An Architecture for Resilient Ubiquitous Systems

Anubis G. M. Rossetto[1], Cláudio F. R. Geyer[1], Carlos O. Rolim[1],
Valderi R. Q. Leithardt[2] and Luciana Arantes[3]
[1]*PPGC, Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil*
[2]*Institute of Technology, Senai, Porto Alegre, Brazil*
[3]*LIP6, University of Paris 6, CNRS, INRIA, 4, Place Jussieu, 75005 Paris, France*

Keywords: Ubiquitous Computing, Self-healing, Healthcare, Fault Tolerance, Fault Detector.

Abstract: With the perspective of ubiquitous computing becoming more common form of technology in our everyday lives, our increasing dependency on these systems will require them to be always available, failure-free, fully operational and safe. They will also enable more activities to be carried out and provide new opportunities for solving problems. In view of the potential offered by ubiquitous computing and the challenges it raises, this work proposes a self-healing architecture to support ubiquitous applications aimed at healthcare The goal is to continuously provide reliable services to meet their requirements despite changes in the environment. We outline the application scenario and proposed architecture, as well as giving a detailed account of its main modules with particular emphasis on the fault detector.

1 INTRODUCTION

Ubiquitous computing has its origins in the visionary work of Marc Weiser who at the beginning of the 1990s predicted that there would be environments saturated with computing devices, with communication capabilities highly integrated with human users (Weiser, 1991). According to Weiser, ubiquitous computing could only be successful, if its functions were transparent to the user. This would allow possible system faults from being masked and would mean that user intervention is only required when absolutely necessary. Ubiquitous computing environments are mainly focused on the interface of a physical environment, where the user seeks to make the devices "invisible" and operate with the minimum of intervention. Thus, the occurrence of a fault in this type of system can be anything from annoying which reduces its applicability/usability, to something that is dangerous and might put the user's life at risk.

According to Weber (2002) entirely foolproof systems are impossible, since failures are inevitable. Thus, the area of fault tolerance attempts to employ mechanisms that provide computing systems with a higher level of confidence. In view of the future scenario, attempts have been made to design continuously evolving systems that constitute complex information infrastructures - from super computers and large data centers to thousands of small portable computers and embedded devices. As a result, it - has been a challenge: to maintain dependability in the systems (Avizienis et al., 2004), despite the occurrence of changes (Laprie, 2008), which is called resilience.

As Kephart and Chess (2003) point out, the management of systems poses a real challenge since evolution of systems brings increasing of total costs of ownership. Thus, one solution is to make the systems more autonomous, to some extent, so as not to depend on human intervention when carrying out basic management tasks. Autonomic computing aims to address the current concerns of complexity and total cost of ownership, since it is able to meet the future needs in pervasive and ubiquitous computing and communications (Sterritt, 2005).

One of the properties of autonomic computing is self-healing. A system designed with this feature is able to identify when its behavior deviates from what is expected and reconfigure itself so that it can correct the deviation (Sterritt, 2005). This property ensures effective and automatic recovery when faults are detected, since it requires not only masking of failures, but the identification of the problem and its repair without any interruption of the service and minimal external intervention. Its

G. M. Rossetto A., F. R. Geyer C., O. Rolim C., R. Q. Leithardt V. and Arantes L..
An Architecture for Resilient Ubiquitous Systems.
DOI: 10.5220/0004910404590464
In *Proceedings of the International Conference on Health Informatics* (HEALTHINF-2014), pages 459-464
ISBN: 978-989-758-010-9

goal is to minimize the number of faults so that applications are kept active and available at all times.

In light of the potential of ubiquitous computing and the challenges it raises, we propose a self-healing architecture to support ubiquitous applications designed for healthcare, to ensure reliable services are continuously provided, even in the face of changes in the environment, as well as to meet its new requirements.

The paper is structured as follows: first we provide the application scenario, then the proposed architecture is outlined in detail; following this there is a discussion of related works and finally the conclusions are given together with suggestions for the next phase of the research.

2 APPLICATION SCENARIO IN HEALTHCARE

This research is concerned with an application scenario in the area of healthcare, particularly for people who require continuous monitoring for some pathological reason.

This application scenario for monitoring people requires the use of several sensors to measure information about the health status of a person in different ways, for instance, vital signs, location, falls, gait patterns, acceleration, variations, balance and symmetry.

On the basis of the collected data, the system can predict disorders and, for example, detect the occurrence of an emergency situation that requires immediate attention, such as a fall. From this perspective, this scenario is critical, since it is dealing with life. Thus, the occurrence of faults in the components can put someone at serious risk.

Research studies into the question of applications in healthcare should be carried out to make ubiquitous computing a real technology in the lives of people. They should be conducted in a transparent way, so as to bring benefits to the work of professionals, improve the quality of life and perhaps in the future act as an alternative means of making savings in health resources.

Thus, the use and expansion of ubiquitous applications largely depends on the security and reliability provided by the environment.

Figure 1 shows the conceived scenario which aims to provide people with monitoring in their homes from various sensors. These sensors are small, have a long battery life and can be deployed in the environment or on the patient's clothing. An intermediary mobile device collects the sensor data and transmits it to a computer base which is also in the environment. The collect of data can also be made directly by the computer base. The computer base processes the data and sends it to a server where it is stored. The data processing can result in a simple log (for historical purposes) or trigger a warning, that can be directed to an emergency situation.

The central infrastructure makes available information that can be accessed by different profiles: doctors, nurses, family, or a health center. Each profile has access to specialized information.

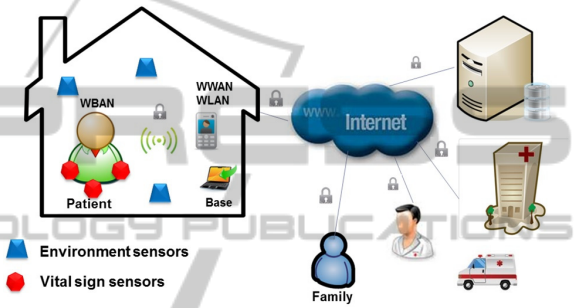

Figure 1: Healthcare Scenario.

3 PROPOSED ARCHITECTURE

Due to the nature of ubiquitous systems, failures are inevitable and may be frequent. For this reason, our aim is to provide an architecture that can continue to provide services even if failures occur. Thus, we proposed an architecture (shown in Figure 2) with different layers to support ubiquitous environments in accordance with the imagined scenario.

The proposed architecture aims to cater for the needs of a scenario where the ubiquitous environment must manage applications that are distributed, mobile, and adaptive to context and available anywhere and at any time. In these environments, detecting a fault in other processes is a key issue. Thus, when a failure is detected, the system has to make the necessary adjustments to avoid error propagation which can result in major damage to the system.

As shown in Figure 2, the proposed architecture has been divided into three layers: **Home**, **Central** and **Interface**. The **Home** layer has the components that are part of every cell in the ubiquitous environment. The **Central** layer contains the components responsible for keeping the complete

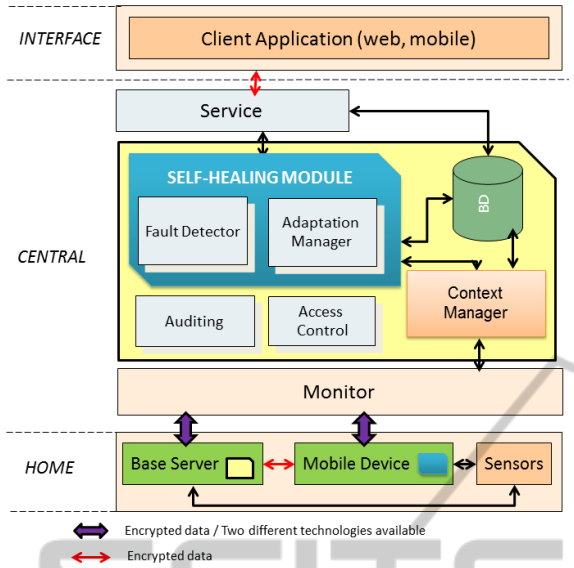

Figure 2: Proposed Architecture.

structure of resources for the ubiquitous environment. And the **Interface** layer is available for applications that seek to access information from the ubiquitous environment.

The ubiquitous environment has what we call here a 'cell', which is a specific structure to respond to the applications in a particular place. The ubiquitous environment is composed of various cells managed by a middleware layer (Central).

Each cell contains a **Base Server** that is responsible for storing and processing data from the sensed context. A cell may contain several sensors, actuators and other mobile devices. The communication between them can occur through different wireless communication protocols (for instance, Bluetooth, NFC, Wi-Fi, and others). In addition, the Base Server has the function of providing the communication with the central server (Central layer) to send data and process the received returns. In the architecture, the data exchanged between mobile devices and the base server must be encrypted in a way that keeps it confidential.

The developed applications for the Interface layer is only concerned with an information query that is available from the Central layer. Since these applications may be any kind of platform (web, mobile), they will have access to the data through a Web Services that is available in the Central layer. This information must also be transmitted on a secure channel (encryption).

The Central layer is made available in a cloud computing structure. Cloud computing makes more efficient use of computing resources, since it has features like availability, elasticity and adaptability of services, and can provide the user with remote access (Armbrust et.al., 2010).

The Central layer has several components with specific responsibilities which are interrelated. The **Monitor** component is responsible for providing interaction with the cell layer (Home). Communication between the layers foresees the presence of multiple communication channels (DSL, mobile,...) through interconnection networks, which makes the system robust with regard to connectivity and faulty nodes.

The Central layer has a database for storing historical data from the ubiquitous system. As well as storing generated information in the system, this database also contains a repository of rules and settings that can be used to determine what adjustments are required when different fault events occur. Another component is the **Context Manager** which is responsible for processing the context and adaptations. This is directly linked to all the other components of the Central layer. The manager is responsible for processing all the contextual information received by the monitor.

The blue frame in Figure 2 defines what is called the **Self-Healing Module** (SHM). Two main components form a part of this module: **Fault Detector** and **Adaptation Manager**. The former consists of an adaptive fault detector that is responsible for detecting faults of different entities that need to be monitored in the system. The Adaptation Manager makes decisions to allow a suitable adaptation strategy to reduce the impact of the failure that is identified. Table 1 shows some examples of failures and their adaptation measures.

The Self-Healing Module is also present in the Home layer of the architecture, in the Base Server and **Mobile Device** components, so that the failure detector is responsible for monitoring all the processes that are at the same level and the information is propagated through the hierarchy.

The deployment of ubiquitous systems requires robust security mechanisms for access control and authentication. Failures in these cases can lead to security breaches and hence impair reliability. Thus, the **Access Control** component is responsible to verifying the authentication of users by giving them restricted access to resources. The basic access policies of this component are defined and stored in the repository, which takes into account aspects of control, role-based access and location. Thus, a user is provided with a digital identity and can be communicated to the system access control.

Table 1: Examples of faults and adaptation.

Fault	Adaptation
without communication between the home and central layers	Local mode operation (store data locally to be sent later)
temperature sensor	Fault notification (responsible), limited operation mode (ignore value)
Smartphone	Backward error recovery
ECG sensor	Fault notification (responsible); Forward recovery.

In addition, since the ability to audit data is very important, all the changes must be the responsibility of some decision system, and made by an autonomous process or specialist, that is, someone must take responsibility for the messages that are being transmitted/stored. In this way, the **Auditing** component will check who is responsible for the information and will record all the actions to allow future audits.

Our main contribution is related to the self-healing module, whose function is to detect faults and make the necessary adjustments to keep the application active and available at all times. Thus, in this paper we focus on the defined characteristics of the Fault Detector component.

3.1 Fault Detector

A failure detector (FD) is a fundamental service that can enable the development of fault tolerant distributed systems (Greve, Sens, Arantes, Simon; 2012). We propose an unreliable fault detector since communications for an application scenario can be considered to be partly synchronous. An unreliable fault detector periodically provides a list of processes that are suspected of having crashed. The partial synchrony model proposed by (Chandra; Toueg, 1996) stipulates that, for every execution, there are bounds on process speeds and message transmission times. The fault detector will handle two types of faults: crash recovery and omission.

In light of the features of ubiquitous environment, we propose a self-tuning failure detector, that is able to calibrate its parameters in order to offer an improved quality of service. Some features were defined to proposed detector:

- Flexibility:

The detector must be able to provide an output that is related to a set of processes (e.g., sensors) and not just to each one individually. This approach can reduce the rate of false responses provided by an unreliable detector. In some scenarios, if one or another sensor fails, it is not serious, but in percentage terms, the system can be badly affected. In this way, the output detector would be a value related to a set of sensors and not a value for each one, as proposed in (Hayashibara et al. 2004).

- Adaptability:

According to Nunes & Jansch-Porto (2004), adaptive detectors have the ability to dynamically adapt their timeout to the behavior of the delay, in accordance with a margin of safety that can enhance the quality of service. The detector must have the ability to self-calibrate with different values of its parameters (for instance, the heartbeat interval, detection time, time of expected arrival ...) according to the requirements of the process (or processes) monitored. Thus, we seek to reduce the number of false suspicions, runtime and number of mistakes, by taking into account that there are periods of burst in the network with loss and delay messages. Moreover, by means of the criteria established for the detector adaptation, it is possible to consider the history of faults and mistakes. In other words, it can be determined if there is a burst of failures in a given period and whether in this same period, the number of mistakes is low, which indicates that multiple sensors are probably faulty and action needs to be taken. Moreover, in some cases, the accuracy of detection, may be pre-defined by the user for each monitored node (or group of nodes), for instance, with regard to hours (at night a finer monitoring process is required but this can be relaxed during the day).

- Economy:

In carrying out its functions, the fault detector receives the control messages, such as the heartbeat mechanism (Arantes; Strike; Sens, 2011). This mechanism is based on two temporal references, in the interval for sending the message *I_am_alive!* (*th*) and in the timeout to wait for the reception of a message.

Every *th* time unit, each monitored component *q* sends a *I_am_alive!* message to the monitor component. If the detector of a monitored component receives a message from *q* before its timeout expires, then the related timeout is restarted. If the detector of the monitor component does not receive a message within timeout, then *q* is added to the list of suspects.

With the aim of measuring the energy consumed by different types of devices that may be in the ubiquitous environment, it will seek strategies to reduce the number of heartbeats, find an appropriate agreement about the detection time, and the number

of mistakes, and frequency of heartbeats. An approach that can be applied is the use of the application message rather than heartbeats (Fetzer; Raynal; Tronel, 2001) (Satzger et al., 2008). Thus, if the application sends a message, the detector considers this message to be an "*I am alive*" message and delays sending the *I_am_alive!* self-message.

- Scalability:

The fault detectors of a distributed system composed by several processes may or may not collaborate with each other. This feature determines their degree of scalability. In this approach, we seek a detector that can be applied to a scenario with a hierarchical organization of modules, where the detector is used in different layers and provides this result to the higher layers.

4 RELATED WORKS

With regard to related work, references were consulted that focus on ubiquitous systems and employ fault handling.

Gaia is a middleware architecture that has the capacity to manage resources in physical spaces (Chetan; Ranganathan; Campbell, 2005). A fault tolerance technique was incorporated in the Gaia system that considers a model of the faults fail-stop type and only devices that can host applications, such as laptops and portable devices. This tolerates application faults that can be masked by the restart of applications; thus, the applications periodically save their states at checkpoints. When it detects that there is a lack of heartbeat messages, it infers a contextually appropriate surrogate device where the application can be restarted (rollback).

The SAFTM is considered to be a fault-tolerant middleware self-adaptive for ubiquitous computing with a dynamic environment in ad hoc networks. The authors propose the architecture of a fault-tolerant system that applies a scheme based on policies that seek to address faults in hardware, software and the network (CAI; PENG; JIANG; ZHANG, 2012). It detects faults by means of continuous monitoring of the state of the component (CPU, memory, OS, I / O, network operations) and dynamically builds the self-adaptive mechanism in accordance with the various types of failures.

The Self-healing unit of MARKS (ad-hoc) middleware is called ETS (efficient, transparent, and secure) Self-healing, and contains a healing manager, to handle faults (Sharmin; Ahmed; Ahamed, 2006). In predicting faults, it conducts an analysis of the changing rate of status of each device (memory, energy, communication signal). With regard to fault containment, it isolates the faulty device and assigns the service to a provider of alternative resources.

The fault tolerant service framework selection (FTSS) keeps track of all the services that are allocated to registered users and monitors whether the execution of the allocated service has been completed successfully (Silas; Ezra; Rajsingh, 2012). When a failure occurs, it waits for '*t*' time to examine whether it is a transient failure. If it is not restored, it delivers a generated report at the checkpoint (checkpoint) to the next service provider.

Table 2: Related Works.

	Focus	Self-healing	Fault Detector	Adaptation
Gaia	Middleware	No	Yes/ Heartbeat	Checkpoint rollback
Marks	Middleware Ad-hoc	Yes	No/ Changing rate of status	Isolation/ alternative resources
FTSS	Framework	No	No/ Monitor execution of service	Checkpoint rollback
SAFTM	MiddlewareAd-hoc	Yes	No/ Monitor the state of device	Dynamic
Proposed	Middleware	Yes	Yes/ Fault detector	Dynamic

In the evaluation of related work, it can be seen that there are limitations with regard to adaptation, with most of the studies employing a fixed criterion for adaptation when failures occur. Moreover, none of the studies employs the use of an adaptive fault detector as proposed in this work. Table 2 summarizes the features of related works.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we have outlined a self-healing architecture that enables the development and execution of ubiquitous applications that are reliable,

and allow their evolution and adaptation in the face of change. This proposal is based on an application in the health scenario which is of critical importance since it deals with people´s lives. Thus, the self-healing mechanism employed in this scenario is essential to ensure effective recovery. It operates automatically when faults are detected, in order not to interrupt the service and only requires a minimum of external intervention, by keeping the applications active and available at all times. We have described the architecture with its layers and main components, while focusing on the characteristics of the Fault Detector.

As a further stage of this research, we are planning to implement the failure detector and make an evaluation with regard to some metrics supplied by the QoS of fault detectors. The purpose of this is to determine how quickly faults can be detected and the exact extent of false detections (Chen; Toueg; Aguilera, 2002).

REFERENCES

Arantes, L., Greve, F., & Sens, P., 2011. Unreliable Failure Detectors for Mobile Ad-hoc Networks. In: Cruz-Cunha, Maria Manuela; Moreira, Fernando. *Handbook of Research on Mobility and Computing: Evolving Technologies and Ubiquitous Impacts*.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., & Zaharia, M., 2010. *A view of cloud computing. Communications of the ACM*, 53(4), 50-58.

Avizienis, A., Laprie, J. C., Randell, B., & Landwehr, C., 2004. Basic concepts and taxonomy of dependable and secure computing. Dependable and Secure Computing, *IEEE Transactions on*, 1(1), 11-33.

Cai, H., Peng, C., Jiang, L., & Zhang, Y. (2012, April). A Novel Self-Adaptive Fault-Tolerant Mechanism and Its Application for a Dynamic Pervasive Computing Environment. In Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012 *15th IEEE International Symposium on* (pp. 48-52). IEEE.

Chandra, T. D., & Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM* (JACM), 43(2), 225-267.

Chen, W., Toueg, S., & Aguilera, M. K. (2002). On the quality of service of failure detectors. Computers, *IEEE Transactions on*, 51(5), 561-580.

Chetan, S., Ranganathan, A., & Campbell, R. (2005). Towards fault tolerance pervasive computing. Technology and Society Magazine, *IEEE*, 24(1), 38-44.

Fetzer, C., Raynal, M., & Tronel, F. (2001). An adaptive failure detection protocol. In Dependable Computing, 2001. *Proceedings. 2001 Pacific Rim International Symposium on* (pp. 146-153). IEEE.

Ganek, A. G., & Corbi, T. A. (2003). The dawning of the autonomic computing era. *IBM systems Journal*, 42(1), 5-18.

Greve, F., Sens, P., Arantes, L., & Simon, V. (2012). Eventually Strong Failure Detector with Unknown Membership. *The Computer Journal*, 55(12), 1507-1524.

Hayashibara, N., Defago, X., Yared, R., & Katayama, T. (2004, October). The φ accrual failure detector. In Reliable Distributed Systems, 2004. *Proceedings of the 23rd IEEE International Symposium on* (pp. 66-78). IEEE.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. Computer, 36(1), 41-50.

Laprie, J. C. (2008, June). From dependability to resilience. *In 38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*.

Nunes, R. C., & Jansch-Pôrto, I. (2002, February). Non-stationary communication delays in failure detectors. *In Proceedings of the 3rd IEEE Latin-American test Workshop* (LATW'02), Montevideo-Uruguay (pp. 16-21).

Satzger, B., Pietzowski, A., Trumler, W., & Ungerer, T. (2008, March). A lazy monitoring approach for heartbeat-style failure detectors. In Availability, Reliability and Security, 2008. ARES 08. *Third International Conference on* (pp. 404-409). IEEE.

Sharmin, M., Ahmed, S., & Ahamed, S. I. (2006, April). MARKS (middleware adaptability for resource discovery, knowledge usability and self-healing) for mobile devices of pervasive computing environments. In Information Technology: New Generations, 2006. ITNG 2006. *Third International Conference on* (pp. 306-313). IEEE.

Silas, S., Ezra, K., & Rajsingh, E. B. (2012). A novel fault tolerant service selection framework for pervasive computing. Human-centric Computing and Information Sciences, 2(1), 1-14.

Sterritt, R. (2005). Autonomic computing. Innovations in systems and software engineering, 1(1), 79-88.

Weber, T. S. (2002). Um roteiro para exploração dos conceitos básicos de tolerância a falhas. Relatório técnico, Instituto de Informática UFRGS.

Weiser, M. (1991). The computer for the 21st century. Scientific american, 265(3), 94-104.