Towards Fault Tolerant Pervasive Computing

Shiva Chetan, Anand Ranganathan and Roy Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
{chetan, ranganat, rhc}@uiuc.edu.

Abstract

Pervasive computing envisions an environment that seamlessly integrates digital and physical devices. Users can access digital data and applications from the environment as easily as accessing them through their computers. Since pervasive computing exists in the user's environment, the technology is sustainable if it is invisible to the user and does not intrude the user's consciousness [3]. This requires that functioning of the multitude of devices in the environment be oblivious to the user. Therefore, the system has to be resilient to various kinds of faults and should be able to function despite faults. In this paper, we discuss various classes of failures, their implications to pervasive computing and the challenges to be addressed in designing a fault tolerant pervasive computing system. We also describe our prototype fault tolerant pervasive system and propose future directions.

1. Introduction

Pervasive computing ushers in a new era of computing that integrates digital and physical devices. It envisions a world of computers, handheld devices, sensors and actuators integrated seamlessly with everyday physical devices such as electrical appliances and automobiles. In addition, pervasive computing provides a platform for context-aware computing [1,2] that enables automatic configuration of a pervasive system based on the environment context.

Mark Weiser in his paper "The Computer for the Twenty-First Century" defines pervasive computing as a technology that "weaves itself into the fabric of everyday life until it is indistinguishable from it" [3]. He mentions that for pervasive computing to be successful, it's functioning should be transparent to the user. Such transparency is achievable if faults in the system are masked and user intervention is sought only when absolutely required.

Pervasive computing technology exists in the user's environment and aids the user in performing various tasks. The sustainability of this technology depends on it being non-intrusive. In order to achieve this goal, faults in a pervasive system should be automatically masked and user notified only when absolutely required.

Fault Tolerance issues have not been well explored so far in pervasive computing research. Since pervasive computing environments operate in the same physical (as well as virtual) space as humans, they can be exasperating (and sometimes hazardous) if they are not resilient to faults. Several researchers have expressed the need for reliable pervasive systems and mention that reliability issues must be readdressed in the realm of pervasive computing. [4] mentions that one of the paramount concerns of smart home occupants is reliability. In [5], the authors mention that traditional fault detection and recovery techniques would need to be modified to fit the needs of pervasive computing. [6] discusses dependability requirements of pervasive computing in a healthcare environment.

Pervasive computing is finding immediate applications in healthcare facilities [7], aware-homes [8] and assisted-living for the elderly [9]. Sensors are used to monitor conditions of patients in hospitals, onset of age-related disorders in the elderly and status of various electrical appliances in aware-homes. Failures in such scenarios can lead to disasters and so fault tolerance is vital.

The ramifications of faults in a pervasive system can stretch beyond immediate consequences. Faults can lead to incorrect context sensing, security and privacy breaches and misuse of resources. Therefore, fault containment is a very important aspect of deploying a pervasive system into the physical world.

We have incorporated some fault-tolerance mechanisms in our prototype Pervasive Computing

Environment [10]. While our current system handles some kinds of failures, much research still remains to be done to make it a comprehensive system that is completely fault-tolerant (if it is ever possible to build such a system). The aim of this paper is to highlight the various challenges and issues that confront fault tolerant pervasive computing, present some solutions to these problems and describe how some of these solutions are implemented in our system.

In this paper, we discuss various issues involved in designing a fault tolerant pervasive system. In section 2 we classify various failures in a pervasive system. We discuss the implications of those failures in section 3 and present various challenges and suggest solutions in sections 4 and 5. In section 6, we present our fault tolerant pervasive system that tolerates some application and device faults and discuss its fault handling techniques. In Section 7, we discuss future fault tolerance enhancements planned for our pervasive system. We cite a few pervasive system projects that address fault tolerance issues in section 8 and discuss their techniques. We finally conclude the paper in section 9.

2. Classification of Failures

A typical pervasive system consists of commercial off-the-shelf (COTS) software and devices whose reliability is not guaranteed. COTS software are sold as "black boxes" and may not be subject to rigid development, verification or testing processes [11]. Interoperability issues further reduce the reliability of a pervasive system. Mobile devices such as handhelds and laptops, with limited battery power, cannot be regarded as totally reliable. Connectivity failures due to devices going out of range or other errors in networks add to faults in a pervasive system. Besides, a pervasive system has a core set of services (like naming, trading, file system, event delivery, discovery and context services) that provide necessary functionality. These services can also fail. Broadly, faults in a pervasive system can be classified into device, application, network and service failures. We discuss these individually in the following sections.

2.1. Device Failures

A pervasive system consists of different kinds of devices such as desktops, laptops, handhelds, sensors, actuators, displays, speakers, scanners, cameras and projectors. Each device has its own set of faults that can potentially contribute to the failure of the pervasive system. Mobile devices, such as laptops and handhelds, have physical constraints such as finite battery power and limited signal strength. So if the battery goes down or if the signal strength is too low they get disconnected from the pervasive system and are regarded as having failed. A more acute problem with devices is when they are alive but operate incorrectly. This is common in faulty sensors and is called a Byzantine failure [12].

2.2. Application Failures

Designing reliable software is an expensive process and the cost of debugging, testing and verifying can easily range from 50 to 75 percent of the total development cost [13]. Even in well-tested software systems, bugs of varying severity are found [14]. Pervasive computing includes commercial off-the-shelf applications that may not be well tested. In some situations, applications may work well as stand-alone software but may not inter-operate correctly or reliably with other software. Therefore, pervasive systems should make few reliability assumptions about applications.

Application failures include application crashes due to bugs, operating system errors, unhandled exceptions and faulty usage. Pervasive applications are also likely targets for malicious software such as viruses and worms. Viruses and worms cause fail-stop [15] or Byzantine failures.

2.3. Network Failures

Pervasive systems consist of wired and wireless devices. Therefore, a reliable pervasive system should account for network failures caused by low signal strength, devices going out of range and unavailability of communication channels due to heavy traffic. Network failures lead to unreachable devices that may be wrongly perceived as device failures. Automatic detection of the failure type is an important issue in pervasive computing.

2.4. Service Failures

As mentioned above, a pervasive system is supported by various services that enable different functionalities. Some of these services are essential while others add features to a pervasive system. Essential services include naming, event and discovery services. Some pervasive systems support other services such as a trading service that enables device discovery, context services that enable context-aware computing and file system services for ubiquitous data

access. Examples of service failures include service crashes due to bugs and operating system errors, faulty operation of services like sensing incorrect context, wrong inferring and lossy delivery of events. Service failures can potentially lead to failure of the pervasive system.

3. Implications of Failures

Pervasive computing integrates digital devices seamlessly in our physical environment. Digital devices co-exist with physical devices to aid in accomplishing everyday tasks. Therefore, faults in pervasive systems can be bothersome and result in user annoyance. Consider, for instance, an aware-house [8] that uses radio-frequency badges to identify users. When a user enters the house, the pervasive system identifies him and configures the house to meet his requirements. It adjusts the temperature, turns on his favorite television channel and preferred lights in the house. Failure to correctly identify the person can result in a different configuration and can be a source of annoyance to the user.

3.1. Hazard to Life

Pervasive systems are being deployed in healthcare [7] and assisted-living facilities [9] for the elderly. These systems monitor conditions of patients and automatically request assistance. In assisted-living facilities, pervasive systems are used to identify age-related disorders in elderly by observing their everyday activities. Therefore, users of such pervasive systems rely heavily on them. Failures in such systems can be hazardous and can result in loss of life.

3.2. Inappropriate Resource Control and Usage

A pervasive system controls several digital and physical resources. Faults can lead to inappropriate control and usage of these resources. For instance, faults in a temperature sensing system could lead to over-heating or over-cooling of the physical space. Similarly, faults in an intrusion detection system can cause false alarms. In wireless devices, retransmissions due to failed network channel leads to excessive consumption of battery power.

3.3. Security Vulnerabilities

The ubiquity of deployment of pervasive systems necessitates robust security mechanisms for access control and authentication. Faults can lead to security breaches and consequent compromise of trust. A failed intrusion detection system may not detect an intruder while failure in an authentication system may let users misuse resources and data.

3.4. Inferring Incorrect Context

One of the key elements of a pervasive system is *context*. Context information is used to proactively configure and adapt the environment to meet a user's or group's needs [1,2].

Various sensors are used to sense context. These include cameras, voice recognition systems, temperature sensors and RF identifiers. Inputs from faulty sensors lead to inaccurate detection of context. A faulty light sensor may erroneously sense the level of natural light in a room to be too low and unnecessarily cause artificial lights in the room to be turned on. In systems that use RF badges for sensing location of people, a faulty RF badge may result in inaccurately determining the location of a person. This could have security implications if location is used to give people certain access control rights.

A bigger challenge in context usage is in inferring context accurately. Various methods are used to fuse data from one or more sensors and infer higher-level contexts. For example, rules could be used to infer the activity in a room. However, these rules could sometimes result in the wrong context being inferred. For instance, a rule such as *"If the number of people in a room is more than two and a PowerPoint application is running, then there is a presentation going on in the room"* [2] may be correct most of the times. However, there could be situations when it does not hold - for example, if there are two people in the room and one of them is *editing* a PPT file. Incorrect context can potentially lead to inappropriate resource usage, user annoyance or failure of the pervasive system. In the previous example, incorrectly sensing the editing activity as a presentation, the pervasive system might start a presentation recording application, dim the lights in the room and turn on a microphone to be used by the presenter, which are all inappropriate for the editing activity.

4. Challenges Facing Fault Tolerance

The area of fault tolerance in computing systems has been enriched through decades of research. Fault tolerance issues have been addressed in various areas of computing systems such as computer architecture, operating systems, distributed systems, mobile

computing and computer networks. Despite these advances in tolerating faults, each new area poses its own set of challenges for which the past techniques have limited applicability. In this section, we discuss fault tolerance issues in the realm of pervasive computing.

4.1. Fault Detection

Efficient fault detection is a tough challenge in a pervasive system. An application or device that stops on failure can be detected through timeout techniques such as *heartbeat* messages. An entity sends "I'm alive" messages periodically to the fault detector. If the heartbeat message is not received for a certain interval of time the fault detector senses an error in the entity. In a pervasive system, with a plethora of devices and applications, heartbeat messages significantly add to the network traffic and the number of messages can overwhelm fault detectors. Further, network failures can lead to unreachable nodes making it complex to distinguish between entity failure and network failure.

Byzantine fault detection is a tougher problem. The Byzantine fault model includes failures in which entities do not stop but operate incorrectly. Byzantine faults result in inferring incorrect context and inappropriate resource usage. Heartbeat messages cannot be used to detect all Byzantine faults, as entities do not stop sending heartbeats.

4.2. Fault Containment

Once a fault is identified, it should be isolated to prevent its propagation to other parts of the system. A pervasive system contains inter-dependent applications and services that communicate frequently. This communication fosters fault propagation and makes fault containment a tough challenge. For instance, consider a "smart" room that infers an activity based on the people in the room and applications being used. A faulty RF id sensor can result in wrongly sensing the number of people in the room. This can, in turn, result in inferring the wrong activity currently taking place in the room and configuring the room inappropriately.

4.3. Transparent Fault Tolerance

Mark Weiser envisioned pervasive computing as a system that blends in with the physical environment and whose functioning is transparent to the user. In order to realize this vision, faults should be tolerated with minimal user awareness.

4.4. Good Fault Reporting Mechanisms

When a fault cannot be tolerated, it should be reported to the user in a non-intrusive manner so as to cause minimal disruption to user's activity. Faults can be reported through visual representation on display devices, audio representation on speakers or any other means that can be perceived by the human senses. Determining appropriate means of reporting faults is an interesting problem for research and involves various parameters such as user preferences, resource availability and user location.

5. Fault Tolerance Approaches

The goal of a fault-tolerant system is to mask faults and continue to provide service despite faults. All fault tolerance techniques use some form of redundancy to tolerate faults. Depending on the class of faults redundant devices, networks, data or applications are used.

In this section, we discuss several approaches that can be employed in pervasive computing for tolerating faults. Each approach tackles one or more classes of failures.

5.1. Surrogate Application/Device Usage

A common fault tolerance technique is to detect failure of a process and restart it. In order to minimize the loss of computation, the state of the process is periodically stored on a stable storage device. Upon failure, the process is restarted using the stored state. This technique can be used in a pervasive system to tolerate some application and device faults. We have implemented this scheme in our pervasive system and we will discuss it in section 6.

If devices fail due to hardware problems, applications on the device cannot be restarted on the same device. So the solution is to find a surrogate device that can provide the same functionality as the failed device and restart the application on it. If the surrogate device cannot support the execution of the same application, an equivalent application that provides similar functionality can be used. For instance, if a PDA running a WinAmp music player fails, a laptop running Windows Media Player can be used since it provides functionality similar to the WinAmp player. There are other issues such as availability, user preferences and security that need to be considered while making this choice.

5.2. Alternate Notification Mechanisms

Pervasive computing can offer multiple ways of reaching users. For example, in an assisted living facility, pervasive systems are used to monitor the status of facility users. In an emergency, the system notifies healthcare personnel for assistance. The system can notify the personnel through their cell phones, pagers, calling help lines (911 in the US) or as a text message on display devices or speakers in an emergency monitoring station. So these "channels" of communication provide inherent redundancy in the pervasive system that the system can leverage to tolerate faults. If the system discovers that a notification device has failed it should reroute the message through a different channel of communication.

5.3. Handling errors in sensing and inferring context

Detecting errors that may occur while sensing and inferring contexts is not easy. The primary way of detecting and handling such errors is by employing redundancy. Multiple sensors that sense the same (or similar) pieces of information can be used to overcome errors by one or more sensors. For example, in our prototype pervasive environment [16], we deploy many badge detectors in the same room so that even if some of them do not detect the presence of a badge (due to interference, physical occlusion or other reasons), there are others that do.

Similarly, using multiple algorithms to infer the same higher-level context from sensed contexts can help overcome wrong inferences made by one or more such algorithms. Such meta-level learning and inferring approaches are widely used in approaches like boosting [17]. They help in increasing the level of accuracy of the inferences made.

Finally, however, until other techniques become more effective, the most reliable way of detecting an error in sensed or inferred context would be to allow users to indicate any errors that they may observe. For example, [18] allows users to mediate context sensing. More generally, this would require intuitive user interfaces that show users what contexts were sensed or inferred and allow them to correct anything that was sensed or inferred wrongly.

5.4. N-version approach

In an N-version software system [19], each software module is made of N different implementations. Each module performs the same task and submits the result to an arbitrator. The arbitrator determines the correct answer and returns that as the result of the task. In effect, the N-version technique provides functional redundancy.

In the realm of pervasive computing, an N-version system should be more generalized to support different kinds of implementations. For example, in an assisted-living facility, user status can be monitored by different means such as vision inputs, audio inputs and inputs from wearable sensors. An arbitrator should decide on the state of the person through these inputs. This complicates the arbitrator and necessitates usage of inferring techniques.

5.5. Fault Notification Mechanisms

If an unrecoverable fault occurs in a pervasive system, the system should provide good fault notification mechanisms to notify the user. A few research challenges should be addressed in designing a good fault notification system.

If a user uses only a small set of devices, the user would only be concerned about the status of those devices. In such situations, the user should be notified of faults in devices used by the user. Though the user may only be using a small set of devices, the devices may be dependent on various services of the pervasive system. This creates a dependency graph that could span a large number of applications, services and devices. Faults in each of these entities may be reported to the user or to an administrator who is in charge of any failed services. Determining the dependency graph is an important research challenge. Since the user is not responsible for some entities such as system services and file systems, the dependency graph should be appropriately pruned to notify the user of faults that can be addressed by the user. This is a tougher problem to address and could employ context information to prune the dependency graph.

6. Prototype Fault-Tolerant Pervasive Computing System

We envision a pervasive computing system as a device-rich environment that integrates properties of digital and physical devices seamlessly and refer to it as an *Active Space*. Our active space is comprised of digital devices such as laptops, plasma displays, handheld devices, finger print scanners, desktop computers, RFID badges and infrared beacons integrated in a physical space. The Gaia meta-operating system [10] provides a set of services to manage the

active space. The active space supports context-aware computing through the Gaia Context Infrastructure [2].

We have developed a fault tolerance technique that makes use of context information to tolerate application and device faults. We consider a fail-stop fault model [15] consisting of device and application faults. Further, we consider only devices that can host applications such as laptops and handheld devices. Therefore, a device failure is treated as failure of applications running on that device. This technique tolerates application faults that can be masked by restarting applications. Therefore, the fault model only considers application failures caused by transient errors such as device failures, network faults and failures due to faulty usage.

Figure 1 presents the architecture of the fault management system. Applications periodically save their states onto a checkpoint storage. The reliability of the storage can be improved by traditional techniques such as RAID [20] and so we do not address its failures. The checkpoint interval is determined by the application. In our implementation, we store the high-level state of the application as the checkpoint. For instance, the high-level state of a music player is the music clip being played and the elapsed time, while the high-level state for a presentation application is the presentation file name and the slide being displayed. This is done only for convenience and can be easily modified to store the entire memory image of the application.

Each application sends a periodic heartbeat message to the Gaia OS informing that it is alive. Since we consider a fail-stop fault model, when an application fails, it terminates and no longer sends heartbeat messages.

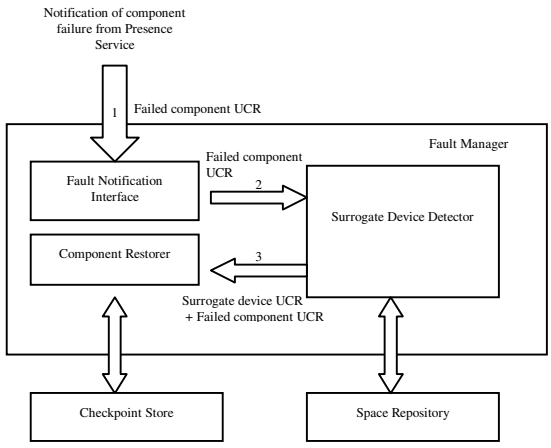

Fig. 1 Architecture of Fault Manager

When the Gaia OS detects the absence of heartbeat messages from a certain application, it notifies the fault manager. The fault manager obtains information about the current context from the context infrastructure and it gets device and application properties from the Space Repository. It uses this information to infer a contextually appropriate surrogate device on which the application can be restarted. The failed application is then restarted on the surrogate device using the saved state from the checkpoint storage. This is called *rollback recovery* and reduces loss of state on failure [21]. The fault management system uses a Prolog-based reasoning mechanism to determine the most appropriate surrogate device. It considers parameters such as availability of the device, user preferences and application capability while reasoning. These parameters can be set by the user.

We employ rule-based techniques for inferring the surrogate device. For example, the rule,

surrogate(D1, D2) :- compat(D1, D2), avail (D2)

specifies that device D2 is a surrogate for device D1 if it is compatible with D1 and is available for usage. The predicates, *compat(D1, D2)* and *avail(D2)* can be inferred from other rules or facts.

Faults are addressed as soon as they are detected by the fault detector. Therefore, fault containment depends on the interval required to detect the fault.

7. Future Work

We plan to extend our work to use the technique of alternate notification mechanisms discussed in section 5. In the present system, when a surrogate device is detected upon failure, the fault manager displays the identity of the surrogate device on a display device. We plan to investigate usage of alternate notification mechanisms for notifying the user of the surrogate device.

The present implementation uses heartbeats for failure detection. Loss of heartbeats can also be due to non-reachability of the device. This situation arises due to network congestion and devices going out of range of the fault detector. N-version techniques for pervasive systems can be used to confirm an application failure. We are also investigating such techniques.

8. Related Work

Many research projects have employed techniques that provide some form of fault tolerance. For example, iROS [22] uses an EventHeap for communication between various entities. The EventHeap is based on the tuplespaces model originally proposed by Linda [23]. The EventHeap decouples entities and so avoids error propagation. But the project does not seem to consider restarting failed applications as we have proposed.

The Ninja project aims to develop a software infrastructure to support Internet-based applications. The architecture is based on the concept of a *service*, which is an Internet-accessible application. The service state is partitioned into a *hard* state and a *soft* state. The hard state is a persistent state and is maintained in a carefully controlled environment, which is engineered for high availability and scalability. Soft state can be regenerated in case of loss. The project addresses application failures but does not seem to address device failures. In our work, we detect an alternate device automatically when a device fails using context information.

The one.world project [24] enhances the robustness of ubiquitous systems by providing transaction-level persistence and support for disconnected operations. The project does not address device or application failures.

Mobile computing addresses various fault tolerance issues that are relevant to pervasive computing. But many fault tolerance problems in mobile computing are complicated by other characteristics of pervasive computing and therefore need to be readdressed [25].

9. Conclusion

Transparency is an important characteristic of pervasive computing. If pervasive computing has to be sustainable, it should be unobtrusive and its faults transparent to the user. This requires that the system automatically mask various kinds of faults that are commonplace in a computing system. In this paper, we have classified faults in a pervasive system and discussed various research challenges. We have also proposed solutions to some of the research challenges.

References

[1] A.K.Dey, D.Salber and G.D.Abowd, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications", *Human-Computer Interaction*, pp. 16(2-4):97-166, 2001.

[2] A.Ranganathan and R.Campbell, "A Middleware for Context-Aware Agents in Ubiquitous Computing Environments", *ACM/IFIP,USENIX International Middleware Conference*, June 2003.

[3] M.Weiser, "The Computer for the 21st Century", in *Scientific American*, 1991, pp. 94-104.

[4] W.K.Edwards and R.E.Grinter, "At Home with Ubiquitous Computing : Seven Challenges", in *Ubicomp 2001*, 2001.

[5] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy B.Sussman, Deborra Zukowski, "Challenges: an application model for pervasive computing", in *Mobile Computing and Networking*, 2000, pp. 266-274.

[6] Jurgen Bohn, Felix Gartner and Harold Vogt, "Dependability Issues of Pervasive Computing in a Healthcare Environment", http://citeseer.nj.nec.com/562030.html

[7] M.Bang, A.Larsson, and H.Eriksson. NOSTOS: A Paper-Based Ubiquitous Computing Healthcare Environment to Support Data Capture and Collaboration. In: *Proccedings of the 2003 AMIA Annual Symposium,* Washington DC, Nov 8-12, 2003. p 46-50.

[8] C.D.Kidd, R.Orr, G.D.Abowd, C.G.Atkeson, I.A.Essa, B.MacIntyre, E.D.Mynatt, T.Starner and W.Newstetter, "The Aware Home: A Living Laboratory for Ubiquitouss Computing Research", *in Cooperative Buildings CoBuild'99*, 1999, pp.191-198.

[9] D.J.Patterson, O.Etzioni, D.Fox and H.Kautz, "Intelligent Ubiquitous Computing to Support Alzheimer's Patients: Enabling the Cognitively Disabled", in *First International Workshop on Ubiquitous Computing for Cognitive Aids UniCog*, 2002.

[10] M.Roman, C.Hess, R.Cerqueira, A.Ranganathan, R.H.Campbell and K.Nahrstedt, "Gaia: A Middleware platform for active spaces", *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 4, pp.65-67, 2002.

[11] L.Sha, J.B.Goodenough and B.Pollak, "Simplex Architecture: Meeting the Challenges of using COTS in High-Reliability Systems", *CrossTalk: The J. Defense Software Eng*., vol. 11, pp.710, Apr. 1998.

[12] Lamport, Shostak and Pease, "The Byzantine Generals Problem", in *Advances in Ultra-Dependable Distributed Systems, N.Suri, C.J.Walter, and M.M.Huue(Eds.) IEEE Computer Society Press*. 1995.

[13] B.Hailpern and P.Santhanam, "Software debugging, testing and verification", *IBM Systems Journal*, vol. 41, no. 1, 2002.

[14] J.A.Whittaker, "What is Software Testing ? And why is it so hard ?", *IEEE Software*, vol. 17, no. 1, pp. 70-79, Jan 2000.

[15] F.B.Schneider, "Byzantine Generals in Action: Implementing Fail-Stop Processors", *ACM Transactions on Computer Systems*, pp. 2(2):145-154, May 1984.

[16] M.Roman, C.Hess, R.Cerqueira, A.Ranganathan, R.H.Campbell, and K.Nahrstedt, "A middleware infrastructure for active spaces", in *IEEE Pervasive Computing*, 2002, vol. 1, pp. 74-83.

[17] Robert E.Schapire, “A Brief Introduction to Boosting”, in IJCAI, 1999, pp.1401-1406, http://citeseer.nj.nec.com/schapire99brief.html

[18] Anind.K.dey, Jennifer Mankoff, Gregory D.Abowd and Scott Carter, “Distributed Mediation of Ambiguous Context in Aware Environments”, in *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST 2002)*, Oct. 2002.

[19] A.Avizienis, “The N-Version Approach to Fault-Tolerant Software”, *IEEE Transactions on Software Engineering*, vol. SE-11, no. 12 pp.1491-1501, 1985.

[20] D.A.Patterson, G.Gibson and R.H.Katz, “A Case for Redundant Arrays on Inexpensive Disks (RAID)”, in *Proceedings on the 18th ACM SIGMOD International Conference on Management of Data*. 1988, pp.109-116, ACM Press.

[21] E.Gelenbe, “A model of roll-back recovery with multiple checkpoints”, in *Proceedings of the 2nd International Conference on Software Engineering*, 1976, p.251-255.

[22] S.R.Ponnekanti, B.Johanson, E.Kiciman and A.Fox, “Portabiloity, Extensibility and Robustness in iROS”, in *Proceedings of PerCom*, Dallas-Fort Worth, Texas, USA, 2003.

[23] D.Gelernter, “Generative Communication in Linda”, *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 1, pp.80-112, 1985.

[24] L.Arnstein, R.Grimm, C.Hung, J. Kang, A.LaMarca, S.Sigurdsson, J.Su and G.Borriello, “Systems support for ubiquitous computing: A case study of two implementations of Labscape”, in *Proceedings of the First International Conference in Pervasive Computing*, 2002, Springer-Verlag, Germany.

[25] M.Satyanarayanan, “Pervasive Computing: Vision and Challengges”, *IEEE Personal Communications*, pp.10-17, Aug. 2001.