiss, J. Philip Craiger, Ubiquitous Computing, retrieved April 27, 2009, from: http://www.siop.org/tip/backissues/TIPApr02/pdf/394_044to052.pdf.
[6] J. Mencl, Benefits and Drawbacks of Ubiquitous Computing, retrieved April 27, 2009, from: http://www.mencl.us/pdfs/ubicomp.pdf.
[7] A. Gaddah, T. Kunz, A survey of middleware paradigms for mobile computing, Technical Report SCE-03-16, Carleton Univ. Systems and Computing Engineering, 2003.
[8] Kjær, K., E.: A Survey of Context-Aware Middleware. In: 25th conference on IASTED International Multi-Conference: Software Engineering, pp. 148--155. ACTA Press (2007)

[9] Miraoui, M., Tadj C., Amar, C. B.: Architectural Survey of Context-Aware Systems in Pervasive Computing Environment. In: Ubiquitous Computing and Communication Journal, vol. 3, no. 3 (2008)

[10] G. Paroux, I. Demeure, D. Baruch, A survey of middleware for mobile ad hoc networks, Technical Report 2007D004, Ecole Nationale Supérieure des Télécommunications, France, January 2007.

[11] Jameela Al-Jaroodi _, Imad Jawhar, Alyaziyah Al-Dhaheri, Fatmah Al-Abdouli, Nader Mohamed : Security middleware approaches and issues for ubiquitous applications. Computers and Mathematics with Applications 60 (2010) 187_197, 0898-1221/$ _ see front matter ' 2010 Elsevier Ltd.

[12] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti and Subhajit Dutta. : Role of Middleware for Internet of Things: A Study. International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.3, August 2011.

[13] Liu JingYong, Zhang LiChen, Zhong Yong and Chen Yong. : Middleware-based Distributed Systems Software Process. International Journal of Advanced Science and Technology Volume 13, December, 2009.

[14] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference, volume 8, pages 11{11, 2010.

[15]. Abhishek Dubey, Subhav Pradhan, and Douglas C. Schmidt. : The Role of Context and Resilient Middleware in Next Generation Smart Grids. M4IoT 2016, December 12-16, 2016, Trento, Italy @ 2016 ACM. ISBN 978-1-4503-4663-4/16/12. DOI: http://dx.doi.org/10.1145/3008631.3008632.