A Robust and Adaptive Ambient Services Management Scheme for Smart Homes

*1Chun-Feng Liao, 2Ya-Wen Jong, 2Li-Chen Fu

*1*Department of Computer Science, National Chengchi University, Taipei, Taiwan,*
**cfliao@nccu.edu.tw*
2*Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan,*
{r96922022, lichen}@ntu.edu.tw

Abstract

A considerable number of studies have been made on ambient service management platform for smart home environments. Few of them addressed the robustness issue, which is fundamental for such environments. Robustness is critical because there are usually services that are critical to the habitants health and life in Smart Homes. In this paper, we propose an efficient and adaptive failure detection and recovery protocol for an ambient service management platform, in which failures are detected efficiently with a novel rotating roll-call algorithm without a centralized monitor. Adaptive techniques are also proposed to maintain network stability. The analysis and experiment results show that the proposed solution is capable of detecting software failures efficiently without causing network instability, hence preserving robustness of an ambient service platform at home.

***Keywords**: Smart Home, Service Management, UPnP, Service Discovery, Service Management Platform*

1. Introduction

The Smart Home Environment is considered one of the most promising next generation computing environments because of the rapid emerging of intelligent devices and sensors. Typically, these intelligent devices and sensors cooperatively provide sophisticated services to the user based on an ambient service management platform. Among design issues of an ambient service management platform in a Smart Home, robustness is a paramount concern of inhabitants [10, 12].

The reasons that services at Smart Homes have to be robust is twofold. a) In a laboratory environment, there are likely to be enough technologically educated people to administrate computer systems and to fix faulty hardware and software right on the spot [10]. On the other hand, in most of the homes, this kind of people do not exist. In other words, the occupants of a Smart Home are usually non-technical users, that is, the people setting up and maintaining the home services are everyday consumers, with little or no knowledge of technologies. b) When staying at home, habitants expect that their appliances will not crash [10], most of the domestic technologies are expected to work 24-7.

As the consumers are unable to pinpoint the source of failures [9], the ambient service management platform must be able to detect and to recover from failures autonomously. Also, the detection and recovery procedures have to be carried out in a minimal amount of time. Otherwise, the failures can lead to a very frustrating user experience and bad marketing perception for the vendors of home services. Consequently, a robust ambient service management platform in the Smart Home has to be self-diagnosable, self-recoverable, and as efficient as possible. Many ambient service management

* Corresponding Author

Received: Nov. 16, 2016, Revised Dec. 22, 2016, Accepted: Dec. 28, 2016

platforms were proposed in the last few years e.g. UPnP (Universal Plug and Play) [25], Jini [2] and SLP (Service Location Protocol) [14]. Unfortunately, few of them address the robustness issue.

The robustness of an ambient service management platform is largely affected by the underlying architectural styles, which fall into two categories. In the process-centric style, the logic of flow is controlled by a process, which searches services in a centralized service registry and invokes services in a synchronized way. The classical Context ToolKits [21] belong to this type. Although these works address important aspects of ambient service management platform, they are criticized for their synchronous and centralized nature [26, 20]. On the other hand, more loosely coupled and asynchronous architectural styles such as Tuple-space (TS) [11] and Message-Oriented Middleware (MOM) [3] are proposed to avoid the problems of process-centric systems. A TS server is essentially an associative virtual shared memory storing serialized objects. Distributed clients can read, write or take serialized objects from TS server. TS tends to be inefficient as they are implemented using a more database-like approach of using locks with read/writes to entries. Moreover, TS tends to store serialized objects, which is usually a penalty on performance and causing the TS server to become a bottleneck. Besides, a TS based solution is often language dependent. On these grounds, MOM is a more promising architecture for ambient service systems. For example, MQTT (MQ Telemetry Transport) is a popular MOM-based connectivity protocol standard for Internet of Things (IoT) in recent years [4]. The loosely-coupled nature of MOM brings about the problems of robustness. Components are so decoupled that it is hard to be aware of the status among one another. To address this issue, we proposed a message-oriented service model for Smart Homes called Pervasive Service Application Model (PerSAM), and a robust service management protocol called Pervasive Service Management Protocol(PSMP) [19]. In PerSAM, the Manager Nodes are assumed never fail and are responsible for detecting the failures of Worker Nodes and then recover the failed node if there is one. We found that the hierarchical and centralized service management architecture is more efficient than consensus-based ones. In real-world scenarios, it is nearly impossible that the Manager Nodes never fail.

In this paper, a consensus-based and adaptive failure detection and recovery protocol is proposed and is integrated into PerSAM/PSMP. In particular, we enhance PerSAM/PSMP by proposing a novel collaborative failure detection and recovery protocol for PerSAM, called Roll-call Protocol for Manager nodes(RPM), to detect and to recover from failures of Manager Nodes efficiently. This approach is inspired by the roll-call procedure. Specifically, the names of people from a list are called one after one to determine the presence or absence of the listed people. In this protocol, every node takes a turn to initiate a roll-call, and therefore a failed node can be detected and be marked as suspected. To avoid making false detections, we consider a semi-consensus policy here, that is, a node is considered as failed when more than half of the nodes in the environment suspect it. Finally, the period for calling the roll is adaptive, that is, it varies depending on the context of the network and status of the nodes. Based on the proposed protocol, the enhanced PerSAM/PSMP employs a centralized approach for Worker Node and a consensus-based approach for Manager Nodes.

The insight of such design is that consensus-based approaches are usually more robust but less efficient and centralized approaches are more efficient but suffer from single-point-of-failure. Using consensus-based approaches to managing Manager Nodes and centralized approach to managing Worker Nodes are cost effective since the number of Manager Nodes is much smaller than that of Worker Nodes.

In the following sections, we will begin by explaining preliminaries of this work. After that, the proposed approach is presented. Then, we describe the analytical and experimental results of this work. Finally, related works, conclusions, and suggestions are presented for further research.

2. Service Model

Before entering into detailed discussions of the proposed approach, this section briefly introduces the message-oriented service model and management protocol for ambient services, namely, PerSAM. The underlying communication infrastructure of PerSAM is a Message-Oriented Middleware (MOM). The MOM creates a "software bus" for integrating heterogeneous entities, namely, the "nodes." The logical messaging pathways among nodes are called "topics." A messaging endpoint is called a "PerNode." We can divide PerNodes into two categories, namely, the Manager Nodes and the Worker

Nodes, according to their responsibilities. Manager Node is designed for administrative purposes. We can further classify Worker Nodes into three sub-categories: Sensor Nodes, Actuator Nodes, and Logic Nodes, whose names imply the functions of nodes.

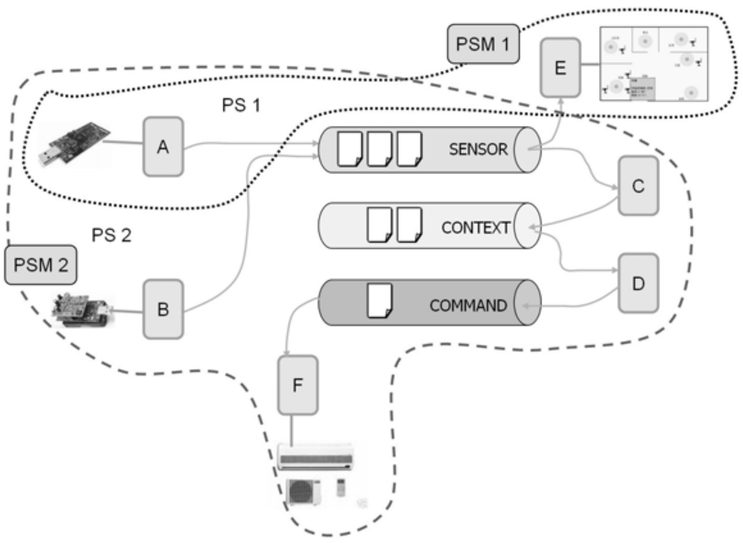


Figure 1. A typical PerSAM-based service system.

In PerSAM, a PerNode $p \in P$ is a basic logical software entity in a system, where P is the universe of PerNodes. Note that we will use "PerNode" and "node" interchangeably in the following. PerNode has two subtypes: Worker Node and Manager Node. A Worker Node $w \in W$ is a node that encapsulates a unit of application logic, where W is the universe of Worker Nodes. We can further classify Worker Nodes into 3 sub-categories according to their capabilities: Sensor Node, Logic Node, and Actuator Node. Taking Fig. 1 as an example, A and B are Sensor Nodes, which are connected to sensors and always send sensed contexts to a "SENSOR" topic. Likewise, C and D are Logic Nodes that contain special logics for processing input messages. E and F are Actuator Nodes that are responsible for presenting a graphical view of sensor status and controlling air-pumps, respectively.

Taking Fig. 1 as an example, A and B are Sensor Nodes, which are connected to sensors and always send sensed contexts to a "SENSOR" topic. Likewise, C and D are Logic Nodes that contain special logics for processing input messages. E and F are Actuator Nodes that are responsible for presenting a graphical view of sensor status and controlling air-pumps, respectively. On the other hand, a Manager Node $m \in M$ performs node administrative tasks (e.g. managing the life-cycles and keeping track of status of the affiliated Worker Nodes) instead of providing service to the user, where M is the universe of Manager Nodes in the system.

A Pervasive Service $s \in S$ consists of one Pervasive Service Manager (PSM) as well as one or more Worker Nodes that collectively provide a service to users, where S is the universe of Pervasive Services. The PSM is responsible for composing, activating, and monitoring the corresponding Pervasive Service. All members of a Pervasive Service, but not PSM, are dynamic, and each PSM is responsible for selecting most appropriate Worker Nodes during the process of service composition. A Pervasive Service $s \in S$ can be formally denoted as a tuple:

$$s = \left\langle m^s, W^s \right\rangle \in M \times 2^W ,$$

where m^s is the PSM of s, M is the universe of PSMs, and W^s is the set of Worker Nodes belonging to s. Note that a Worker Node can join several PSs at the same time. Back in Fig. 1, A is a member of PS (ps1) called "Adaptive Air Conditioner" and is also a member of another PS (ps2) named "Sensor Map" at the same time.

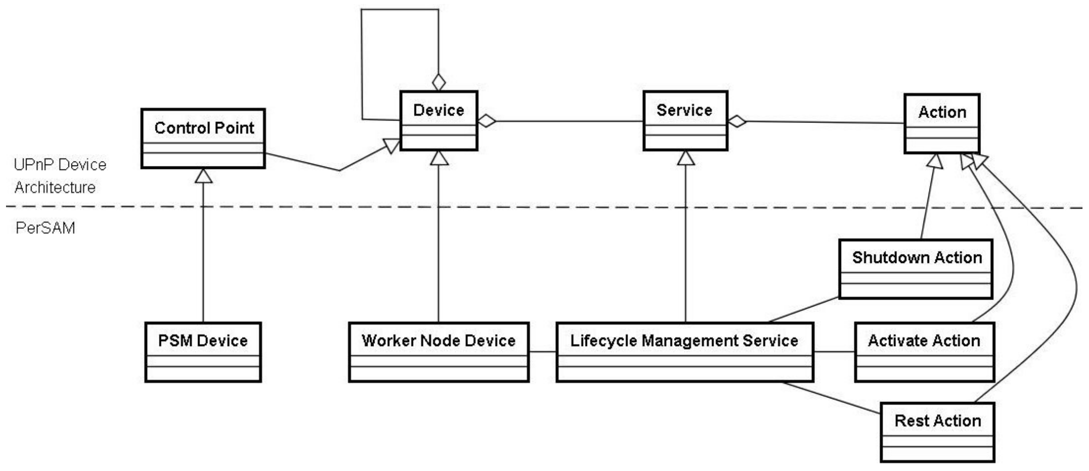


Figure 2. The Projection of PerSAM to UPnP Device Architecture.

In PerSAM, presence management, service discovery and composition tasks are realized based on UPnP since it is one of the few dynamic service discovery protocols that do not need a dedicated and centralized service registry [27]. In addition, UPnP is also platform and language independent, making the proposed protocol easier to be integrated with other platforms and devices. Among sub-protocols of UPnP, SSDP takes charge of service discovery in a UPnP network. By default, SSDP operates based on HTTP-MU (HTTP over UDP multicast), where multicast is an IP-layer mechanism of forwarding IP datagrams to a group of interested receivers via a set of predefined addresses. Since multicast is supported by most network switching equipments, SSDP does not need a central service registry. Generally speaking, SSDP extends HTTP with two message types: Notify and M-Search, resulting in three kinds of SSDP primitive actions: 1) ssdp:alive: announces the presence of a device by using a HTTP Notify message, 2) ssdp:byebye: announces that a device has left the network by using a HTTP Notify message, and 3) ssdp:discover: finds a device that meets certain type specified in the ST (Search Target) header in an M-Search message. The matched devices then reply by sending back HTTP Response messages. In PerSAM, ssdp:alive and ssdp:byebye are used to realize presence management. Meanwhile, service discovery and composition are carried out based on ssdp:discover.

Fig. 2 depicts the structure of PerSAM and its projection to UPnP Device Architecture. Each PerNode-based UPnP Device consists of a UPnP Service, namely, the PerNode Life-cycle Management Service, which manages PerNode life-cycle according to the three UPnP Actions (Activate, Rest, and Shutdown). The PSM Device is a special type of UPnP Device because it has a Control Point. The reason for this design is that a Control Point is capable of invoking UPnP Actions of remote PerNodes to manage their life-cycles. It is worthy to emphasize that despite the similarity in their names, the UPnP Services are different from Pervasive Services: A UPnP Service always resides in an UPnP Device, whereas a Pervasive Service is a virtual group that consists of a set of PerNodes.

3. Roll-call Protocol for Manager nodes (RPM)

As mentioned, the main objective of this research is to purpose a consensus-based ambient service management mechanism, which can be combined with PerSAM/PSMP to form a more robust hierarchical service management architecture. In particular, as the number of Worker Nodes is typically greater than that of Manage Nodes, consensus-based, which is less efficient but more robust, is more suitable for Manager Nodes. Therefore, this paper purposes a new consensus-based service management approach for Manager Nodes, which is able to enhance PerSAM so that the "persistent Manager Node assumption" is this service model can be removed.

In the following, we describe the above-mentioned new consensus-based approach, called Roll-call Protocol for Manager Nodes(RPM) and auxiliary mechanisms for facilitating adaptive and robust ambient service management among Management Nodes. As RPM is designed specifically for

Manager Nodes, to improve the readability, we use the word "node" instead of "Manager Node" when presenting RPM in this section.

3.1. Detecting Failed Nodes

The design of RPM is inspired by the roll-call procedure in daily lives. The presence management tasks are done by UPnPs SSDP, in which every node keeps tracks of the presence information of all other nodes. When an ssdp:alive is received, the metadata of the node that declares its presence is automatically stored by other nodes locally. In other words, each node in the system maintains presence information of other nodes locally.

To begin an RPM process, all nodes perform an Invitation-based Leader Election procedure to decide the first leader. The key idea is that a node wishing to become the leader "invites" other node to join it in forming a group. Then, each node is the leader of its own group which contains itself only. Each leader periodically "invites" other groups to join it. To avoid the Livelock condition, a priority mechanism is used so that lower priority nodes send out invitation after a longer period. After a leader is elected, it begins to call the roll.

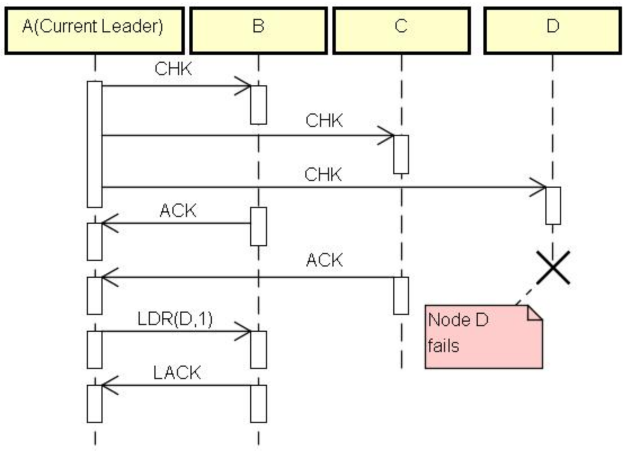


Figure 3. Node A is the current leader. As node D does not reply, it is added to the suspected list. This list is passed along with the leader authority to the next leader, node B.

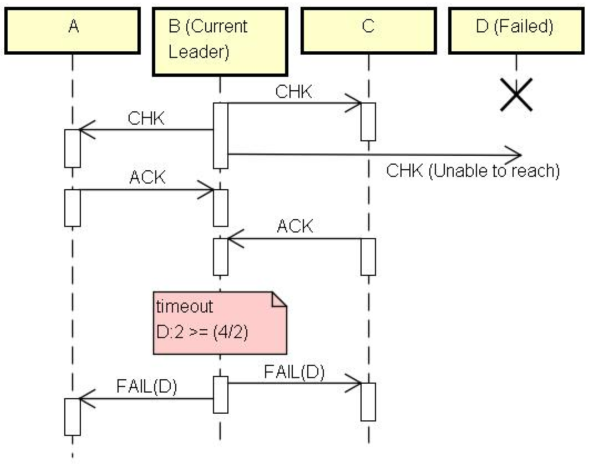


Figure 4. Node B is the current leader. As node D does not reply, the number of nodes that suspect D is incremented. Now half of the nodes in the environment suspect node D, so the leader announces its failure

To start a roll call, the leader sends out a Check (CHK) message, which is received by every node, just like calling a roll. A node that receives this CHK message will wait for a random period and reply with an Acknowledgment (ACK) message. The leader will wait for a specific time to collect the ACK message of every node. If a node doesn’t send back an ACK after the timeout, the leader will add that node to the Suspected List with an integer that represents the number of nodes that suspect it. If the suspected node is not in the list, then the integer is set to one. On the other hand, if the suspected node is already in the list, then the integer that corresponds to the suspected node is incremented by one. This Suspected List is passed along with the leader authority. So, after a leader finish calling the roll, it will select the next node to become the next leader. The current leader will send a Leader (LDR) message with the Suspected List to the next leader. As shown in Fig. 3. The node that receives the LDR message must reply with a LACK message indicating that it has received the leader authority. This LACK message is necessary as the leader authority must not be lost. The reason for rotating the leader authority is to balance the load of each node, see Fig. 4.

It should be noticed that each node in the environment has its task: the process of calling the roll is a burden that they carry out to look out for each other. Every leader, after finishing calling the roll, will verify its Suspected List. If there is a node in the Suspected List with more than half of the nodes in the environment suspecting it, then that node is considered as failed. It must he noted that when counting half of the nodes, the leader needs to consider only the "healthy" nodes; nodes that are already in the Suspected List are not taken into account. The leader is in charge of announcing the failure of the node with a FAIL message. Every node that receives this FAIL message must reply with a FACK message. The process described above is rotated until none of the nodes received a CHK or an LDR message for a period. This may indicate that the node that has the leader authority has failed, and hence the Leader Election procedure is performed again.

To minimize the message count, we use both multicast and unicast for sending RPM messages. All nodes will form another multicast group, different from the original SSDP multicast group. This is to avoid Service Nodes to receive the RPM packets which are useless to them. Therefore the CHK and the FAIL messages of RPM are sent using multicast to the nodes because the CHK and FAIL messages are for nodes. The ACK, LDR, LACK, and FACK messages of RPM are sent using unicast; this is because these messages are only for the leader. If LACK and FACK are not received after an LDR or a FAIL is sent, the leader will add the not responding node to the Suspected List and choose the next leader to pass the leader authority.

3.2. The Failure Detection Algorithm in RPM

The list of protocol elements used in the RPM is shown in Table 1 and the failure detecting algorithm for RPM is shown in Fig. 5.

When a node is started, the variables are initialized. The Alive List is for storing the nodes that are operating properly, whereas the Suspected List is actually a map that lists the suspected nodes and the numbers of nodes that suspect them, respectively. Next, the leader queries for the nodes that are already in the environment (Fig. 5, line 7-10). This is done using the SSDP, and we denote the set of nodes in the environment as π. The event < *flp2pSend* | *dest, m*> is used to request the transmission of message *m* to process *dest* through a fair-loss point to point link. The event < *flp2pMulticastSend* | *m*> is used to request the transmission of message m to the multicast group through a fair-loss point to point link. The event <*flp2pReceive* | *src, m*> is used to receive message *m* sent by process *src* through a fair-loss point to point link.

When a node receives an LDR with the current suspected list Γ, it will send back a LACK immediately and the process of calling the role begins. This node sends out a CHK to all nodes and starts the timer (see Fig. 5, line 12-17). The nodes that reply with an ACK to the leader are added to the Alive List (Fig. 5, line 27-28). When the timer is due, the < *Timeout* > event is triggered, and the leader checks if there is a node that didn’t reply yet, indicating that such node is not in the Alive List (Fig. 5, line 20-21). If yes, then the number of nodes that suspect it is incremented by one in the Suspected List (Fig. 5, line 22). Next, the leader checks if the number of nodes that suspect the above node is more than half of a number of nodes in the environment. If yes again, then the leader will announce the failure with a FAIL (Fig. 5, line 23-24). Finally, the current leader will choose the next leader and pass the leader authority to it with an LDR (Fig. 5, line 25). The actions in PRM are summarized in Table 1.

```
Implements:
   Roll-Call-based Failure Detector

Uses:
   FairLossPointToPointLinks (flp2p)

upon event < Init > do
   trigger < SSDP | $\Pi$ >;
   alive := $\emptyset$;
   $\forall p_i \in \Pi$ : suspected [ $p_i$ ] := 0;

upon event < flp2pReceive | src, [LDR . $\Gamma$] > do
   trigger < flp2pSend | src, [LACK] >;
   alive := $\emptyset$;
   suspected := $\Gamma$;
   trigger < flp2pMulticastSend | [CHK] >;
   startTimer (TimeoutDelay);

upon event < Timeout > do
   forall $p_i \in \Pi$ do
      if $p_i \notin$ alive then
         suspected [ $p_i$ ] := suspected [ $p_i$ ] + 1;
         if suspected [ $p_i$ ] $\geq$ ( $|\Pi|$ / 2 ) then
            trigger < flp2pMulticastSend | [FAIL . $p_i$] >;
   trigger < flp2pSend | NextLeader, [LDR . suspected] >;

upon event < flp2pReceive | src, [ACK] > do
   alive := alive $\cup$ { src };
```

Figure 5. Algorithm of failure detection in RPM.

Table 1. RPM Actions.

Action	Function	Description
CHK	Check	Sent by the leader when checking every nodes status
ACK	Check Acknowledgment	Sent to the leader to indicate that the node is operating properly
LDR	Leader Assignment	Used to pass the leader authority between the nodes
LACK	Leader Assignment Acknowledgment	Indicates that the leader authority is received
FAIL	Failure	Sent by the leader to announce the failure of a node
FACK	Failure Acknowledgment	Indicates that the announcement of failure is received

3.3. Adaptive Timeout

The timeout mechanism used by the leader is initialized with the timeout delay chosen to be large enough such that, within that period, the leader has enough time to send a message to all, and each node has enough time to send back an acknowledgment. This timeout delay is crucial as it affects the interval for calling the roll and consequently the latency for detecting a failure. Thus, this timeout delay must be chosen carefully so that it neither causes too many false detections nor extends the detection latency. However, the round-trip time may vary depending on the network status or the node status. In the case where the network is loaded, the message sent can be lost or arrive after the expected time. One other case is when the node being checked is loaded, which causes the node to discard the message or to reply after some period. Consequently, we suggest that timeout delay should be adaptive. If a leader receives an ACK message from a node that is already in the Suspected List, this may indicate that the suspicion was not accurate, and hence the timeout delay is increased to lighten the load of the network or the load of the suspected node. If after some rounds of normal operation, no suspected node is incorrectly suspected, then the timeout delay is adjusted again. Note that the timeout delay is chosen so that it is always greater than the round-trip time delay.

3.4. Recovering from Failures

The leader that detects the failure of a node is in charge of fixing it. The failure recovery procedure is process dependent as every process might perform different actions while restarting or resuming a job. Since we adopt an MOM-based architectural style, the order for starting each node does not matter. Additionally, the nodes are mutually independent and stateless, and therefore to recover the failure node we just need to re-execute the process. At re-initialization, the recovered node looks for the other nodes in the environment using the SSDP M-SEARCH. Additionally, PHMs must retrieve the information about nodes running and being installed in the current host and PSMs must look for each of the nodes that compose the pervasive service. Next, the recovered node waits for the CHK or LDR message from the leader. As mentioned before, if no CHK or LDR message is received after a period, the node that detects it starts a Leader Election process.

4. Analysis

In this work, we consider a fail-stop failure and a fair-loss link model. In a fail-stop model, a node crashes at time t, after which it does not send any message to any other node and stops executing any local computation. Meanwhile, the fair-loss link model guarantees that a link does not systematically drop any given message. Therefore, if neither the sender process nor the recipient process crashes, and if a message keeps being retransmitted, the message is eventually delivered [13].

4.1. Accuracy

With the proposed approach, a healthy node replies with an ACK message when it receives a CHK message. However, the CHK may be received after the timeout delay. In this case, the next leader will still send CHK message to the suspected node, and if this time the current leader receives the ACK message from the suspected node, the suspicion is corrected. To avoid making false detections, we adopt a semi-consensus technique. Consensus is a process where distributed nodes agree on some common value or decision. A node making its decision purely alone is more likely to incorrectly detect a failure than the majority of nodes making the decision together. Therefore, the proposed approach is designed so that a node is considered as failed only when more than half of the nodes in the environment suspect it. Although this does not guarantee perfect failure detection, where no false detection exits, it reduces the probability of making false detections.

Let us denote k to be the number of nodes, f to be the number of failed nodes, j to be half of the number of healthy nodes, namely, $j = (k - f) / 2$, and P_a to be the probability that the message arrives successfully at the remote node before the timeout delay. In each round, the leader sends out a CHK message to the rest of the nodes, and each of the healthy nodes responds with an ACK. Then, the leader will send a LDR to the next leader and the next leader responds with a LACK. Therefore the probability a round is processed successfully is:

$$P_s = P_a^{(k-1)} + P_a^{(k-f-1)} + P_a^{\,2} = P_a^{(2k-f)} \tag{1}$$

Then, the probability that a round fails to proceed is $P_f = 1 - P_s$. However, RPM tolerates that some rounds fail to proceed as a failed node is first suspected and considered as failed only after half of the number of nodes suspect it. Therefore, as long as the there is no consecutive j failed rounds, RPM works correctly. Otherwise, RPM would make false detections. Hence, the probability that a failure is correctly detected becomes:

$$P_c = 1-(P_f)^j = 1-(1-P_s)^j = 1-(1-P_a^{(2k-f)})^j$$

For example, if there are 10 nodes in the environment where 2 of them have failed, and let P_a be 0.98. By (1) one can conclude that the probability of correctness is higher than 99\%.

4.2. Performance

The time to detect the failure depends on the timeout delay. With the proposed approach, the timeout can be easily adjusted to detect a failure instantly and to maintain the stability of the network at the same time. The timeout delay is initially set to the round-trip time delay plus the processing time. If a node is suspected mistakenly; in other words, it was suspected by a leader, but then an ACK was received by another leader before it is considered as failed again, then the timeout delay is increased. This may extend the detection time, but it also ensures the network stability. If no node is suspected mistakenly for a certain number of rounds, the timeout delay is set back to the initial timeout delay value to minimize the failure detection time.

We can also compare the communication complexity (message count) of RPM and SSDP for detecting a node failure. Consider a total number of \$n\$ nodes in the environment, where there are k Manager Nodes and s Worker Nodes ($n = k + s$). For each round, the number of messages traveled along the network is shown in Table 2.

Table 2. Messages sent and received in each round.

RPM			SSDP		
CHK	Sent	1	ssdp:alive	Sent	$4 \cdot r \cdot n$
	Rcv	k		Rcv	$4 \cdot r \cdot n^2$
ACK	Sent	$k-1$			
	Rcv	$k-1$			
Total		$3 \cdot k - 1$	Total		$4 \cdot r \cdot n \cdot (n+1)$

In each round, RPM sends out one CHK packet to the multicast group which is received by the k Manager Nodes in the environment. Each of the k - 1 nodes, excluding the leader itself, responds with an ACK which is received by the leader. As ACK is sent using unicast, the number of ACK packets sent is equal to the number of packets received. While for SSDP, in each round, each of the \$n\$ Pervasive Nodes, including Worker Nodes and Manager Nodes, sends out an "ssdp:alive" advertisement. Each advertisement is sent out 4 times in this case, and r is the SSDP repeat factor (the redundant messages sent by the devices to avoid message loss). As SSDP is multicast, every Pervasive Node receives the advertisement of every node. To detect a node failure, RPM must run k / 2 rounds, while the minimum number of round for SSDP to detect a failure is one, which is when a failed node fails after sending a "ssdp:alive" and the rest of the nodes detect the failure after the "ssdp:alive" sent by the failed node expires. Therefore, the number of message required to detect a failure for RPM is *k* / 2 · (3 · *k* - 1) and for SSDP is 4 · *r* · *n* · (*n* + 1). Based on the above analysis, it can be concluded that the number of message count is reduced significantly.

5. Evaluation

In this section, we present the implementation and the experimental results of the proposed approach. In this paper, we implement and evaluate protocols by using NS-2 and AgentJ [24]. AgentJ contains a complete native implementation for integrating Java with NS-2. For the network configuration, we simulate a LAN which is used for the most ambient environment such as smart homes. Duplex links are used to connect nodes, having a bandwidth of 100Mb, the propagation delay of 10 milliseconds and a Drop-Tail queue. The limit of the buffer size of the queue is set to 100. We then simulate the network adopting a star topology, where every device and host is connected to a central router or switch, by dedicated links.

5.1. Metrics

The lifecycle of a failure detection and recovery procedure is indicated in Fig. 6. MTTF is the Mean Time to Failure, MTBF is the Mean Time Between Failures, and MTTR is the Mean Time to Repair. MTTF is the average time the system takes to operate until a failure occurs, MTBF is the average time between two consecutive failures, and MTTR is the average time taken to repair the faulty component. In other words, MTTF is the system uptime, and the MTTR is the system down time. To repair faulty component, we must first detect the failure, which is the difficult part, and then the recovery procedure begins. That is why we have decomposed the MTTF into Failure Detection Latency and Failure Recovery Latency. In the experiments, the objective is to investigate the availability and efficiency of our RPM. According to Koren et al. [17], availability is the probability that the system is up at some random time, and is significant only in systems that perform repair of faulty components. The long-term availability can be calculated as follows:

$$Availability = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR}$$

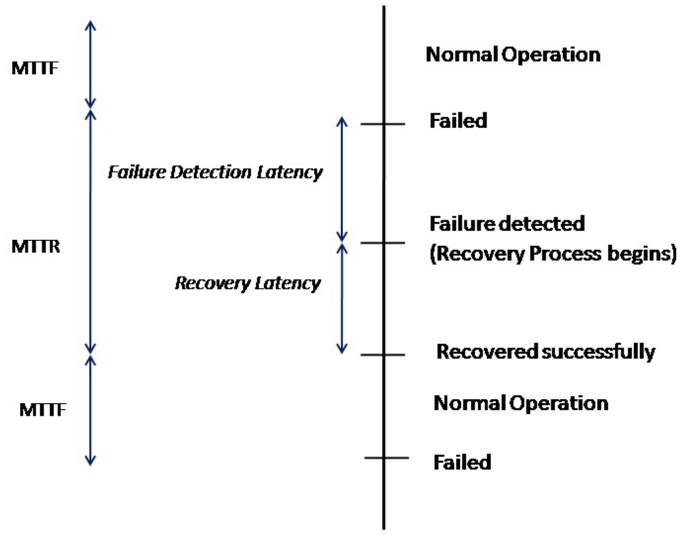


Figure 6. Failure Detection and Recovery Lifecycle.

However, the MTTF is hard to set as we cannot control the MTTF. Therefore, MTTF is considered as a constant and MTTR is measured instead. We can then achieve high availability by minimizing the MTTR. On the other hand, efficiency is measured as the total number of messages traveled along the network to reach the desired state. For failure detection, we measure the message count since the node fails until it is detected by a leader. For failure recovery, we measure the message count since the recovery process begins until the initial configuration is re-established. Efficiency is an important factor as one does not want to flood the network with unnecessary messages.

5.2. Configuration

The experiment compares the failure detection latency and message count between RPM and SSDP. The failure recovery latency of RPM is also measured. What has to be noticed is that since SSDP itself does not support failure detection, SSDP devices cache all the "ssdp:alive" advertisements and suspect a node when that nodes advertisement expires without receiving an "ssdp:byebye" from that node. The latencies and message count is measured as follows. The failure detection latency is the time from a node fails to the time when all the nodes are aware of the failure. Failure recovery latency is the time from the leader re-executes the node to the time when the initial configuration is re-established, and all the nodes are aware of the existence of the recovered nodes. Message count is the number of messages delivered through the network during the failure detection and recovery latency.

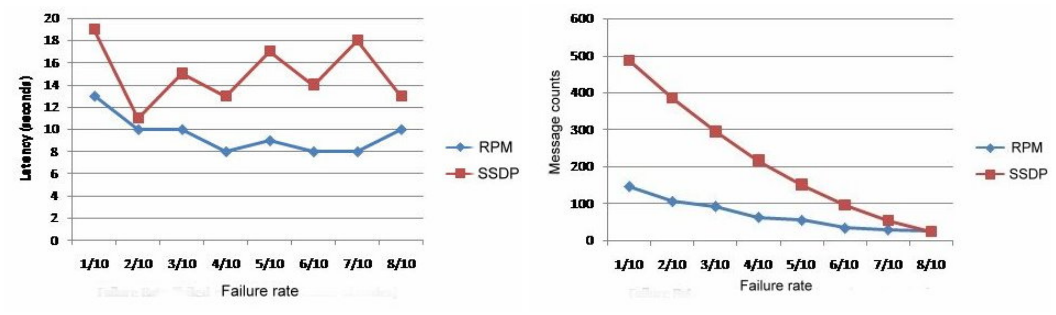
The experiments are conducted with two different configurations. In the first configuration, there are totally ten nodes, and the number of failure nodes varies from 1 to 8. In the second configuration, the number of failure nodes is fixed to 5, but the number of total nodes varies from 5 to 35 in an increment of 5. Each experiment is repeated for 80 times, and ten repetitions for each failure rate. The initial timeout delay of RPM is set to 2500 milliseconds. This delay is set considering the round trip transportation delay plus the random delay accounting for the waiting time of a node after receiving the CHK message to reply with an ACK. To be fair when being compared with RPM, we set the cache-control parameter of SSDP to 2.5 times of the total number of nodes with a unit as second, which is equivalent to every node in RPM takes one turn to be the leader.

5.3. Results and Discussions

The experiment results of failure detection latency with increasing failure rate are shown in Fig. 7a. The failure detection latency of RPM decreases slightly as the failure nodes increase. This is because only healthy nodes can become a leader and only healthy nodes participate in the semi-consensus process, and therefore semi-consensus is attained faster when there are fewer healthy nodes. The failure detection latencies of SSDP for increasing failure rates are more unstable. This is because the failed nodes fail randomly if the failed nodes fail just after sending out an "ssdp:alive" then the failure detection latency would be longer. Hence the failure detection latency of SSDP is bound to the SSDP cache-control. As shown in Fig. 7b, SSDP generates much more messages than RPM, and this is because SSDP uses multicast, as mentioned above, so that the message count drops rapidly as more nodes fail.

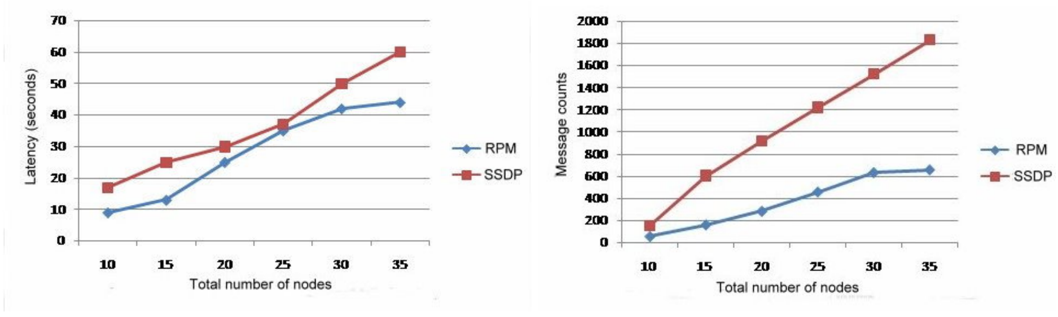
Fig. 8a and 8b show the experiment results of the second configuration for various total numbers of nodes. Fig. 8a indicates that both RPM and SSDP failure detection latency increases as the number of node increases, nevertheless, RPM failure detection latency is shorter than SSDP in general. We can see in Fig. 8b that there is a steep rise in SSDP message count as the total number of node increases; however, RPM message count grows gradually and steadily. Thus, we can conclude that RPM is more robust than only using SSDP for failure detection.

Fig. 9 and Fig. 10 show the experiment results of RPMs failure recovery latency and message count. Note that we do not compare them with those by SSDP because RPM uses SSDP presence announcements when performing service recovery. In the first configuration, the time for recovery does not vary too much as the number of failure node increases, as shown in Fig. 9a. This is because all the failing nodes are re-executed at the same time and uses SSDP to send out their presence announcements. Therefore, the other healthy Pervasive Nodes will notice their presence at the same time.



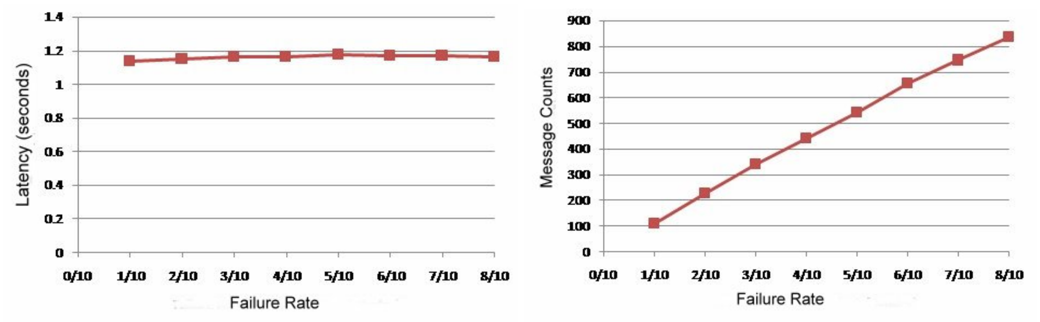
(a) (b)

Figure 7. Failure detection and failure rate: (a) latency, (b) message counts.



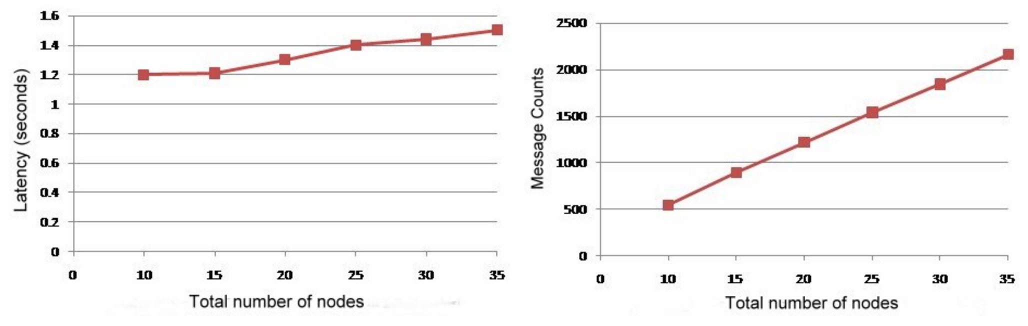
(a) (b)

Figure 8. Failure detection and node counts: (a) latency, (b) message counts.



(a) (b)

Figure 9. Failure recovery and failure rate: (a) latency, (b) message counts.



(a) (b)

Figure 10. Failure recovery and node counts: (a) latency, (b) message counts.

There is a slight increase in the failure recovery latency as the total number of node increases, as shown in Fig. 10a. Although multicast is used to send out "ssdp:alive" messages, however when there are more nodes, a little more time is needed for all the nodes to actually receive the "ssdp:alive" message. The message counts in both configurations rise sharply, as shown in Fig. 9b and Fig. 10b, this is because as more nodes fails, more nodes need to send out presence announcement messages after recovering. The failure recovery latency is less than two seconds with varying total number of nodes and varying failed nodes.

6. Related Work

Many pieces of research focus on developing ambient middleware based on Service-Oriented Architecture (SOA). Service-Oriented pervasive middleware falls into two categories. In the Program-Oriented architecture, the logic of flow is controlled by a program, which searches services in a centralized service registry and invokes services in a synchronized way. The classical Context Toolkit [21], and more recently, Santos et al. and Dey et al. [22, 8] belong to into this category. Many problems arise due to synchronous and centralized nature. First, the services built based on Program-Oriented architecture are usually tightly coupled: they are usually bound to a static IP/Port. Hence, the distributed components must start in a strict order. Moreover, all clients are affected when one of them fails. Second, the failed services are hard to recover because they are tightly coupled. To resume a service, all components must shut down and then restart in a strict order. Finally, the distributed components communicate in a synchronous fashion, and therefore service provider and service user must both be available for communicating at the same time.

More robust ambient middleware such as Tuple-space (TS) [11] and Message-Oriented Middleware (MOM) [3] are proposed to avoid the problems of Program-Oriented architectures. TS is a way to access shared information across multiple concurrent clients, whereas MOM focuses on reliable message delivery. TS and MOM have similar advantages, namely, easy to integrate heterogeneous hardware/software and failure isolation. However, they are two completely different architectures from the technologys point of view: TS is a way to access shared information across multiple concurrent clients, whereas MOM focuses on reliable message delivery. TS tends to store serialized objects, which is usually a penalty on performance and causing the TS server to become a bottleneck. Besides, the TS based solution is often language dependent. On these grounds, we argue that MOM is a more promising architecture for the pervasive system. For example, MQTT (MQ Telemetry Transport) [4] is a popular MOM-based connectivity protocol standard for Internet of Things (IoT) in recent years [18, 1].

An ambient middleware usually accompanies with service management protocols that facilitate interactions among heterogeneous smart devices. Among them, Jini [2], Universal Plug and Play (UPnP) [25], and Service Location Protocol (SLP) [14] are the most often discussed. Among these protocols, UPnP contains a well-known service discovery protocol, SSDP, which is designed for smart homes. The SSDP (Simple Service Discovery Protocol) of UPnP differs from other protocols in that it uses a more decentralized, and therefore more robust, approach which is preferred in an ambient environment.

Several mechanisms have been proposed to enhance SSDP. Nakamura *et al.* [16] studied the efficiency issues with interconnecting UPnP gateways. They proposed to store SSDP Presence Announcement messages in the caches of UPnP gateways to reduce the service discovery traffic. Knauth *et al.* [23] proposed to reduce traffic by introducing proxies among UPnP Devices that serve as cache in LAN (Local Area Network). In [15], the authors enhanced GENA (General Event Notification Architecture), a TCP-based sub-protocol of UPnP used for event notification, by realizing GENA based on IP multicast.

SSDP provides only naive mechanisms for failure detection, and none of them supports failure recovery. As a result, failure detection and recovery are left to the applications and are guided by application-level persistence policies [7]. Traditionally, a system detects failures by using periodic-broadcast-based protocols such as Heartbeat or Polling [6]. These protocols usually depend on centralized failure detector which can become a single point of failure. On the other hand, more robust

Gossip-based techniques [5] are proposed to be a prominent method for detecting failures without a centralized detector. Gossip techniques are designed for large-scale systems because it is highly scalable, but unfortunately, they also introduce considerable detection latency. Unlike in enterprise environments, most ambient environments are with smaller network scale. Therefore, the approach proposed by this paper is to strike a balance between detection latency and robustness by proposing a Gossip-like roll-call-based adaptive algorithm.

7. Conclusion

Robustness is a critical aspect of an ambient service management platform, especially when services are deployed in Smart Homes, in which the services can have a great impact on the healthy or lives of habitats. As little attention has been given to the robustness of such platform, in this paper, we propose a collective and adaptive failure detection protocol, RPM, as well as its supporting failure recovery mechanisms to detect node failures and to fix them in minimal time without a single centralized monitor. RPM is also capable of reducing the probability of false detections and to reduce the failure detection latency. Both analytical and experimental results show that the proposed approach is robust even under elevated failure rate and the number of messages generated is much less than that by SSDP. Nevertheless, from the experiments, we also learned that there is a rapid growth in failure recovery message count as the total number of nodes increases. For this reason, we are currently investigating efficiency boosting techniques to reduce traffic.

8. Acknowledgements

This work is partially sponsored by Ministry of Science and Technology, Taiwan, under grant 105-2221-E-004-004, 105-2627-E-002-003, and 105-2218-E-004-001.

9. References

[1] R. Anand and H. Raj, "Reliable communication for sustainable energy efficient low power smart home application (selsa)," in Proc. of International Conference on Internet of Things and Applications (IOTA), 2016, pp. 281-285.

[2] K. Arnold, B. O'Sullivan, R. Scheifler, J. Waldo, and A. Wollrath, *The Jini Specification*, Addison-Wesley, 1999.

[3] G. Banavar, T. Chandra, R. Strom, and D. Sturman, "A case for message oriented middleware," in Proc. of International Symposium on Distributed Computing (DISC), 1999.

[4] A. Banks and R. Gupta, *Mqtt version 3.1.1 protocol specification*, 2014.

[5] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri, "Efficient gossip protocols for verifying the consistency of certificate logs," in Proc. of IEEE Conference on Communications and Network Security (CNS), 2015.

[6] R. Corti˜nas Rodr´ıguez, "Failure detectors and communication efficiency in the crash and general omisi´on failure models," Ph.D. dissertation, Arquitectura y Tecnología de Computadores, Universidad del País Vasco, 2014.

[7] E. Dabrwoski and K. Mills, "Understanding self-healing in service discovery systems," in Proc. of the Workshop on Self-healing systems, 2002.

[8] S. Dey, A. Roy, and S. Das, "Home automation using internet of thing," in Proc. of Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), 2016, pp. 1-6.

[9] S. Dixit and R. Prasad, *Home Networking Challenges*, Wiley-Inderscience, 2008.

[10] W. K. Edwards and R. E. Grinter, "At home with ubiquitous computing: Seven challenges," in Proc. of International Conference on Ubiquitous Computing (UbiComp), 2001, pp. 256-272.

[11] D. Gelernter, "Generative communication in linda," *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 1, 1985.

[12] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall, "Systems directions for pervasive computing," in Proc. of Workshop on Hot Topics in Operating Systems, 2001.

[13] R. Guerraoui and L. Rodrigues, *Introduction to Reliable Distributed Programming*, Springer-Verlag, New York, 2006.

[14] E. Guttman, "Service location protocol: automatic discovery of IP network services," *IEEE Internet Computing*, Vol. 3, No. 4, pp. 71-80, 1999.

[15] C. L. Hu, Y. J. Huang, and W. S. Liao, "Multicast complement for efficient UPnP eventing in home computing network," in Proc. of IEEE International Conference on Portable Information Devices, 2007.

[16] K. Nakamura, M. Ogawa, T. Koita, and K. Sato, "Implementation and evaluation of caching method to increase the speed of UPnP gateway," in Proc. of IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008.

[17] C. Koren and I. Krishna, *Fault-Tolerant Systems*, Morgan-Kaufman, San Francisco, CA, 2007.

[18] X. Li, L. Nie, S. Chen, D.Zhan, and X. Xu, "An IoT service framework for smart home: Case study on HEM," in Proc. of IEEE International Conference on Mobile Services, 2015, pp. 438-445.

[19] C. F. Liao, Y.W. Jong, and L. C. Fu, "Toward reliable service management in message-oriented pervasive systems," *IEEE Transactions on Services Computing*, Vol. 4, No. 3, 2011.

[20] U. Saif and D. Greaves, "Communication primitives for ubiquitous systems or rpc considered harmful," in Proc. of ICDCS International Workshop on Smart Appliances and Wearable Computing, 2001.

[21] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: Aiding the development of context-enabled applications," in Proc. of International Conference on Human Factors in Computing Systems (CHI), 1999.

[22] P. Santos, J. Casal, J. Santos, L. Varandas, T. Alves, C. Romeiro, and S. Louren¸co, "A pervasive system architecture for smart environments in Internet of Things context," in Proc. of International Conference on Multimodal Interaction (ICMI), 2015.

[23] S. Knauth, R. Kistler, D. Kaslin, and A. Klapproth, "UPnP compression implementation for building automation devices," in Proc. of IEEE International Conference on Industrial Informatics, 2007.

[24] I. Taylor, B. Adamson, I. Downard, and J. Macker, "Agentj: Enabling java ns-2 simulations for large scale distributed multimedia applications," in Proc. of International Conference on Distibuted Frameworks for Multimedia Applications, 2006, pp. 1-7.

[25] UPnP, *UPnP Device Architecture 1.1, ISO/IEC 29341*, UPnP Forum, 2008.

[26] T. Winograd, "Architectures for context," *Human-Computer Interaction*, Vol 16, No. 2-4, 2001.

[27] F. Zhu, M. W. Mutka, and L. M. Ni, "Service discovery in pervasive computing environments," IEEE Pervasive Computing, Vol. 4, No. 4, pp. 81-90, 2005.