A Survey of Testing Context-aware Software: Challenges and Resolution

Songhui Yue [1], **Songqing Yue**[2], **and Randy Smith** [1]
[1]Department of Computer Science, University of Alabama, Tuscaloosa, AL, USA
[2]Department of Mathematics and Computer Science, University of Central Missouri, Warrensburg, MO, USA

Abstract
Testing is an essential method to ensure the quality of software. Research of testing context-aware software is gaining in importance with the rapid development of context-aware software and the increasing needs to ensure their quality. Context-aware abilities bring new challenges to testing context-aware software. This paper investigates this from the perspective of four categories of challenges: context data, adequacy criteria, adaptation and testing execution. We also describe approaches current researchers are using to solve these challenges. Our contributions in this paper include the analysis of the relationships between the identified challenges and an ontology diagram that depicts these challenges and relationships, which may benefit the exploration of future research in related areas.

Keywords: Context-aware, Testing, Quality, Challenges, Resolution

1 Introduction

Nowadays, our electronic devices become more powerful in both computing and obtaining information from the environment. Many new devices employ a multi-core processor, and with the technological advances in networked computing environments, new computing paradigms such as cloud computing have been proposed and adopted [8]. Consumers with mobile devices can access data from a "Cloud" at any time in a fast speed wherever network connection is available. Particularly, a modern smart phone can be equipped with as many as fourteen sensors [9], such as proximity sensor, ambient light sensor, accelerometer, magnetometer, and gyroscopic sensor. As a result, a large variety of information could be used as context to enrich the functionality of software applications. The extra abilities of modern devices could be used by applications to process more information for benefits of users, and this advantage makes context-aware become more and more popular in ubiquitous computing area.

A variety of context-aware applications have already been developed, such as location-aware systems, hospital information-aware systems, office-aware applications, and home-aware applications [4][5][6]. These applications are deployed on different platforms, such as mobile applications, web-based applications [10] and embedded applications. Plenty of concepts and components were introduced for facilitating the development of context-aware software, such as context, context-aware middleware, and adaptation rule. They provide software with context-aware abilities and meantime bring new challenges to testing, thus should be considered thoroughly. We will discuss these concepts in detail in section 2.

The following sections are organized as follows: Section 2 introduces some key concepts as the background for understanding our study. Section 3 describes the four categories of challenges we identify from our survey and various approaches to solving them. Section 4 analyzes the relationship between the areas inspired by the challenges and Section 5 serves as the conclusion.

2 Background

This section provides detailed explanation of important concepts that serve as the basis for understanding testing context-aware software.

2.1 Context

The context definitions given by researchers are slightly different from each other because of their different understanding or application of the term. Schilit and Theimer [14] first introduced "context-aware" in their work and defined context as location, identities of nearby people and objects and changes to those objects (1994). Brown [15] defined context as a combination of elements of the user's environment that the computer knows about (1996). Dey et al. [16] defined context as the user information and user's changing location, the changing objects in the environment, and the familiarity with the environment (1998).

Based on all the prior attempts to define context, Dey & Abowd (2000) [17] provided a comprehensive definition of context which is used by most of the current related studies as "*any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects.*"

In a context-aware application, context data can be retrieved with the assistance of hardware or software. For location based context-aware software, context information contains discrete data to mark the locations, which are usually derived from the hardware level [19]. Sensors are widely utilized to capture changing contextual data and then pass them to the software [18]. Context data may also be generated from the software level. For instance, contextual information can be collected from other applications running in the same or related devices [18].

2.2 Context-aware Middleware

Context-aware middleware is widely used for facilitating development and execution of context-aware software [13]. Middleware refers to software systems, which provide an abstraction and mechanisms between network operating system layer and applications layer [20] [21]. Researchers have developed various middleware systems for building and rapidly prototyping context-aware services [22] [23]. As the work in [24] suggests, typical middleware architecture for developing context-aware software contains two key components: context manager and adaptation manager. Context manager captures and manages context from surroundings, and pushes the context changes to adaptation manager. Adaptation manager is responsible for reasoning on the impact of context changes and then choosing proper reactions for applications behaviors.

2.3 Context-aware Adaptation

Context-aware adaptation refers to the ability of computing systems to adapt their behaviors or structures to highly dynamic environments without explicit intervention from users, with the ultimate aim of improving the user experience of these computing systems [35]. Context can be used by software through triggering the context adaptation rules. Adaptation rules, which are usually maintained, evaluated and applied by adaptation manager of a context-aware system, define a significant portion of an application's behavior [13]. We can use an example of a car system to illustrate how an adaptation rule works. Suppose a car installed with an autonomous-driving system (ADS) needs to change lanes. The adaptation rules in ADS need to assure that the car can take this action only if the current context is safe for changing lanes. There should be some additional rules to define what is safe in a real driving environment, which ADS can use to check the safety. If ADS knows the context is safe, it will choose a way to react according to some other rules: changing to left lane or changing to right lane, and in what speed.

2.4 Boundary testing

Boundary testing is an important traditional testing technique, which can also be applied to testing context-aware software. With boundary value testing, test cases are designed to take extremes of input domain. The extremes include values of maximum, minimum, inside/outside boundaries, typical values, error values, and etc. New challenges emerge when boundary testing is used in testing context-aware software, which may require extra attention.

3 Challenges in Testing Context-aware Software

Context-aware capacity imposes many new challenges in developing and testing applications that support context-awareness. After investigating the state of the art in this area, we have identified four main categories of challenges in testing context-aware software: context source, adequacy criteria, adaptation and testing execution. In this section, we provide detailed description for challenges in each category.

3.1 Context

Wang et al. [7] argue that the added capabilities of context-awareness introduce a distinct input space. Since context changes can affect software behavior at any point during the execution, context as testing data should be well studied and selected. However, context data retrieved from sensors usually have such characteristics as being *inaccurate*, *inconsistent*, and *continuous* which may increase the difficulty in selecting testing data. In this subsection, we mainly discuss the features of inaccuracy and inconsistency in context data and briefly introduce how continuous context may affect boundary testing.

3.1.1 Context Inaccuracy

Sensor data can be inaccurate [25]. Such data should be well studied before using for testing. Traditional testing methods usually use accurate values as test cases. However, for testing context-aware applications, especially those obtaining data directly from sensors, it is reasonable for testing engineers to question the reliability of the data.

Vaninha et al. [25] illustrate the relationships between the context sources (sensors or software) and defect patterns. They show that context sources are closely related to faults of several types: incompleteness, inconsistency, sensor noise, slow sensing, granularity mismatch, problematic rule logic, and overlapping sensors. Each fault type is caused by one ore more failures in context sources, such a Camera, GPS, or WiFi. Table 1 (borrowed from [25]) shows the relationship between context sources and fault types, e.g., ambiguity, as one form of incomplete, may be caused by errors in the context source of RFID/NFC, QR-CODE or Clock/Alarm.

The problem of inaccuracy in context data can cause a high-level defect called context inconsistency, which may relate to multiple context sources or is a defect in interpretation from context [25].

3.1.2 Context Inconsistency

Context inconsistency occurs when there is at least one contradiction in a computation task's context [27]. It can be caused by sensor errors or sensor data inaccuracy [11] [12] [25] [26]. Asynchronous updating of context information can also cause the same problem [13]. As a result of the possible

Table 1. Context-Sources in Combination with Defect Patterns [25]

Context-Source	Incomplete			Sensor Noise				Slow Sensing		Overlapping Sensors	
	Unavailability	Not Interpretable	Ambiguity	Incorrectness	False Reading	Instability	Unreliability	Out-of-Dateness	Wrong Interpretation	Concurrent Values	Unpredictable
Accelerometer	X			X	X			X	X		
Wi-Fi	X						X	X	X		X
Camera	X	X						X	X		
RFID/NFC	X	X	X	X	X						
QR-Code	X	X	X	X	X						
GPS	X			X	X		X	X	X		
Light Sensor	X			X	X			X	X		
Clock/Alarm	X		X					X	X		
Calendar	X			X	X			X	X		
Gyroscope	X							X	X		

inconsistency of context, the application logic that rely on the context can lead to wrong behaviors or execution errors.

We can illustrate context contradiction using the WiFi access point (WAP) application where WAP can be used to detect the location of a device connected to it [11]. Suppose in a location identification service, WAP installed in each room of a building is supposed to detect the location of a person who is wearing a smart device. The smart device has a unique identification for each person. Context inconsistency may happen in the following situation: if WAP S1 installed in room R1 detects person P and claims that P is in R1 now and meanwhile WAP S2 embedded in room R2 detects the same person P and claims P is in R2. This type of inconsistency can happen in the following scenarios: rooms R1 and R2 are near each other or they are in the same coordinates of nearby floor.

Context-aware applications can get raw data from a single sensor or several sensors, and they can also get synthesized context data from middleware [29], which collects data from sensors as well. Raw data from a single sensor have great opportunities to exhibit inconsistency problems, however, data from a middleware, which does not apply consistency checking, may also experience inconsistency problems.

Chang et al. [27] try to solve the inconsistency problem using a framework for realizing dynamic context consistency management. Based on a semantic matching and inconsistency-triggering model, the framework can detect inconsistency problems. The framework also applies inconsistency resolution with proactive actions to context sources.

3.1.3 Continuous Context

Continuous context is used in many context-aware applications [29]. Challenges may arise when applying boundary testing in a continuous context. A straightforward way of modeling continuous context is to directly convert it into discrete one by dividing it into different time windows [29] [32]. Hidasi et al. [32] demonstrate that much information will be lost if such modeling approach is used. This missing information can be the boundary values, which will greatly affect the effectiveness of testing with the technique of boundary value analysis. To build better models for continuous context, Hidasis et al. propose fuzzy modeling approaches. The fuzzy modeling method advocates that context-state is not only associated with the interval it belongs to, but is also influenced by its relative location in the interval and neighboring intervals. Thus, a better understanding of the event or context-state with respect to a specific interval can be achieved, in which way the information loss of boundary values can be complemented.

For context data collection, Chen et al. [31] define *snapshot* as the union of all sensing values at a particular timestamp. The act of collecting multiple continuous snapshots is called continuous data collection (CDC) [30]. Their work focuses on challenges of network capacity, while Nath's [29] work concentrates on reducing sensing cost using a middleware approach when continuous context sensing is required.

3.2 Adequacy Criteria

Testing adequacy criterion is usually defined as a rule or a collection of rules a test set should satisfy [36]. To measure how well a program is examined by a test suite, usually one or more criteria are used. A variety of testing adequacy criteria have been developed for traditional testing while only a few are suitable for testing context-aware software. According to the work of Lu et al. [28], there are three kinds of obstacles that hinder the effective application of standard data flow testing criteria to testing Context-aware Middleware-Centric (CM-Centric) software, namely,

1) *Context-aware faults*: faults in the triggering logics in the middleware;
2) *Environmental interplay*: environmental updates may happen anytime, and test set should be updated in time accordingly;
3) *Context-aware control flow*: it is difficult to enumerate every control flow trace of context changing for some situations.

Recent research is using special approaches to generate testing criteria for context-aware software [26] [28]. For instance, Lu et al. [28] have applied a data flow method to generate adequacy criteria for testing middleware-centric context-aware programs. Different from traditional variables, a context variable can be defined and updated via either an assignment or an environmental update. Therefore, a new definition of "*definition (DEF) of variables*" and "*usages (USES) of variables*" are given, as well as "*update-use occurrences of variables*", which refers to an occurrences of a context definition due to sensing of environmental contexts and a context use. Imitating the conventional def-use (DU) associations, the paper provides definitions of def-use associations for CM-Centric programs, as well as a definition for the pairwise DU associations. Using the defined data flow associations, they generate novel test adequacy criteria to measure the quality of a test set for a CM-centric program.

3.3 Adaptation

Adaptation is the core process of using context for computing in context-aware software. In this subsection, we introduce testing challenges of context-aware software in *adaptation* activities. We explain the challenges in two perspectives: Erroneous adaptation rules and continuous adaptation.

Adaptation rules can be erroneous. Realizing that adaptation rules play an important portion in middleware based context-aware applications, the work of Sama et al. [13] is focused on fault detection in adaptation rules. In their approach, detection is driven by the requirement that the rules and its finite state machine satisfy the following properties: *Determinism*, *State Liveness*, *Rule Liveness*, *Stability*, *Reachability*. For example, determinism requires that for each state of the finite state machine and each possible assignment of values to the context variables in that state, the assignment of the value can only trigger at most one rule.

Continuous adaptation makes it hard to identify which adaptation rule have caused the faults, so it is difficult to set up an effective test oracle [33]. Xu et al. [33] suggest that for context-aware applications, the adaptation to the environmental changes may contain defects when the complexity of modeling all environmental changes is beyond a developer's ability. Such defects can cause failures to the adaptation and result in application crash or freezing. More importantly, they argue that tracking an obvious failure of the system back to the root cause in adaptation is generally difficult [33]. The reasons are as follows. Firstly, a failure is usually a consequence of multiply adaptations, and it is difficult to set up an effective test oracle. Secondly, when a failure happens, it is hard to collect all the context data because some of the data are from outside sensors. Thirdly, it is hard to repeat an observed failure. In their work, they propose a novel approach, called ADAM (adaptation modeling), to assist identifying defects in the context-aware adaptation.

3.4 Testing Execution

Testing execution refers to the process of executing a test plan, in which all the challenges mentioned in above categories should be considered. It not only needs to consider making test plans to resolve aforementioned challenges, but also to realize them by creating novel tools or mechanisms. In this subsection, we discuss the challenges of generating context for testing and introduce an open topic that new mechanisms are necessary for facilitating testing execution.

3.4.1 Context Testing Data Generation

Context can be complex and plenty of work has concentrated on context testing data generation. Two approaches can be used to provide context test information: real world testing and simulator testing. Real world testing means to evaluate an application in real devices with multiple sensors and network conditions. Repeated real-world testing can be expensive in time and effort, sometimes even infeasible when context and environment are complex, e.g. aerospace. However, real-world testing is still highly recommended before the acceptance or commercialization of an application.

Simulator testing can be an alternative when real-world testing is expensive or unpractical, and it is a frequently used approach [2] [3] [18] [34]. Designers need a set of models and tools that aim to achieve the objective of "design for reality". In real world, as we have discussed, context derived from sensors can be inaccurate, inconsistent, and continuous. Besides, sensor reading and network connections may strongly depend on the providers of sensors and networks. Thus it is very challengeable to build a well-equipped simulator. Eleanor et al. [18] propose a testing platform for the user-centered design and evaluation of context-aware services by using a 3D virtual reality simulation to show the environment to users and generate the simulated environment's context. They recognize that to simulate the sensors is very difficult.

3.4.2 Adoption of New Mechanism

Some new mechanisms have been adopted to facilitate context-aware software testing. Griebe et al. [1] use model transformation approach on context-enriched design-time system models to generate platform specific and technology specific test cases. For fulfilling testing criteria, Wang et al. [7] use a component of *Context Interleave Generator* to form potential context interleaving that may be of value a context-coverage criterion requires.

When an observed failure happens, repeating it is a common method to track to its original defect. Collecting all the runtime information can help to achieve this purpose. However, when data is from outside sensors, the task can be difficult [33]. Asynchronous updating of context information can also lead to inconsistencies between external states and internal states. To our best knowledge, these problems have not been thoroughly discussed and new methods for resolving them needs to be explored.

4 Relationships among Challenges

In this section we give our analysis of the relationship among the four identified categories of challenges. As shown in Figure 1, an ontology diagram is built to illustrate these challenges and their relationships.

There are two outstanding features in context testing data, data defects and being continuous, which greatly affect the generation and usage of testing data. Testing criteria are used to evaluate how well software can be tested. The criteria can be used to direct testing data generation and usage, and are also related to adaptation and testing execution. Adaptation can be erroneous and continuous. It should consider context-testing data because continuous context can affect the adaptation as discussed in section 3. Testing execution should not only consider all the challenges from aforementioned categories, but it also needs to consider new mechanisms for implementation of testing plans, e. g., collecting run time data.

5 Conclusion and Future Work

In this paper, we study the challenges of testing context-aware software, divide them into four categories and present the solutions current researchers use to overcome those challenges. After analyzing the relationship among the challenges of the four categories, we developed an ontology diagram to represent the challenges and their relationships. As far as we know, there is no automatic testing framework that considers all of the above challenges. We are currently building such a framework as an execution platform to ease the difficulty of testing context aware software. We will concentrate on addressing the challenges mentioned in the category of testing execution. Since context plays an important part in assuring the quality of context-aware software, we also plan to collect data from context-aware software testing processes and try to find the fault patterns that lead to system error or failure with respect to data inconsistency and adaptation.

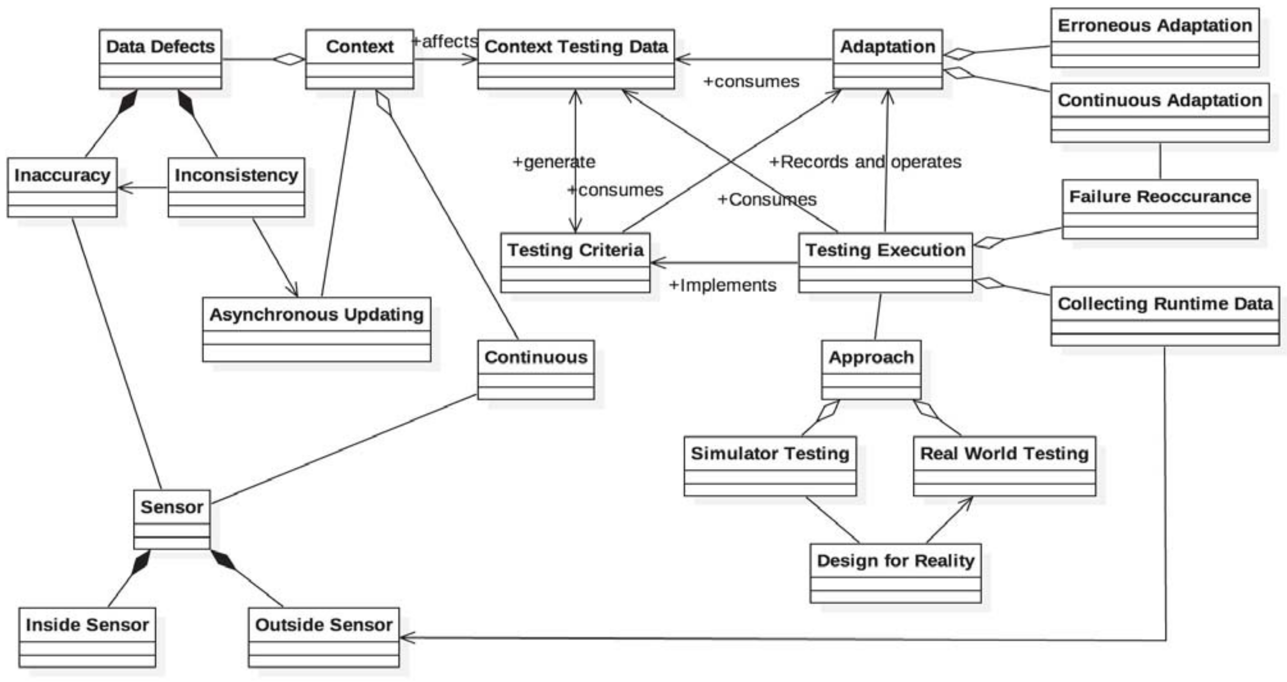

Figure 1: The ontology of identified testing challenges and their relationships

References

[1] Tobias Griebe, Volker Gruhn. “A model-based approach to test automation for context-aware mobile applications”. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14). ACM, New York, NY, USA, 420-427. 2014

[2] Vaninha Vieira, Konstantin Holl, and Michael Hassel. “A context simulator as testing support for mobile apps”. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). ACM, New York, NY, USA, 535-541. 2015

[3] Minsu Jang, Jaehong Kim, Joo-Chan Sohn. "Simulation framework for testing context-aware ubiquitous applications" ICACT 2005. The 7th International Conference on Advanced Communication Technology, vol.2, no., pp.1337-1340, 0-0 0. 2005

[4] Hao Yan and Ted Selker. “Context-aware office assistant”. In Proceedings of the 5th international conference on Intelligent user interfaces (IUI '00). ACM, New York, NY, USA, 276-279. 2000

[5] Sven Meyer and Andry Rakotonirainy. “A survey of research on context-aware homes”. In Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003 - Volume 21 (ACSW Frontiers '03), Chris Johnson, Paul Montague, and Chris Steketee (Eds.), Vol. 21. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 159-168. 2003

[6] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. “A survey on context-aware systems”. Int. J. Ad Hoc Ubiquitous Comput. 2, 4 (June 2007), 263-277. 2007

[7] Zhimin Wang, Sebastian Elbaum, David Rosenblum. "Automated Generation of Context-Aware Tests" ICSE 2007. 29th International Conference on Software Engineering, vol., no., pp.406,415, 20-26. 2007

[8] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal. "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities" HPCC '08. 10th IEEE International Conference on High Performance Computing and Communications, vol., no., pp.5-13, 25-27. 2008

[9] https://blogs.synopsys.com/configurablethoughts/2012/05/sensing-your-world/

[10] Stefano Ceri, Florian Daniel, Maristella Matera, and Federico M. Facca. “Model-driven development of context-aware Web applications”. ACM Trans. Internet Technol. 7, 1, Article 2 . 2007

[11] Dik Lun Lee, Qiuxia Chen. “A model-based WiFi localization method”. In Proceedings of the 2nd international conference on Scalable information systems (InfoScale '07). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, Article 40 , 7 pages. 2007

[12] Jalal Mahmud, Jeffrey Nichols, and Clemens Drews. “Home Location Identification of Twitter Users”. ACM Trans. Intell. Syst. Technol. 5, 3, Article 47, 21 pages. 2014

[13] Michele Sama, David S. Rosenblum, Zhimin Wang, and Sebastian Elbaum. “Model-based fault detection in context-aware adaptive applications”. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (SIGSOFT '08/FSE-16). ACM, New York, NY, USA, 261-271. 2008

[14] Bill N. Schilit, Marvin M. Theimer, “Disseminating Active Map Information to Mobile Hosts”. IEEE Network, 8(5) 22-32. 1994

[15] Brown, P.J. “The Stick-e Document: a Framework for Creating Context-Aware Applications”. Electronic Publishing ’96 259-272. 1996

[16] Dey, A.K., Abowd, G.D., Wood, A. “CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services”. Knowledge-Based Systems, 11 3-13. 1999

[17] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, Pete Steggles. “Towards a Better Understanding of Context and Context-Awareness”. HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. Publisher: Springer-Verlag. September 1999

[18] Eleanor O'Neill, David Lewis, Kris McGlinn, and Simon Dobson. “Rapid user-centred evaluation for context-aware systems”. In Proceedings of the 13th international conference on Interactive systems: Design, specification, and verification (DSVIS'06), Gavin Doherty and Ann Blandford (Eds.). Springer-Verlag, Berlin, Heidelberg, 220-233. 2006

[19] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. “A survey on context-aware systems”. Int. J. Ad Hoc Ubiquitous Comput. 2, 4, 263-277. June 2007

[20] Licia Capra, Wolfgang Emmerich, Cecilia Mascolo. "CARISMA: context-aware reflective middleware system for mobile applications". IEEE Transactions on Software Engineering, vol.29, no.10, pp.929,945, Oct. 2003

[21] Kristian Ellebæk Kjær. “A survey of context-aware middleware”. In Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering (SE'07), W. Hasselbring (Ed.). ACTA Press, Anaheim, CA, USA, 148-155. 2007

[22] Tao Gu, Hung Keng Pung, Da Qing Zhang, “A service-oriented middleware for building context-aware services”, Journal of Network and Computer Applications, Volume 28, Issue 1, Pages 1-18, ISSN 1084-8045. January 2005

[23] Qin, Weijun; Shi, Yuanchun; Suo, Yue, “Ontology-based context-aware middleware for smart spaces”. Tsinghua Science and Technology , vol.12, no.6, pp.707,713, Dec. 2007

[24] Di Zheng; Hang Yan; Jun Wang, “Research of the Middleware Based Quality Management for Context-Aware Pervasive Applications”. 2011 International Conference on Computer and Management (CAMAN), vol., no., pp.1,4, 19-21. May 2011

[25] Vaninha Vieira, Konstantin Holl, and Michael Hassel. “A context simulator as testing support for mobile apps”. In Proceedings of the 30th Annual ACM Symposium on

Applied Computing (SAC '15). ACM, New York, NY, USA, 535-541. 2015

[26] Heng Lu, Chan W.K., Tse T.H.. "Testing pervasive software in the presence of context inconsistency resolution services". ICSE '08. ACM/IEEE 30th International Conference on Software Engineering, vol., no., pp.61,70, 10-18 May 2008

[27] Chang Xu and S. C. Cheung. "Inconsistency detection and resolution for contextaware middleware support". In Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-13). ACM, New York, NY, USA, 336-345. 2005

[28] Heng Lu, W. K. Chan, T. H. Tse. "Testing context-aware middleware-centric programs: a data flow approach and an RFID-based experimentation". SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. November 2006

[29] Suman Nath. "ACE: exploiting correlation for energy-efficient and continuous context sensing". In Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys '12). ACM, New York, NY, USA, 29-42. 2012

[30] Shouling Ji, Jing (Selena) He, A. Selcuk Uluagac, Raheem Beyah, and Yingshu Li. "Cell-based snapshot and continuous data collection in wireless sensor networks". *ACM Trans. Sen. Netw*. 9, 4, Article 47 (July 2013), 29 pages. 2013

[31] Siyuan Chen, Shaojie Tang, Minsu Huang, Yu Wang. "Capacity of Data Collection in Arbitrary Wireless Sensor Networks" in *INFOCOM, 2010 Proceedings IEEE* , vol., no., pp.1-5, 14-19. March 2010

[32] Balázs Hidasi and Domonkos Tikk. "Approximate modeling of continuous context in factorization algorithms". In *Proceedings of the 4th Workshop on Context-Awareness in Retrieval and Recommendation* (CARR '14). ACM, New York, NY, USA, 3-9. 2014

[33] Chang Xu, S.C. Cheung, Xiaoxing Ma, Chun Cao, Jian Lu. "Adam: Identifying defects in context-aware adaptation". Journal of Systems and Software, Volume 85, Issue 12, Pages 2812-2828, ISSN 0164-1212. December 2012

[34] Stefan Taranu and Jens Tiemann. "General method for testing context aware applications". In Proceedings of the 6th international workshop on Managing ubiquitous communications and services (MUCS '09). ACM, New York, NY, USA, 3-8. 2009

[35] Edwin J.Y. Wei, Alvin T.S. Chan. "CAMPUS: A middleware for automated context-aware adaptation decision making at run time". Pervasive and Mobile Computing, Volume 9, Issue 1, Pages 35-56, ISSN 1574-1192. February 2013

[36] Paul Ammann and Jeff Offutt. "Introduction to software testing". Cambridge University Press. 2008