ETS (Efficient, Transparent, and Secured) Self-healing Service for Pervasive Computing Applications

Shameem Ahmed, Moushumi Sharmin, and Sheikh I. Ahamed
(Corresponding author: Sheikh I. Ahamed)

Department of Mathematics, Statistics and Computer Science
Marquette University, 1313 West Wisconsin Avenue, Milwaukee, WI 53233, USA
(Email: {sahmed02, msharmin, iq}@mscs.mu.edu)

(Received Nov. 15, 2005; revised and accepted Jan. 3, 2006)

Abstract

To ensure smooth functioning of numerous handheld devices anywhere anytime, the importance of self-healing mechanism cannot be overlooked. Incorporation of efficient fault detection and recovery in device itself is the quest for long but there is no existing self-healing scheme for devices running in pervasive computing environments that can be claimed as the ultimate solution. Moreover, the highest degree of transparency, security and privacy attainability should also be maintained. ETS Self-healing service, an integral part of our developing middleware named MARKS (Middleware Adaptability for Resource discovery, Knowledge usability, and Self-healing), holds promise for offering all of those functionalities.

Keywords: ETS self-healing, MARKS, pervasive computing, ubiquitous computing

1 Introduction

Ubiquitous computing [22], also known as pervasive computing has evolved during the last few years due to the rapid developments in portable, low-cost, and lightweight devices. It extends human thought and activity as well as provides a pragmatic world augmented by the behavioral context of its users [15].

Despite the prevalence of embedded handheld devices, limited processing capability, restricted battery life, inadequate memory space, slow expensive connections, recurrent line disconnection, confined host bandwidth etc. are the challenges in pervasive computing arena till date [16]. Middleware has evolved to play an important role to cope with these ever-growing requirements.

The system that continues its operation even in presence of faults is termed as fault tolerant system [11]. The contour of self-healing goes beyond fault tolerance since it also provides the device with the capability of recovering from fault by itself or with the assistance of other devices present in the network. Pervasive computing makes it even complex as the resource poor devices are not connected to any fixed infrastructure and the network they work is ad-hoc in nature.

Considerable research has already been done in distributed dependable real-time system [6]. Some solutions along with prototype for pervasive computing fault tolerant systems have been advised [7]. Self-healing autonomous systems are also addressed in [21]. But no solution has been proposed for self-healing system in pervasive computing yet, let alone the implementation of such a system. Since future technology trend lies in pervasive computing, it is utmost important to have an efficient, transparent, and secure self-healing system. Currently, we are developing a middleware named MARKS [1, 2, 19, 20], which is suitable for embedded devices running in pervasive computing environments. The Self-healing unit plays a vital role from the above perspectives. We have named it ETS self-healing and also developed its first prototype on a test bed of PDAs, which are connected with short range ad hoc wireless.

Any healing approach will be in vain without proper setup of security. Efficiency should also not be overlooked. ETS self-healing is unique from those perspectives too. Modified secret sharing approach [18], not only to cope with the limited storage capacity of the embedded devices but also to guarantee the security, is being used in our approach.

Should it be called self-healing if it needs more intervention of users to heal? Simple answer is, no. Being conscious about this, ETS self-healing provides the third feature (transparency) by performing most of the healing process without users' interference.

To auto revive the device from specific fault like system failure (e.g. low battery power, insufficient signal strength etc.) with the help of other devices is an intricate healing

process. Albeit ETS self-healing is capable to handle it in some particular cases, some special kinds' of failures are not addressed in our implementation yet. We have recommended some solutions that we are planning to include in our next prototype.

We detail the approach in the remainder of this paper from the light of ETS. To offer the exhilarating glance of power of ETS self-healing mechanism, some practical scenarios have been described in Section 2. We illustrate the description of our design in Section 3 along with the working procedure that ETS self-healing mechanism follows. How our approach maintains all the aspects of ETS is portrayed in Section 4. The result of investigation of several other researches is mentioned in Section 5. The implementation details along with evaluation are depicted in Section 6. We conclude with some novel directions of our future research in Section 7.

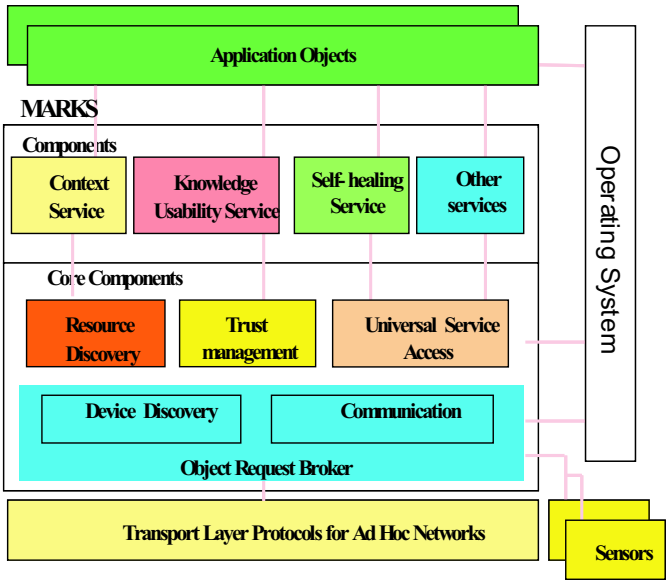


Figure 1: MARKS architecture [1, 2, 19, 20]

2 Motivation

Scenario 1:

A group of high school students appear in a wireless examination. After getting the questions in their PDAs from their teacher Dr. John's PDA (let X, the healing manager), they start their tests. During the exam, all on a sudden, one student's PDA (let Y) starts unusual behavior. After calculating the rate of changes of all of the status of the device, Y finds out that there might have a fault due to the malfunction of a running application. Without any delay, Y sends SOS message along with the student's answer files. X isolates Y from the entire network by removing all entries of Y as a service provider. By this time, Y informs the student about the problem. By using the system interrupt, Y kills that application. As a result, the device starts to operate smoothly.

Scenario 2:

Returning from a visit of a museum, a group of high school kids want to share their experience (stored in their PDAs) to enrich their knowledge. During their information exchange, the healing manager gets informed by SOS message of one device that it is having a high probability of going down and wants to store some of its important information for future use.

To avoid the loss of data stored in that device, healing manager disseminates the stored information to the remaining devices in a secure manner. Consulting the logbook, necessary measures are taken to restore the device's prior working state. Disseminated information content will be refurbished to the device to help it to work to its full extent.

3 ETS Self-healing: Design Overview

ETS Self-healing service is an integral part of MARKS [20]. The fault detection and fault recovery related issues are taken care of by this service. Figure 1 depicts the MARKS architecture along with the self-healing service.

To uphold an effective self-healing scheme, any system should have the following properties:

- No regular functionality of the network will be hampered due to any fault of any device.
- All significant information of the faulty device should be preserved in secured fashion.
- The device will be facilitated to heal its fault by itself or at best with the assistance of other devices of that network.
- After reviving, the faulty device should be able to regain its previous states in such a way that it should feel there was no fault.

To address these challenges in an apposite manner, ETS Self-healing pursues quite a few steps:

- Fault Detection
- Fault Notification
- Faulty Device Isolation
- Alteration
- Information Distribution
- Fault Healing

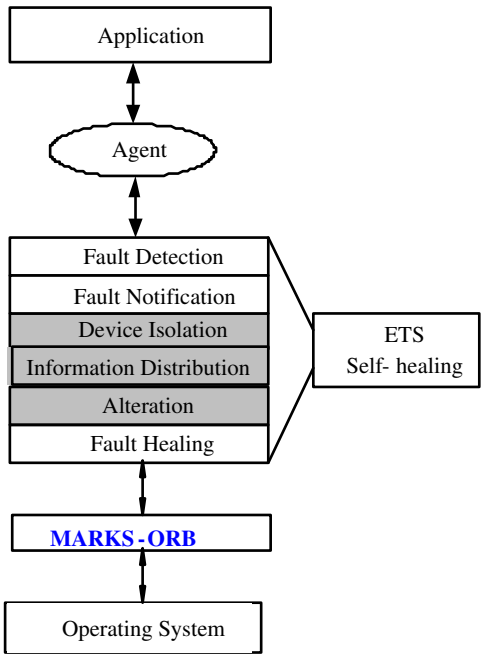


Figure 2: ETS self-healing architecture

Figure 2 shows the architecture of ETS self-healing unit embedded in MARKS. The fault-detection and fault-notification unit is used by each device running MARKS. The fault detection unit may be utilized by both the healing manager and the device itself. But the other three units of ETS Self-healing unit (faulty device isolation, alteration, information distribution) are employed only by the healing manager.

3.1 Fault Detection

High-quality fault detection, the first stride of self-healing process, not only prevents loss of resources but also lessens healing time. To ensure supreme-quality, ETS Self-healing periodically monitors as well as assembles the status of all of the running applications, memory, power, communication signal etc. Drastic changes in those values will generate faults.

By using the rate of change of these over time, it tries to figure out the existence as well as the reason of fault, if there is any.

Here is the formal definition of fault detection:
Device Status: Let $Z_t(x)$ be the status of a device at time t, where x represents an arbitrary input vector [e.g. rate of change (dy/dt) of power, memory, communicational signal etc. over time]
Test: $T = \{v_1, v_2, \ldots vn\}$ where $v_1, v_2, \ldots v_n$ are input vectors and $Z_t(v_i)$ represents the status of the device accordingly.
Fault Detection: T detects a fault in the device if [$(Z_t(v_i)\ Z_t + 1(v_i)) >$ predefined threshold value].

For an example, if the change of signal strength is 10% at time t and change of memory space is 25% at time $(t+1)$ and if the threshold value is 12% then since the rate of change of memory space is greater than the threshold value, so according to our approach, there should be fault.

3.2 Fault Notification

Not only to push any information but also to notify its aliveness or its fault, the device itself need to communicate with the healing manager periodically. In Gaia, Chetan et al. [7] used heart beat message mechanism only to inform the aliveness of any device. Absence of heart beat message implies the existence of fault. Thinking ahead a little bit more, we have incorporated generic message passing scheme not only to facilitate the functionality of heart beat message but also the efficacy of SOS message for helping the healing manager to be informed about the faulty device's current situation. In this scheme, each device will send any one of the following messages to the healing manager:

- *OK message*: it simply sends a packet containing "OK" string. It's nothing but a heart beat message.
- *SOS message*: After identifying any fault in its own device, ETS self-healing of that one sends SOS message, which may include some file names along with that message. An example of such type of message is: "SOS, exam3cosc060, log status". This means that the faulty device is requesting to save files named "exam3cosc060" and "log status".
- If the healing manager doesn't get such type of message for a pre-fixed threshold period of time, right away it will commence the next steps (device isolation, information distribution, alteration) assuming that the device is in fault. If the healing manager gets SOS message along with some file names, then healing manager will initiate to get the files from the device and will store those among other devices in a secured distributed manner.

3.3 Faulty Device Isolation

The isolation of faulty devices from remaining network, a grand challenge of fault tolerant as well as self-healing system, is achieved in ETS Self-healing in a very simple way. In case of MARKS, every device is mapped with another one by means of service availability in these devices. It is adequate to remove the entry of the faulty device from that mapping to ensure its isolation from the entire network.

Table 1 shows the mapping of the service # and the service provider. We have followed a standard for the service #. For an example, service # 1 means "internet service", service # 2 means "office software", service # 100 means "music software", etc. ETS Self-healing incorporates a list named serviceList by which the service can easily be identified. For an example, serviceList

Table 1: Mapping of service # and provider

Service #	Service provider			
1	$D1$	$D7$	$D9$	$D3$
2	$D19$			
3	$D12$	$D12$	$D2$	
4				
5	$D1$	$D3$	$D4$	$D9$
. . . . .	$D20$			
100	$D2$	$D9$	$D18$	

Table 2: Mapping of service #, service provider, and service consumer

Service provider	Service consumer						
	$D1$	$D2$	$D3$	. . .	$D9$	. . .	$D20$
$D1$		1			1,5		
$D2$			100		100		
$D3$	5						
$D9$		1,5					5
. . . . .							
$D20$	15		99				

(100) will return "music software". Table 2 exemplifies the three-dimensional mapping of service provider, service consumer, and service #. Here $D1$ means Device 1, $D2$ means Device 2, etc. These mappings are implemented in ETS self-healing by using hash table. By means of standard "remove" function, ETS Self-healing can remove the entry of faulty device from the hash table as well as from the entire network.

3.4 Alteration: Responsibility Reassignment in Other Devices

Since there might have some devices largely depending upon the faulty device for any specific service, it is also imperative to have some alternative to continue the smooth functionality. Some middleware like Gaia [7, 15] uses surrogate device concept for alternate solution, where it needs to find out another device that is available as well as compatible with the faulty device [7]. These two fold processes (availability and compatibility) can be performed effortlessly in ETS Self-healing since the previously mentioned hash tables, regulated and updated by healing manager, shows only those devices that are compatible as well as available.

3.5 Information Distribution

To assist the faulty device to keep all the important information safe and secured, the healing manager will distribute the important information (e.g. important database) among other existing devices. To cope with the limited storage capacity of all other devices, secret sharing (N, t) approach [18] has been used in ETS Self-healing. In Secret Sharing (N, t) approach the secret is shared among N number of devices and at least t number of devices has to be present to recover the secret. We modified this slightly and added the owner of the device among t devices. This preserves the security (make information inaccessible from any intruder or unauthorized user). In this case, without the faulty device's consent, the information of the faulty device can't be retrieved by anyone even not by the healing manager.

3.6 Fault Healing

ETS Self-healing of the faulty device itself, first of all, will try to heal the fault. Failure to do so compels itself to take help from the healing manager.

3.6.1 Healing by Device Itself

In case of typical problems like abrupt change of memory status or processor speed or signal strength, ETS Self-healing unit of the device itself simply restarts the system by calling interrupt. Before doing so, it always prompts message to the user about the problem as well as its action what it is going to do in that regard. In the evaluation section we presented screen shots of our prototype.

To treat the unusual behavior (abrupt change of memory status or processor speed or signal strength) of the system due to the use of any specific application, ETS Self-healing simply kills that application without notifying the user.

3.6.2 Healing by Healing Manager

Healing manager, in the majority of the circumstances, is not directly associated to healing, rather it assists the device to revamp its previous impeccable situations. It stores all the crucial information including logstatus file of the faulty device when the device falls in trouble. After recovering from fault, healing manager re-collets all those information and sends it to that device including logstatus file so that the device can restore easily its previous condition. In case of failure due to the lack of memory space, healing manager can help by storing its important but not currently using information among other devices using modified secret sharing approach [19]. The user of the faulty device will have to choose those file names through ETS self-healing.

3.7 Service Manager

Healing manager is also not beyond of any fault. To prevent one point failure due to the collapse of the healing manager itself, service manager [19] is used as an alternative in ETS Self-healing. It will not only act as healing manager after any fault of the current healing manager

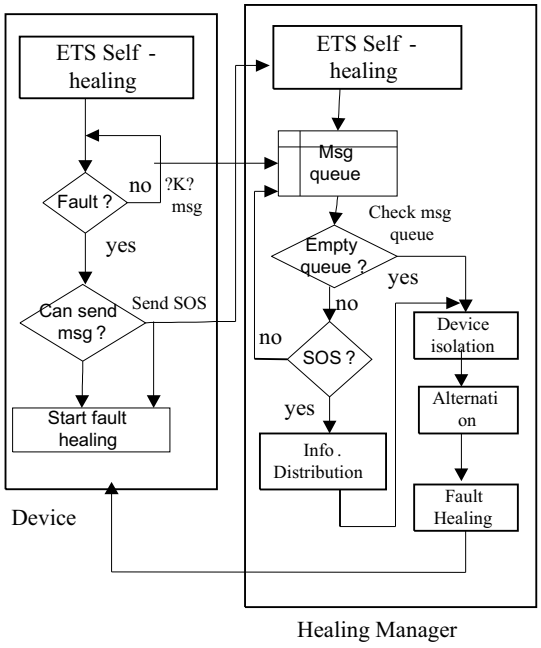


Figure 3: Flow diagram of ETS self-healing

but also help to find out the Byzantine problem [17] if there is any by consulting healing manager.

3.8 Selection of Healing Manager

It depends on different criteria like fast processing capability, large available memory space, more functionality and mostly the nature of the applications running among the devices of the ad-hoc network. In the first prototype of ETS Self-healing, a simple yet powerful approach is being used. The main communication point like a server in distributed computing is primarily selected as healing manager. To identify that central point, the status of running application has been checked. The device, which involves the applications mostly, is considered as the healing manager. For an example, in case of wireless exam application (it would be discussed shortly), the teacher's device acts like a server (it sends exam to all student's and also gets answer from them) and that is why it is chosen as the healing manager.

3.9 How Does ETS Self-healing Work?

The entire healing process is based on the combined effort of the device and the healing manager. The diagram of the message flow between a normal device and healing manager is shown in Figure 3. If a device cannot send any msg ("OK" or "SOS") for a threshold time, then the healing manager will understand that the device is in fault.

Table 3: Space for ETS self-healing service and MARKS-ORB

	Lines of Code	Size of EXE File (KB)
ETS Self-Healing Service	4974	124
MARKS-ORB	416	7

3.10 Relevant Functionalities of MARKS-ORB

Due to the ad-hoc nature of the pervasive computing environment, healing manager needs to perform two important activities:

1) Device discovery, to find which device is present within its network range.

2) Continuous wireless communication between all the devices and the healing manager.

To make the healing manager free from discovery and communication related activities, MARKS-ORB will perform those actions on behalf of it.

4 ETS Self-healing: Attributes

ETS Self-healing is based on three most crucial attributes that ought to be included in every effective self-healing mechanism. The way to achieve these features in our approach is described here.

4.1 Efficient

ETS self-healing is efficient from various standpoints:

Space: One replica of healing manager is preserved in service manager. Even then it is less space consuming since in this approach it is not needed to keep the replica of all the devices while this is a common scenario for most of other schemes. Moreover, MARKS-ORB and ETS Self-healing service occupy a very little space in the PDA. Table 3 shows the lines of code and the size of the executable file of ETS Self-healing Service and MARKS-ORB.

Healing: In most cases, the faulty device, with the help of ETS self-healing, is adequate to heal itself though in some cases the help of the healing manager might be needed.

Speed: The time complexity for different functions of ETS Self-healing service and MARKS-ORB is shown in Table 4.

According to the flow diagram of ETS Self-healing service, there is a loop for message queue inside of which four

Table 4: Time complexity of various functions of ETS self-healing service and MARKS-ORB

	Functions	Time Complexity
ETS Self-healing	Info. Distribution	$O(n)$
	Device Isolation	$O(1)$
	Alteration	$O(1)$
	Fault Healing	$O(n)$
MARKS-ORB	Device Discovery	$O(n)$
	Communication	$O(n)$

functions named Info. Distribution, Device Isolation, Alteration, and Fault Healing are running. Since the complexity of each of these functions is $O(n)$, the overall time complexity for ETS Self-healing is $O(n^2)$.

On the contrary, Device discovery and Communication, the two functions of MARKS-ORB are independent to each other and that is why the time complexity for MARKS-ORB is $O(n)$.

This low time complexity makes our approach efficient from speed perspective.

4.2 Transparent

ETS self-healing tries to assure the nominal involvement of user. In most cases, without any kind of user's interruption, the faulty device would be healed by itself or at best with the assistance of the healing manager of the ad-hoc network. ETS self-healing involves the users when decision largely depends upon the users' preference.

4.3 Security

ETS Self-healing maintains security in different stages.

Authentication: In Healing Manager and other devices:

To provide a high degree of security, a simple yet effective secret code system has been devised in ETS Self-healing. No one will be able to use the device unless it knows the secret code. In the evaluation section we present the authentication screen that we added in our prototype implementation.

Security: Regarding Information Distribution:

Modified secret sharing approach, in the very first prototype of ETS Self-healing has been implemented in a very simple fashion. Random number procedure has been chosen to generate the key that is XOR-ed with all the information that should be distributed among N devices. Only t (t¡=N) devices are needed to extract that actual information. After revival of faulty device, its participation is needed to extract that actual information. This approach promises the security that no device will be able to abuse the information of the faulty device.

Privacy: Responsibility Re-assignment:

During updating the hash table regarding service re-assignment, the consent of the service provider, a member of that ad-hoc network, is needed which ensures user's privacy as well as security.

5 Current State of the Art

The issues of self-healing are addressed from different standpoints. Researchers are working on several policies like architecture based System [8, 10], infrastructure based approach [3] for a long time. Most of these models are suitable for physically connected computers in distributed computing environment. There is no established method that provides solution for devices running in pervasive computing environment where it is assumed that the devices are connected to each other wirelessly.

Soila and Priya presented proactive recovery in Distributed CORBA [14] applications. They did not concentrate on fault-prediction technique; rather they focused on the exploitation of fault prediction in systems that had real time deadlines. Our system deals with pervasive computing and it can predict the fault by calculating different states of the system.

To handle transient software failures, a proactive approach named software rejuvenation, was proposed by Huang et al [12]. According to this approach, if errors are accumulated beyond a threshold, then it will kill and re-launch the application. A lot of works about rejuvenation policies, to increase system availability and to reduce the cost of rejuvenation, were done by [5, 9, 23]. However, to hand-off the existing state of the faulty device just after its re-launching was overlooked here. Our approach preserves this state among other devices in a secured manner (Secret Sharing) so that the healing manager can help the faulty device to get its actual state after healing.

Garlan and Schmerl presented a system [10] that uses architectural model for monitoring, problem detection, and repair. But their main focus is on distributed systems with high performance requirement.

Eric et al. [8] describes a system based on software architecture that uses software components and connectors for repair. This system is also targeted for distributed systems. They use infrastructure support for repair purposes.

Gordon et al. [4] in their paper presented an analysis of the role of "Reflection" to support self-healing systems. They also suggested that middleware would be the appropriate place for including self-healing unit. They offered their primary analysis based on distributed systems. But they did not implement the middleware and also not the self-adaptive, self-healing unit.

AMUN [21] is a middleware that deals with self-healing for ubiquitous environment but they concentrate only in indoor environment like inside an office building. They

use "Smart Doorplates" that use and display situational information of the owner of the office. This idea restricts its use only in a smart environment.

L. Kant [13] proposed a self-healing mechanism for wireless network. He claimed that this mechanism can provide seamless restoration of affected services due to random/sporadic network facility failures. But this approach is not suitable for pervasive computing since the ad-hoc nature is totally overlooked here.

6 ETS Self-healing: Prototype Implementation and Evaluation

The first prototype of ETS self-healing, of late, to uphold the design, has been developed and integrated along with our current developing middleware named MARKS [20]. WinCE running on a set of Dell Axim X30 pocket PCs (Process type is Intel@PXA270 and speed is 624 MHz), to demonstrate our approach, are used as platform. .NET Compact framework along with C# is used as implementation language. Bluetooth, as the underlying wireless protocol, has been used though it is also suitable for IEEE 802.11.

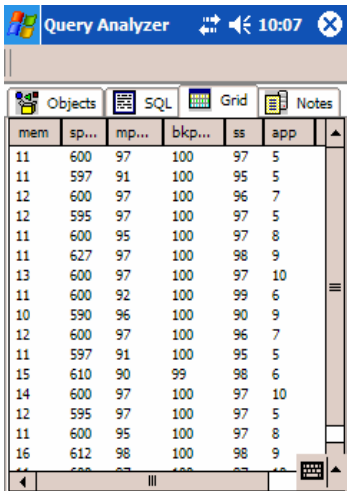
Socket and thread programming have been used for successful communication among all the devices in the pervasive computing. In healing manager, one thread is used for each device. The thread number is also dynamic due to the ad-hoc nature of the network. SQLCE database, to store the information, is also used.

To evaluate the performance of ETS Self-healing, several applications have been developed. Wireless exam is such an application by which one teacher can send questions to the students (PDA to PDA communication) and also can collect the answers from the students. There are some selected screen shots captured from the implemented prototype below.

Figure 4 presents a log file stored in an embedded device (a pocket pc exploited in the application which used the first prototype of ETS self-healing). Figure 5 illustrates the nature of rate of change of used memory space over time through which ETS self-healing can determine the possibility of a problem between time stamp 14 and 15 due to the sharp change of rate of used memory space.

Figure 6 presents the status of the battery power for five devices where the prototype our ETS Self-healing and MARKS-ORB are running. The sharp change of the status of the battery power for D4 indicates that there is some problem in that device and needs healing immediately.

By using the "status changing rate" process, ETS self-healing of the student's device itself tries to find out the fault as well as the reason if there is any. Figure 7 shows a typical message generated by the devices' healing unit. This message is intended to inform the user about the abrupt change of device status and action taken by the ETS Self-healing.



mem	sp...	mp...	bkp...	ss	app
11	600	97	100	97	5
11	597	91	100	95	5
12	600	97	100	96	7
12	595	97	100	97	5
11	600	95	100	97	8
11	627	97	100	98	9
13	600	97	100	97	10
11	600	92	100	99	6
10	590	96	100	90	9
12	600	97	100	96	7
11	597	91	100	95	5
15	610	90	99	98	6
14	600	97	100	97	10
12	595	97	100	97	5
11	600	95	100	97	8
16	612	98	100	98	9

Figure 4: Status of a device

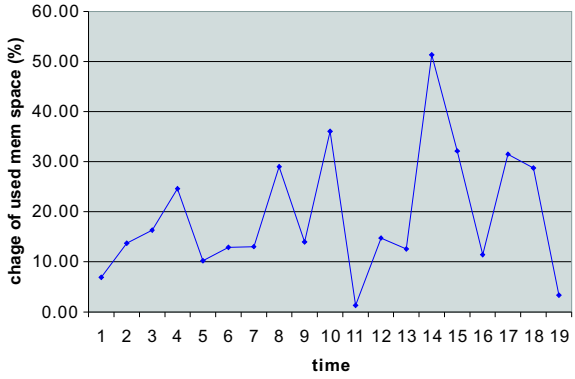


Figure 5: Rate of change of used memory space

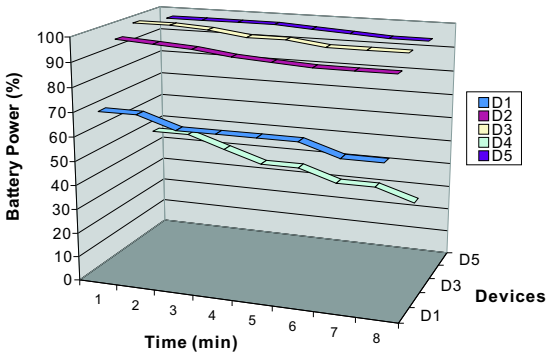


Figure 6: Status of the battery power of five devices

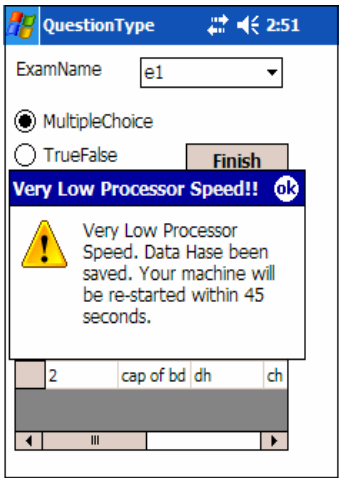


Figure 7: Message to the user regarding low processor speed

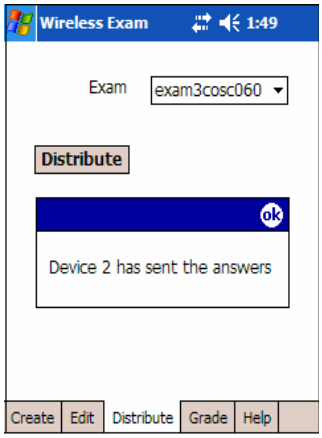


Figure 8: Device 2 is running without any problem

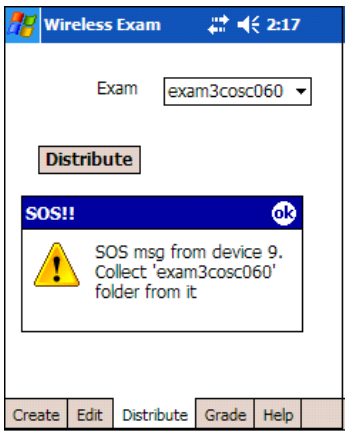


Figure 9: Device 9 sends SOS message

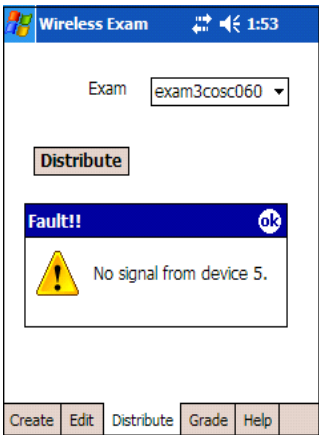


Figure 10: No signal from device 5

There are situations where all the devices operate without any error. Then only the "OK" message is sent to the healing manager periodically. Let device 2 has no problem and it periodically sends OK message to teacher, the healing manager of this network. Within a specified period of time, it also sends all the answers to the teacher's PDA. This scenario is shown in Figure 8.

Now device 9 finds some problem. It simply sends SOS message including file name exam3cosc060. Without any delay, the healing manager will collect that file from this device. Figure 9 portrays this event.

Another case can occur where the device is unable to send any message due to fault. If the healing manager does not get any message for a long period of time from any device, it will take appropriate action assuming that the device is in fault. Device 5 is unable to send any message for a long time. So, the healing manager takes rapid action regarding device 5. This incident is illustrated in Figure 10.

As both device 5 and 9 are faulty now, healing manager removes the entry of device 5 and 9 from hash table. It also updates the table to reassign the services.

Along with the SOS message, device 9 sends the list of important file names (selected by the user of that device) that need to be saved. This is shown in Figure 11. Device 5 and 9 will try to be healed without the help of others. After healing, healing manager will resend the files that it got before their fault.

Simple yet powerful authentication scheme is provided in our first prototype. A random combination of 7 digits is used as the secret code to operate the ETS self-healing unit. To break this authentication system, even if one full secret code is entered within 1 second, it will take approximately 9 years. Figure 12 depicts the user authentication screen.

We have also developed the MARKS-ORB, to provide the device discovery and communication functionality of the devices. Figure 13 shows the battery power consumption with respect to time while MARKS-ORB is running in that device. Here S1 indicates the battery power

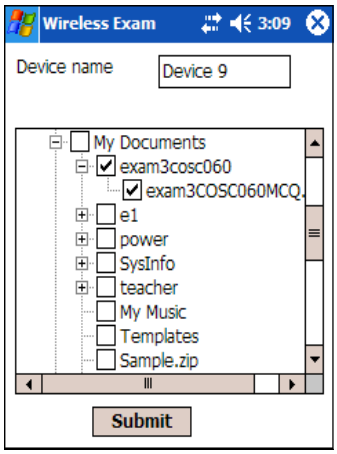


Figure 11: Important files name selected by user to be saved

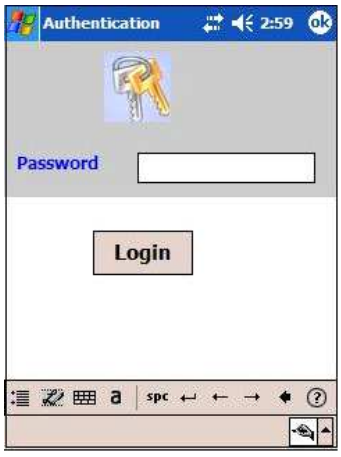


Figure 12: Authentication of the user

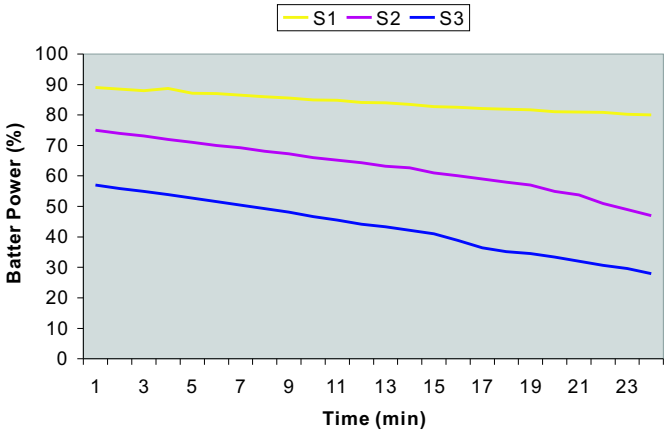


Figure 13: Time (min) vs. battery power (%) for MARKS-ORB

consumption while the Pocket PC is on but the wireless mode is off (no wireless communication via 802.11 or Bluetooth). S2 means that wireless mode is on. S3 indicates that wireless mode is on and MARKS-ORB is running in the device. It clearly indicates that MARKS-ORB consumes a very little amount of battery power.

MARKS-ORB itself transfers data mainly for device discovery. It broadcasts its own IP address and receives the IP addresses of other devices reside in the same ad-hoc network. Figure 14 shows the data transmission by MARKS-ORB in every 5 seconds. It clearly shows that it does not need to transmit so much data for device discovery.

7 Conclusion and Future Direction

In this paper, we have described the design as well as the implementation of a ETS Self-healing service, a fundamental part of MARKS, a dependable middleware designed for devices running in pervasive computing environments. Fault detection and healing are covered in this unit. It promises the least possible time not only to detect the fault but also to restore its smooth functionality. It also promises the lowest degree of user intervention and hence the highest level of transparency. The powerful authentication scheme augments its security features.

In future, we will develop a generic approach to handle different kinds of healing (system failure, link failure, Byzantine problem, etc.). Moreover, hard healing (recovery process for special kind of system failure) problem will also be taken care of. For an example, to heal a faulty device running out of battery power, power would be collected from other devices of the same ad-hoc network and supplied to the faulty device in packet format. The suitability of self-healing protocol will be evaluated using simulations and other more generic applications. To develop

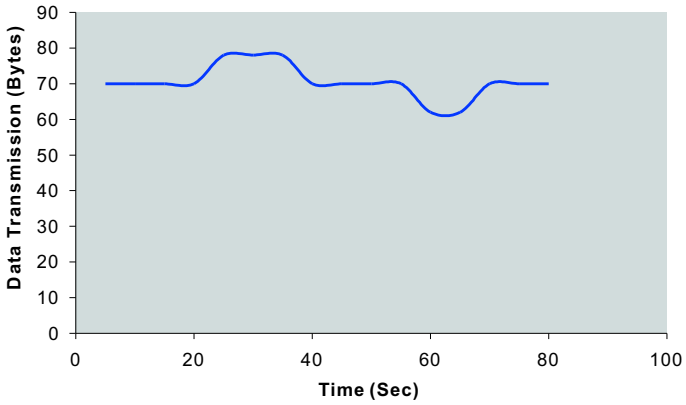


Figure 14: Time (sec) vs. data transmission (Bytes) for MARKS-ORB

a really effective benchmark to select healing manager is another goal of near future.

References

[1] S. I. Ahamed, M. Sharmin, S. Ahmed, M. J. Havice, and S. Anamanamuri, "An assessment tool for out of class learning using pervasive computing technologies," *Journal of Information*, vol. 8, no. 5, pp. 751-768, Sept. 2005.

[2] S. Ahmed, M. Sharmin, and S. I. Ahamed, "Knowledge usability and its characteristics for pervasive computing environments," in *Proceedings of the 2005 International Conference on Pervasive Systems and Computing (PSC-05) in conjunction with The 2005 International Multi-conference in Computer Science and Engineering*, pp. 206-209, Las Vegas, NV, USA. Jun. 27-30, 2005.

[3] K. J. Appavoo, M. S. Hui, R. W. Wisniewski, D. D. Silva, O. Krieger, and C. A. N. Soules, "An infrastructure for multiprocessor run-time adaptation," in *Proceedings of the first workshop on Self-healing systems*, pp. 3-8, Charleston, South Carolina, 2002.

[4] S. Blair, G. Coulson, L. Blair, H. D. Limon, P. Grace, and R. M. N. Parlavantzas, "Reflection, self-awareness and self-healing in OpenORB," in *Proceedings of the first workshop on Self-healing systems*, pp. 9-14, Charleston, South Carolina, 2002.

[5] A. Bobbio and M. Sereno, "Fine grained software rejuvenation models," in *Computer Performance and Dependability Symposium (IPDS 98)*, pp. 4-12, Sept. 1998.

[6] T. D. Bracewell and P. Narasimhan, "A middleware for dependable distributed real-time systems," in *Joint Systems and Software Engineering Symposium*, Falls Church, VA, Apr. 2003. URL:http://www.ece.cmu.edu/ mead/raytheonSymp-2003.pdf (accessed in May 2006).

[7] S. Chetan, A. Ranganathan, and R. Campbell, "Towards fault tolerant pervasive computing," in *Pervasive 2004 Workshop on Sustainable Pervasive Computing*, pp. 38-44, Linz/Vienna, Austria, Apr. 2004.

[8] E. M. Dashofy, A. V. D. Hoek, and R. N. Taylor, "Towards architecture-based self-healing systems," in *Proceedings of the first workshop on Self-healing systems*, pp. 21-26, Charleston, South Carolina, 2002.

[9] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. Trivedi, "A methodology for detection and estimation of software aging," in *International Symposium on Software Reliability Engineering*, pp. 283-292, Nov. 1998.

[10] D. Garlan, and B. Schmerl, "Model-based adaptation for self-healing systems," in *Proceedings of the first workshop on Self-healing systems*, pp. 27- 32, Charleston, South Carolina, Nov. 18-19, 2002.

[11] D. Garlan, V. Poladian, B. Schmerl, and J. P. Sousa, "Task-based self-adaptation," in *Proceedings of the ACM SIGSOFT 2004 Workshop on Self-Managing Systems (WOSS'04)*, pp. 54-57, Newport Beach, CA, Oct-Nov 2004.

[12] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software rejuvenation: Analysis, module and applications," in *International Symposium on Fault-Tolerant Computing*, pp. 381-390, Pasadena, CA, Jun. 27-30, 1995.

[13] L. Kant, "Design and performance modeling & simulation of self-healing mechanisms for wireless communication networks," in *Proceedings of the 35th Annual Simulation Symposium*, pp. 35-42, Apr. 2002.

[14] S. Pertet and P. Narasimhan, "Proactive recovery in distributed CORBA applications," in *IEEE Conference on Dependable Systems and Networks (DSN)*, pp. 357-366, Florence, Italy, Jun. 2004.

[15] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "Gaia: A middleware infrastructure for active spaces," *IEEE Pervasive Computing*, pp. 74-83, Oct-Dec 2002.

[16] M. Satyanarayanan, "Fundamental challenges in mobile computing," in *ACM Symposium on Principles of Distributed Computing, (PODC'96 invited lecture)*, pp. 1-7, 1996.

[17] F. B. Schneider, "Byzantine generals in action: Implementing fail-stop processors," *ACM Transactions on Computer Systems*, vol. 2, no. 2, pp. 145-154, May 1984.

[18] Secret Sharing, URL: www.cmpe.boun.edu.tr/ courses/cmpe471/spring2003/download/cmpe47109-2003.ppt (accessed in May 2005).

[19] M. Sharmin, S. Ahmed, and S. I. Ahamed, "SAFE-RD (Secure, adaptive, fault tolerant, and efficient resource discovery) in pervasive computing environments," in *Proceedings of the IEEE international Conference on Information Technology (ITCC 2005)*, pp. 271-276, Las Vegas, NV, USA, Apr. 4-6, 2005.

[20] M. Sharmin, S. Ahmed, and S. I. Ahamed, "MARKS (Middleware adaptability for resource discovery, knowledge usability and self-healing) for mobile devices of pervasive computing environments," To appear in *Third International Conference on Information Technology: New Generations (ITNG 2006)*, Apr. 2006, Las Vegas, NV, USA.

[21] W. Trumler, J. Petzold, F. Bagci, and T. Ungerer, "AMUN - An autonomic middleware for the smart doorplate project," in *System Support for Ubiquitous Computing Workshop at the Sixth Annual Conference on Ubiquitous Computing (UbiComp 2004)*, pp. 274-275, Notthingham, England, Sep. 7, 2004.

[22] M. Weiser, "Some computer science problems in ubiquitous computing," *Communications of the ACM*, vol. 36, no. 7, pp. 75-84, Jul. 1993.

[23] B. Yujuan, S. Xiaobai, and K. S. Trivedi, "Adaptive software rejuvenation: Degradation model and rejuvenation scheme," in *International Conference on Dependable Systems and Networks*, pp. 241-248, Jun. 2003.

Shameem Ahmed is a graduate student in the department of Math., Stat., and Computer Science at Marquette University, USA. Ahmed received his B.Sc. in computer science and engineering from the Bangladesh University of Engineering and Technology, Bangladesh in 2003. His research interests are user modeling in ubiquitous/pervasive computing, location-aware computing, and human computer interface. He can be contacted at shameem.ahmed@mu.edu; http://www.shameemahmed.com

Moushumi Sharmin is a graduate student in the department of Math., Stat., and Computer Science at Marquette University, USA. Sharmin received his B.Sc. in computer science and engineering from the Bangladesh University of Engineering and Technology, Bangladesh in 2003. Her research interests are resource discovery in ad-hoc networks, sensor networks, and middleware for ubiquitous/pervasive computing. She can be contacted at moushumi.sharmin@mu.edu; http://www.mscs.mu.edu/ msharmin.

Sheikh I. Ahamed is an assistant professor in the department of Math., Stat., and Computer Science at Marquette University, USA. He is a member of the IEEE, ACM, and the IEEE Computer Society. Dr. Ahamed received the B.Sc. in computer science and engineering from the Bangladesh University of Engineering and Technology, Bangladesh in 1995. He completed his Ph.D in Computer Science from Arizona State University, USA in 2003. His research interests are security in ad hoc networks, middleware for ubiquitous/pervasive computing, sensor networks, and component-based software development. He serves regularly on international conference program committees in software engineering and pervasive computing such as COMPSAC 04, COMPSAC 05, COMPSAC 06, and ITCC 05. He is the Workshop Program Co-Chair of International Workshop on Security, Privacy, and Trust for Pervasive Computing (SPTPA 06). He also directs the Ubicomp research lab (www.mscs.mu.edu/ ubicomp) in the the department of Math., Stat., and Computer Science at Marquette University, USA.. Dr. Ahamed can be contacted at iq@mscs.mu.edu; http://www.mscs.mu.edu/ iq.