Hindawi Publishing Corporation
Journal of Computational Engineering
Volume 2014, Article ID 725960, 7 pages
http://dx.doi.org/10.1155/2014/725960

Review Article

QoS-Aware Middleware for Ubiquitous Environment: A Review and Proposed Solution

Anas Al-Roubaiey[1] and M. AL-Rhman Alkhiaty[2]

[1] *Computer Engineering Department, King Fahd University of Petroleum and Minerals, P.O. Box 1498, Dhahran 31261, Saudi Arabia*
[2] *Computer Science Department, King Fahd University of Petroleum and Minerals, P.O. Box 1498, Dhahran 31261, Saudi Arabia*

Correspondence should be addressed to Anas Al-Roubaiey; roubaiey@kfupm.edu.sa

Received 4 November 2013; Accepted 7 February 2014; Published 19 March 2014

Academic Editor: Hongli Dong

Copyright © 2014 A. Al-Roubaiey and M. Alkhiaty. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Ubiquitous computing has introduced a new era of computing. Compared to traditional distributed systems, ubiquitous computing systems feature increased dynamism and heterogeneity. In traditional computing environments (mainframe and PC), users actively choose to interact with computers. Ubiquitous computing applications are likely to be different, where computing systems are available anywhere but not visible. The underlying ubiquitous computing infrastructures are more complex and bring up many issues. In this work we survey the literature to demonstrate, in detail, the characteristics and the challenges of the ubiquitous computing as well as the requirements for building ubiquitous software that brings these characteristics into reality. Furthermore we present some existing middleware solutions for ubiquitous environments, and propose our middleware-based architecture to facilitate the user interaction in such environment. To the best of our knowledge this is the first work proposing DDS-based solution for ubiquitous computing as a unified middleware.

1. Introduction

> *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." Mark Weiser, 1991.*

Mark Weiser, by stating this, was dreaming of a world that would be flooded with embedded devices, note pads, and electronic dust. The remarkable advances that have been achieved over the past ten years in the manufacturing of semiconductor devices and microsystems have shown that there can be no doubt about the question whether or not Mark's dream will come true [1, 2]. At that time, Weiser [3] was searching for a term to describe his vision of an embedded computing world within which computers would be an invisible part of our everyday lives. Ahead of their time, Weiser and colleagues used the terms ubiquitous and pervasive computing interchangeably to describe how computing was going to change from desktop, personal computing to a more distributed, mobile, and embedded form. But what does the term ubiquitous actually mean?

Despite being used interchangeably with the term "pervasive," they do refer to different forms of computing. Ubiquitous computing refers to building a global computing environment where seamless and invisible access to computing resources is provided to the user. It deals with giving users the ability to access information services, applications, and resources globally available through any device over different kinds of networks all the time and irrespective to their location. Pervasive computing is concerned with mobile data access, smart spaces, and context awareness as well as dynamic and proactive services. It focuses on nomadic users and the way they interact with the environment [2]. A ubiquitous computing environment is created by sharing knowledge and information between pervasive computing environments. Collaboration of the pervasive environments can create an effective ubiquitous computing environment [4].

Building software for such environment is very difficult due to its heterogeneity and dynamic and unpredictable behaviour. Middleware technology has emerged to facilitate the development of applications on heterogeneous environments; many middleware-based solutions have been proposed to tackle the challenging issues of ubiquitous computing [5–8]; in this study we investigate the challenging issues and the requirements of building ubiquitous software. Moreover, we propose a middleware solution for building ubiquitous software.

Our proposed middleware is based on Data Distribution Service (DDS) middleware. DDS has been standardized by Object Management Group (OMG) in 2004 [9]. DDS is a middleware based on publish-subscribe communication model which has distinguished characteristics that make it the best solution for building ubiquitous software. In contrast to tightly coupled solutions, DDS is more flexible and all its components are decoupled in time and space, which makes it more flexible and scalable. It supports dynamic environment due to its discovery protocol that can automatically discover any change in the network components. Additionally, mainly it is standardized for real time distributed applications; it supports many QoS policies that can be utilized to significantly improve the performance of ubiquitous computing. In this paper we discuss the most important QoS policies that can improve the performance of ubiquitous computing and we give some examples to show how it can be used to manage and improve the application behaviour to get the optimum performance.

This paper is organized as follows. Section 2 presents the research goals and motivations. Section 3 characterizes the ubiquitous applications. In Section 4 we demonstrate the ubiquitous software requirements. Section 5 we present a literature review for the middleware-based architectures for ubiquitous computing. Then our proposed architecture is described in Section 6 and the conclusions and future work are discussed in Section 7.

2. Goals and Motivations

Ubiquitous computing has introduced a new era of computing. Compared to traditional distributed systems, ubiquitous computing systems feature increased dynamism and heterogeneity. In traditional computing environments (mainframe and PC), users actively choose to interact with computers. Ubiquitous computing applications are likely to be different, where computing systems are available anywhere but not visible. Figure 1 shows the emergence of three different computing eras. And Figure 2 shows the need for small, embedded, and wearable computers and inspires us to recognize the environments that ubiquitous computing concept will transfer us to.

The underlying ubiquitous computing infrastructures are more complex and bring up many issues such as user mobility, disconnection, dynamic introduction and removal of devices, and heterogeneous network connections, as well as the need to integrate the physical environment with the computing infrastructure.

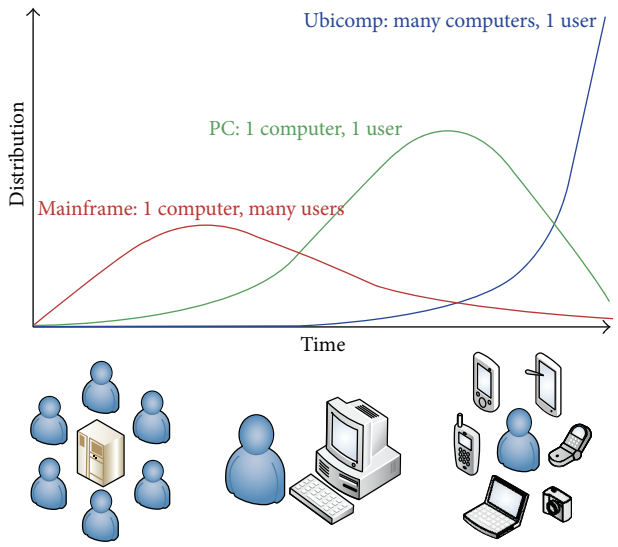


Figure 1: Computing evolution.

Figure 2: Traditional computing: large, stationary, desktop.

The characteristics (Section 3) along with the complexity and the requirements of ubiquitous computing environments (Section 4), as a new paradigm of computing, introduced a compelling demand for new platforms and software engineering technologies; new requirements engineering; new software architectures; and new software design and developments models. The main challenges for building ubiquitous middleware software are the support to the user mobility, the environment dynamicity, the underline heterogeneity, the context awareness and adaptiveness, services and network interoperability and more importantly, providing the right service/data to the right user in the right time within certain context.

Motivated by the above issues, our goal in this work is to survey the literature to demonstrate, in detail, the characteristics and the challenges of the ubiquitous computing as well as the requirements for building a ubiquitous software that make these characteristics a reality. Our ultimate goal, after presenting some existing middleware architecture for the ubiquitous computing, is to come up with middleware architecture to facilitate the user interaction in such environment.

Ubiquitous middleware software is a platform, which abstracts and facilitates the complexity, dynamicity, mobility, and heterogeneity of the underlying distributed and context aware environment with its depth of network technologies, hardware, operating systems, and programming languages. Ubiquitous computing is a challenge for the design of middleware. The reasons are the set of characteristics and the requirements for such environment. The middleware for such environment has to be tailored to the application scenario as well as to the target platform. Middleware plays a key role to make the ubiquitous computing vision becomes true. It supports the application programmer, who builds distributed applications and services.

Such middleware must deal with the various characteristics of ubiquitous computing scenarios and handle its requirements that will be listed soon. Conventional middleware is not sufficient for such era of computing. The ubiquitous middleware has to be built from minimal fine-grained components, and the system structure must be highly configurable to support and facilitate the computing in such smart and context aware environment [10].

3. Ubiquitous Computing Characteristics

In order to identify the requirements of the ubiquitous middleware software, first summarize the main characteristics (with example) of the ubiquitous environment that differentiate it from the traditional computing paradigm (i.e., mainframe and PC). These characteristics make the conventional human-computer interaction knowledge that we have built up overtime inadequate [11]. Some of these characteristics are physical integration; spontaneous interoperation; universal accessibility of devices to information services; effective collaboration among the parties accessing the information services, proactively; computing in a social environment; and invisibility.

3.1. Physical Integration. UC involves some integration between computing nodes and physical world (see Figure 3).

Referring to the scenario in [12], the limousine serves as a car in the usual way but also it uses sensors to detect who the passenger is to adjust (seat, screen, keyboard, and voice recognition software).

3.2. Spontaneous Interoperation. In UC, components must spontaneously interoperate in changing environments. A spontaneously interacting component changes partners during its normal operation, without needing new software or parameters. Again referring to the scenario in [12], when he just boards the plane, his PDA again scans for possible services in his environment and detects that it can connect to the screen and keyboard built into the plane's seat using an airplane wireless network.

3.3. Context Awareness. Context is used to characterize the situation of an entity (user, place, and physical or computational object). Context aware software is able to assign meaning to the events in the outside world and use that information effectively. UC vision is to adapt behaviour based on the user's profiles and preferences. Again, referring to the scenario, a simple example is a message system may choose to deliver a message by email or by voice (telephone) depending on the current state of the user (is the user in a meeting? handling a crisis? etc.).

Figure 3: Ubiquitous computing: small, mobile, palmtop.

3.4. Proactive Service. It is the ability of the system to serve the user without his request. It is based on the intelligent environment's capability of context awareness. Computer should remember the past, recognize the present, and predicate the future. In our running scenario, sequence of future operations were done after changing flight reservations; the agent rebooked the flights; it cancelled the hotel room reservation; it cancelled the meetings with the supplier; it sent emails.

4. Ubiquitous Software Requirements

After discussing the characteristics of the ubiquitous computing, this section will point out set of ubiquitous computing requirements to achieve this computing paradigm and make it a reality. According to [2], the requirements of ubiquitous computing systems are classified into three categories: system (platform), software, and business and organization. The term system refers to computing platforms. Ubiquitous software refers to services and components of a ubiquitous system realized by software technologies. Business refers to the network of actors who provide software components and services for the system development and deployment.

Based on the aforementioned characteristics of the ubiquitous computing, the requirements of ubiquitous computing environment and thus the ubiquitous software are interoperability, heterogeneity, mobility, adaptability, security and privacy, self-organization, and augmented reality and content scalability. Most of these requirements have to be coupled with the system, software, and business and organization levels.

4.1. Interoperability. Interoperability is the capability of the software to communicate, execute programs, or transfer data among various functional units, regardless of

the heterogeneity of the underlying technology. So, when information is exchanged among distributed software platforms, will software be able to understand this information and able to react with this information.

4.2. Heterogeneity. Heterogeneity is defined as the ability to handle different programming languages, operating systems, hardware, or communication protocols. It also outlines the diversity between entity's capability and possibilities offered by various unknown functionalities of new smart objects [13].

In the ubiquitous environment, the networks is heterogeneous connecting devices with different screen resolutions, user interaction methods, radio capabilities, memory, power, and processing capabilities, as well as mobility. Services can be accessed with widely varying transport capability, quality and usage cost, and they may have different requirements for bandwidth, real-time capabilities, and input output methods. User interaction with computers is performed by various mechanisms that enable the users to interact with devices using multimodality such as speech, hand movement, screens, and buttons. This kind of interaction requires novel solutions in the form of embedded sensors.

4.3. Scalability and Resource Discovery. Scalability refers to the ability of a system to grow in the future and to extend to higher-load applications or to a wider network. The mobility and availability of the infinite number of heterogeneous resources at the same time entail requirements such as scalability and resource discovery.

As the number of services and devices grows up, identifying required services, verifying service compositions, and mapping services to devices become correspondingly more complex for the system developer. The three fundamental scalability challenges, which can occur independently of any application, are service identification, verification of service compositions, and intelligent service-to-device mapping [1].

4.4. Mobility. Mobility means that the user can freely get a same personal service or interact with system while moving among places. There are three kinds of mobility, from the point of view of software:

(i) actual mobility: this concerns with the idea that the software can automatically transfer its execution (code, data, and status of execution) to the nodes where the resources—it needs to access—are located,

(ii) wirtual mobility: the ability to understand the variety of networked execution environments,

(iii) physical mobility: the use of wireless computing devices and mobile that are connecting using dynamically changing access points.

4.5. Adaptability. The environment where the user lives is usually surrounded by different kinds of devices, network capabilities, and dynamically evolving context. Those devices need to be aware of changes in their environment.

Software services must adapt themselves to the environmental changes by generating actions that lead to ensuring the continued satisfaction of user needs. This adaptation should support the user with the preferred services at a certain time or situation. Because of mobility in ubiquitous environment, dynamically changing environment makes adaptability a big challenge.

4.6. Survivability and Security. In ubiquitous environment, the behavior of components may be unpredictable because of changes in the environment (e.g., network connection failure, services failure to provide the expected functional and/or nonfunctional qualities, etc.). Survivability is the ability of the system to achieve its mission within time and deliver its essential services even in the presence of attack or failure [2]. So, such systems need an infrastructure that can tackle itself. Survivable system require self-healing infrastructure with improved qualities, such as security, performance, reliability, availability, and robustness. Security can be an important concern in some applications since ubiquitous computing may use private data from the user.

4.7. Self-Organization. Self-organization is the ability of the system to adapt and organize its behavior without the need for external environment control. This also includes the ability of the system to create its own organization, that is, the ability to dynamically reorganize the structure of the software (having dynamic software architectures.)

5. Middleware Architecture in Literature

In this section we first list and summarize some of the ubiquitous middleware models introduced in literature WComp [13], Gaia [5], EXORB [14], Oxygen [15], Aura [16], and so forth.

Odyssey [17] is a lightweight middleware, which monitors resources such as bandwidth, CPU cycles, and battery power and interacts with each application to best exploit them. The focus of Odyssey is the resource adaptation. For example, when connectivity is lost due to a radio, Odyssey detects the change and notifies the interested applications. Reaction to the notification depends on the application.

Gaia [5] is a metaprogramming environment that exports a service to query, uses existing resources and context, and provides a framework for developing user-centric, resource-aware, multidevice, context-sensitive mobile applications. Gaia's main characteristic is the functionality it provides for the interaction of individual services. This interaction provides users and developers with an abstract ubiquitous computing environment as a single reactive and programmable entity, instead of a collection of heterogeneous individual devices. Gaia has been applied in a ubiquitous computing environment similar to digitally augmented meeting rooms. It provides five basic services based on top of CORBA middleware. These services are event manager, context service, presence service, space repository, and context file system. The main focus of Giga is to handle the heterogeneity involved in such environment.

Aura [16] is a context aware middleware that can be used to create mobile applications. The main goal of the Aura

architecture is to maximize the use of the available resources and to minimize user distraction. It represents the user by its aura, like a personal area network, and brings the appropriate resources from the services of the environment to support the user's task. Aura migrates tasks depending on context changes, which are notified by events. It is also interesting.

EXORB [14] is a project aiming at construction of configurable, updatable, and upgradable middleware services. EXORB uses IIOP and XML-RPC, enabling heterogeneity. Its software configuration can change at runtime, implying an adaptability potential.

Oxygen [15] addresses human needs using speech and vision technologies that enable the user to communicate with it as if the user was interacting with a person. It enables pervasive human-centered computing. It defines intelligent networks with dynamic topologies according to devices locations, fixed and mobile devices with embedded software. Code can be automatically updated thanks to that. Network rules can be specified to allow sets of users to use particular resources. However, it uses objects, communicating with method invocation, and does not handle reactivity or heterogeneity.

WComp [13] is a platform that integrates intelligent devices that may appear and disappear dynamically. It joins three main paradigms: an event-based Web services approach, a lightweight component-based approach to design composite Web services, and an adaptation approach. It is based on UPnP discovery protocol.

6. Proposed Middleware Architecture

After pointing out the important characteristics and requirements of the ubiquitous world and some existing architectures in literature, we are about to present our proposed architecture, which is built as a response to basic and the fundamental requirements of the ubiquitous world. These fundamental requirements can be simply stated as follows. A nomadic user (with his wearable or portable devices) wants a specific service within certain context (where heterogeneity and dynamism are highly presented) on the right time. Thus, our proposed architecture will be structured around these requirements, which are the main challenges in such environment, taking into account set of constraints, like time constraints, resource limits, and security.

The proposed architecture is shown in Figure 4. As it is clear in this figure, we have a user (normally nomadic user) carrying devices (represented as an abstract entity). These devices can be portable or wearable as shown in the figure. When the user joins the context, his devices contact the context manager using the DDS middleware to announce his presence and the discovery service in DDS will dynamically register him as a new participant and create a user profile with his associated QoS requirements. Also it will match his requests with the existing published topics; if it will match and no incompatible QoS parameters are found then the connection will be established between the publisher and subscriber; since DDS supports the communication in a publisher subscriber fashion. The user acts as a publisher and subscriber to the context. As a publisher he provides the context manager with his information; he may also configure or update his profile and publish it to the context manager. As a subscriber he gets the history from the context about previous interaction within that context and can get the previously published data based on the durability QoS policy.

This is clearly depicted in the upper side of Figure 4. When the context recognizes the user and becomes aware of his existence and preferences (may be based on previous interaction and preferences or based on currently updated and configured information), it may act proactively and provide the user with his services; the user may or may not require confirming the service. According to [18] there are three ways to achieve personalization:

(i) location based: meaning that a user's location is taken into consideration for service provision,

(ii) context aware: meaning that beyond location information, service provision takes into account user's environment context information. Obviously this helps delivering the right service at the right time,

(iii) situation aware: meaning that an abstraction of context information could be done by translating this information into logical situation.

The user can ask for some services within that context. In this case, the user acts as a subscriber to the service provider who is the service publisher.

Time management module uses several DDS QoS policies like Deadline, Time-based filter, and Liveliness. This quality-of-services can be used to control the timing conditions to meet the environment requirements such as sending rate to high and low rate capabilities devices; Time-based filter is used for that purpose. Also for limited resources devices the resource management module use the resource-limits QoS policies to specify the required resources such as memory space. Content-based filter QoS used to specify the desired data in each participant, such that they will not be overwhelmed by unwanted data.

For different communication types and data traffic, DDS has been examined in both academic researches and commercial products. For example, for video traffic DDS has been approved that it is suitable standard to be used in video streaming [19–21]. And also for limited resources networks it has been enhanced to meet the limited-band width networks by RTI Company [22]. For sensor networks several optimisations have been proposed to adapt DDS to work with tiny devices, for example, TinyDDS [23], μDDS [24], and DDS micro [22].

The proposed architecture can be instantiated and realized as publish-subscribe programming model for distributed systems. DDS is a well known specification for the real-time publish subscribe middleware. It is supported by a set of quality of services that make it a practical real time publish subscribe application. As it is clear from the characteristics and the requirements of ubiquitous computing, DDS seems as a promising solution to be adapted to such environment.

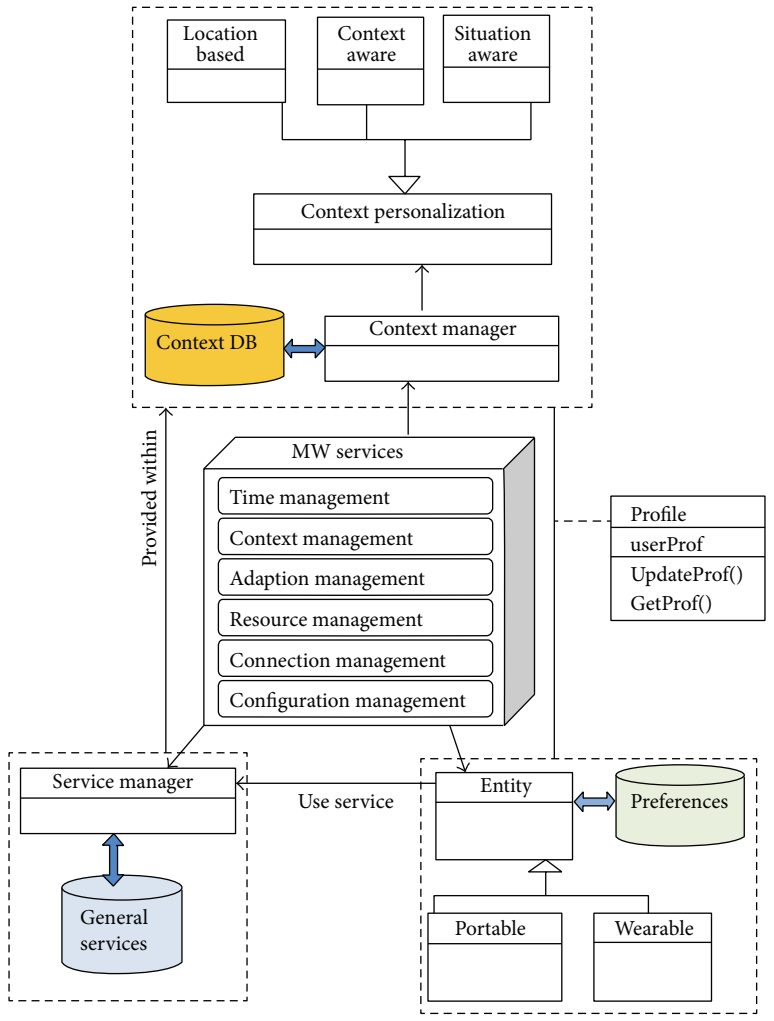


Figure 4: Proposed MW architecture for UC environment.

7. Conclusion and Future Work

In this work we first studied characteristics and requirements of ubiquitous computing middleware and paradigms. Then we summarize some existing middleware architecture presented in literature. Doing so gave us ideas of what was mandatory and lacking in ubiquitous computing. As a result, we introduced our proposed middleware architecture as a response to the characteristics and requirements in such environment. The proposed architecture can be instantiated and realized as publish-subscribe programming model for distributed systems, for which DDS is a well known specification. As a future work our proposed architecture needs to be extended to handle the security and trust policies and mechanisms in such open environment.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] M. Muhlhauser and I. Gurevych, Eds., *Handbook of Research on Ubiquitous Computing Technology for Real Time Enterprises*, IGI Publishing, Hershey Pa, USA, 2007.

[2] E. Niemelä and J. Latvakoski, "Survey of requirements and solutions for ubiquitous software," in *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia*, October 2004.

[3] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 3, pp. 94–104, 1991.

[4] S. Singh, S. Puradkar, and Y. Lee, "Ubiquitous computing: connecting pervasive computing through Semantic Web," *Information Systems and E-Business Management*, vol. 4, no. 4, pp. 421–439, 2006.

[5] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A middleware infrastructure for active spaces," *IEEE Pervasive Computing*, vol. 1, no. 4, pp. 74–83, 2002.

[6] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, "A survey of context data distribution for mobile ubiquitous systems," *ACM Computing Surveys*, vol. 44, no. 4, article 24, 2012.

[7] G. Chen, M. Li, and D. Kotz, "Data-centric middleware for context-aware pervasive computing," *Pervasive and Mobile Computing*, vol. 4, no. 2, pp. 216–253, 2008.

[8] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, "QoS-aware middleware for ubiquitous and heterogeneous environments," *IEEE Communications Magazine*, vol. 39, no. 11, pp. 140–148, 2001.

[9] Object Management Group, "Data Distribution Service for Real-time systems," 1.2 formal/07-01-01 edition, January 2007, http://www.omg.org/.

[10] S. Apel and K. Böhm, "Towards the development of ubiquitous middleware product lines," in *Software Engineering and Middleware*, vol. 3437 of *Lecture Notes in Computer Science*, pp. 137–153, 2005.

[11] A. Ndiwalana, C. Chewar, J. Somervell, and D. McCrickard, *Ubiquitous computing: by the people, for the people [M.S. thesis]*, June 2003.

[12] G. Banavar and A. Bernstein, "Challenges in design and software infrastructure for ubiquitous computing applications," *Advances in Computers*, vol. 62, pp. 179–202, 2004.

[13] J.-Y. Tigli, S. Lavirotte, G. Rey et al., "WComp middleware for ubiquitous computing: aspects and composite event-based Web services," *Annals of Telecommunications*, vol. 64, no. 3-4, pp. 197–214, 2009.

[14] M. Roman and N. Islam, Eds., *Dynamically Programmable and Reconfigurable Middleware Services*, vol. 3231 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2004.

[15] Computer Science and Artificial Intelligence Laboratory, Mit oxygen project, 2004, http://oxygen.lcs.mit.edu/.

[16] J. P. Sousa and D. Garlan, "Aura: an architectural framework for user mobility in ubiquitous computing environments," in *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, Montreal, Canada, 2002.

[17] B. D. Noble, D. Narayannan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," in *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Saint Malo, France, 1997.

[18] Z. Jaroucheh, X. Liu, and S. Smith, "A perspective on middleware-oriented context-aware pervasive systems," in *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '09)*, pp. 249–254, July 2009.

[19] R. Nossenson, O. Yudilevich, and O. Markowitz, "Client-server architecture and algorithms for ubiquitous video service," in *Proceedings of the 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA '12)*, pp. 762–769, July 2012.

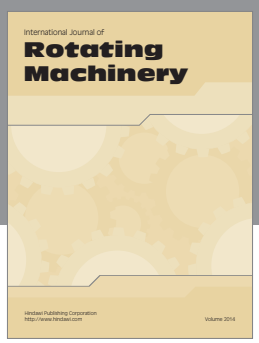
[20] B. Al-Madani, A. Al-Roubaiey, and T. Al-Shehari, "Wireless video streaming over Data Distribution Service middleware," in *Proceedings of the 3rd IEEE International Conference on Software Engineering and Service Science (ICSESS '12)*, pp. 263–266, June 2012.

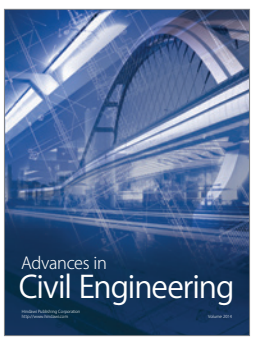
[21] A. Detti, P. Loreti, N. Blefari-Melazzi, and F. Fedi, "Streaming H.264 scalable video over data distribution service in a wireless environment," in *Proceedings of the IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM '10)*, pp. 1–3, June 2010.

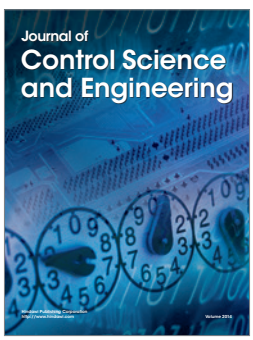
[22] RTI Data Distribution Service, Real-Time Innovations, Inc, 2006, http://www.rti.com/.

[23] P. Boonma and J. Suzuki, "Toward interoperable publish/subscribe communication between wireless sensor networks and access networks," in *Proceedings of the 6th IEEE Consumer Communications and Networking Conference (CCNC '09)*, pp. 1–6, January 2009.

[24] A. González, W. Mata, L. Villaseñor et al., "DDS: a middleware for real-time wireless embedded systems," *Journal of Intelligent & Robotic Systems*, vol. 64, no. 3, pp. 489–503, 2011.







Hindawi
Submit your manuscripts at
http://www.hindawi.com