Parser as a Novel Reliability Implementation in the Pervasive Computing

Erwin Adi, Michael Sunur, Kenrick Tjandrean, Minaldi Loeis
School of Computer Science, Binus International
Bina Nusantara University
Jakarta, Indonesia
eadi@binus.edu, michael.sunur@gmail.com, kenrick_92@yahoo.com, minaldi@binus.edu

Abstract—**The ubiquity of mobile devices affects the way society works beyond voice and text messaging. Smart phone capabilities have become similar to those of computers. They promote users to engage in social networking, flash reports, and other vital applications. These mobile devices can also be used to control other devices. However, the heterogeneity of operating systems, hardware, and protocols has brought about the challenge of ensuring that messages could be reliably transferred between these mobile devices with different communicating equipments. Hence in this paper, we showed how a parser could be used as a reliability mechanism over our proposed system. Additionally, we showed how message queue was used as a technique for the pervasive computing to interoperate.**

Keywords-Parser; message queue; reliability; distributed systems; pervasive; ubiquitous

I. INTRODUCTION

The rise in the use of mobile devices has caused changes in the way communication technologies are engineered. The old paradigm of the client-server model in which a client would request a computing service from a server has shifted to more ubiquitous solutions in which all devices could communicate with each other [1]. However, since mobile devices have smaller memory and processing power than desktop computers, programming techniques should consider a digestible architecture for these small devices to interoperate with other devices aside from computers. For example, turning on a home light through a mobile device has become a reality rather than fiction.

Integrating inanimate objects means increasing the degree of pervasiveness of computing which is one of the important challenges of the abovementioned distributed systems.

Interoperability among all devices is the utmost requirement of pervasive computing [2]. However, this has proven to be quite a challenge. For instance, it is difficult for software applications to interoperate due to the lack of ability of the current techniques to locate building objects [3]. New middleware technologies require adaptation of standard specification, which could be timely to implement [4]. CORBA, another interoperability implementation also poses some ambiguity, does not guarantee substitutable implementations [3], and is not an efficient solution for a fault management system to implement reliability mechanisms [4].

We therefore, propose parsing as a reliability technique for a remote device to reliably send messages to other internet-connected devices, or things. The purpose of our research is to show that reliability mechanisms on the pervasive computing could be implemented over the proposed parsing technique.

The remainder of this paper is organized as follows: Chapter II explains the motivation behind our research. Chapter III breaks down the design of our experiment. Chapter IV describes how testing was done, and Chapter V discusses how we tested the reliability of our proposed system. Finally, the paper is concluded in Chapter VI.

II. BACKGROUND MOTIVATION

Consider a mobile device that could turn a home light on or off as described in Figure 1. The device needed to connect a home light—described as "Things" in the figure—through the Internet. The Data Terminating Equipment (DTE) at home needed to connect to the Internet through a DTE at an Internet Service Provider (ISP). However, this connection would normally be transported over a local IP address. There was then a need for the transported message from the device at the public Internet to locate the home light behind a local IP address. Hence an Inter Process Communication (IPC) technique was required to enable both ends to interoperate. IPC is a set of methods for the exchange of data among multiple processes that run on different computers [5]. Several IPC techniques are pipes, shared memory, memory-mapped file, MQ, etc.

However, it is essential for the remote device to confirm message delivery and execution. For example, it would be in

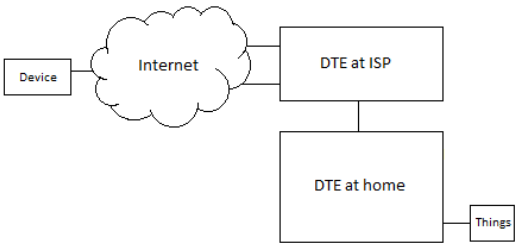

Figure 1. The Internet must reach the things through a local IP address between ISP and home.

DOI: 10.5176/2010-3043_2.3.194

the best interest of the user of the remote device that a message to "Turn on water heater" did not leave water heater off by the time the user arrived home. Hence, a two-way communication was required in this sense. A reliability mechanism should enable the things at home to give feedback to the remote device to confirm whether the command was carried out.

To illustrate and test this scenario, we implemented a home automation system to determine which architecture enabled mobile devices interoperate with remote things reliably.

III. Design of the Proposed System

A pipe is a unidirectional IPC. Hence, pipes were not suitable for our implementation. Other IPC techniques such as shared memory and memory-mapped file need the hardware, i.e., the memory and the disk respectively, to be installed under the same machine where the two processes communicate. Hence, they were not suitable for the home automation system where there were two DTEs—at the ISP and at home. MQ served to solve the communications of the two DTEs.

To see how MQ could ensure reliability for the pervasive computing, we designed a system that could control a relayed I/O board remotely. Figure 2. shows that our system consisted of five layers. It follows the naming order of Open System Interconnection, i.e., the bottom or the physical layer being the first layer.

The fifth layer was an application server. The server was used to deploy a web application so that users could access our system through the Internet. We used GlassFish 3.1 as a server, while the web application was made out of JSP pages.

The fourth and the third layers, being the server and the client respectively, carried the MQ which served as the inter process communication mechanism between two machines—the DTE at the ISP and the DTE at the subscriber. However, we implemented these two layers under the same machine. All the five layers in the diagram were implemented within one computer. Our research did not intend to observe delay and throughput caused by the communications channel between the two DTEs, hence we viewed that this arrangement was appropriate.

We hypothesized that one MQ technology could deliver a more integrated solution over the others for our system. Hence, we implemented the fourth and the third layer using three different technologies: OpenMQ, ActiveMQ and RabbitMQ, substituting one over the other during the testing. We found no significant differences between these implementations. However, OpenMQ was the easiest to implement since its documentation was clear and examples were plenty.

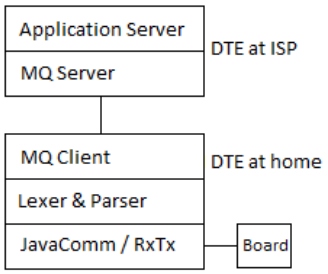

Figure 2. The home automation system was a five-layered architecture that controlled an I/O board.

A lexer was used at the second layer to check the vocabulary of a message that had been received by the system. The token was at one-word level. After passing the lexer checker, a parser was used to check the grammar. Our strategy for parsing was to to look at the Predicate + Object construction, which was suitable for home automation scenarios such as "Turn on living room TV". The following is a snippet of code to illustrate our parser.

```
if(myArr.get(0).equalsIgnoreCase("open")){
  if(myArr.get(1).equalsIgnoreCase("the")){
    if(myArr.get(2).equalsIgnoreCase("living")){
      if(myArr.get(3).equalsIgnoreCase("room")){
        if(myArr.get(4).equalsIgnoreCase("door")){
          return 1;
        }
        if(myArr.get(4).equalsIgnoreCase("curtain")){
          return 2;
        }
      }
    }
    if(myArr.get(2).equalsIgnoreCase("room")){
      if(myArr.get(3).equalsIgnoreCase("door")){
        return 3;
      }
    if(myArr.get(3).equalsIgnoreCase("curtain")){
        return 4;
    }
  }
  if(myArr.get(2).equalsIgnoreCase("toilet")){
    if(myArr.get(3).equalsIgnoreCase("door")){
      return 5;
    }
  }
}
else
...
```

In order to transfer a series of messages to the board, the first layer acted as the interface between the application programs at the upper layers and the physical ports at the board. Originally, we used Javacomm library for this layer. However, after running well on the system for a while, the board started to receive messages other than what had been originally sent by the system. Migrating the library to RxTx fixed this problem with no other change in the system. Hence, we assumed that the problem we observed was due to an undocumented phenomenon of Javacomm.

The board that we used was an Atmel-based microcontroller board. The purpose of this board was to continuously read a signal from the serial cable. The following code was downloaded onto the board, which served to illustrate its purpose:

```
while (1) {
  if (rx_counter > 0){
    x = getchar();
    switch(x) {
        case 'a': PORTA.0 = 1;
                  printf("on\n");
                  break;
        ...
    }
  }
}
```

The microcontroller board was connected to an Input/Output (I/O) board with eight relays that controlled lights. These lights were used as test signals to tell if we could turn them on and off remotely from a remote device.

IV. TESTING

Controlling the I/O board remotely was tested through three client devices: A generic mobile phone, an iDevice, and a web browser. Figure 3. shows how these testing devices were connected to the board.

Each of the three devices was able to successfully control the board by turning a light on and off. These devices were connected through the Internet to reach the home automation system.

1) Mobile phone: We built an SMS gateway that communicated with the home automation system. Hence, any mobile phone was able to send a message through its ubiquitous SMS capability. The SMS gateway essentially consisted of a DCE to receive a GSM signal, a physical layer to convert the signal into an API, a lexer and parser to tokenize the SMS message, and an MQ client. Connection between this part of the system and the home automation system relied on the MQ mechanism over the Internet Protocol.

2) iDevice: This client application designed to control the home automation system had several other features, such as scheduling switches according to different profiles. However, the detailed discussion of what our iDevice application could do was out of the scope of this paper. We ported the application to an iPod Touch 4th Generation, connected to the home automation system through an HTTP connection.

3) Web browser: A JSP application could be requested from any web browser, connected through an HTTP connection. The JSP application showed a drop-down menu that represented control messages such as " Turn on living room TV".

In order to test the reliability of the messages sent from these devices, we defined two scenarios: failure after the bottom layer, and failure before the top layer. To test the first scenario, we disconnected the cable between the home automation system and the board. A reliable system should be able to have a mechanism that fed this failure information back to the client. If no message had been sent from a client device, then the home automation system could simply reply to the client that its request could not be fulfilled. The challenge appeared when the line was cut after the client sent a message, but the board had not yet executed the message. Under this scenario, the messages from the MQ had been cleared. Therefore, we implemented a buffer at the first layer of the home automation system to hold the message from the MQ. This buffer served to hold the message until the first layer received a confirmation signal from the board that the message had been executed.

The second scenario described a failure at the Internet connection, preventing HTTP communication between the home automation system and the client device. If this failure happened before the client sent a message, then the user could

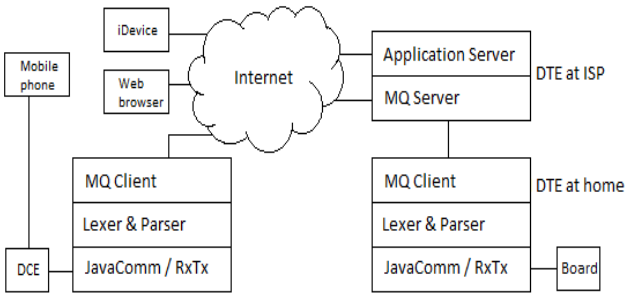

Figure 3. Three client devices used to test communication with the board.

have simply detected that there was a problem with connecting to the Internet. A more complicated situation would be had if the client device had already sent a message, but the Internet failed before the whole message was received by the home automation system. This situation was anticipated through an implementation at the parser. If the parser detected that the message was not understood, i.e., the grammar did not match, it informed the application server to give an appropriate error message back to the client in its response packet. (Obviously the client device could only receive this response message when the Internet was reconnected.) Figure 4 illustrates that a command had been successfully executed on the remote device.

The limitation of our experiment was that it was impossible to physically disconnect the line (any one of the two lines described above) before the message was actually executed by the board. The message traveled from the testing device to the I/O board faster than our hand could physically cut the line. Hence, we simulated this cut programmatically. We implemented a user interface that had a "disconnect" button, which essentially held the message to be forwarded. In reality, connection between the devices and the things could be separated by a wider magnitude, introducing a bigger chance where failure happened before the message arrived. We viewed that our methodology of cutting the line programmatically could simulate this situation.

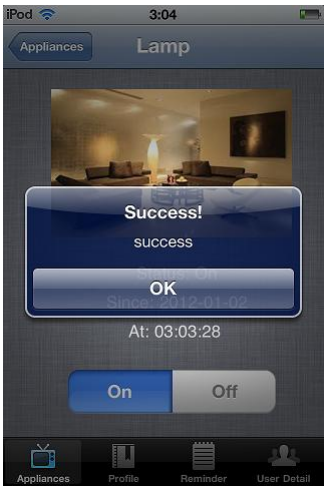

Figure 4. A confirm message on iDevice.

Figure 5. Web applications could avoid ambiguous inputs.

V. Discussion

Our parser attempted to find the left-most derivations of the user input, since the tokens were parsed from left to right to mimic the human behavior of reading and writing. This is known as a top-down parsing technique [6]. The drawback of this methodology is that the parser may spend an exponential complexity time when supplied by ambiguous inputs. However web applications could provide all possible inputs through the use of a dropbox which avoided this ambiguity (figure 5). Critics may argue that the exponential time to parse may come from SMS inputs, in which users could supply gibberish inputs. While this is true, SMS is by definition a short message. Regular users would not supply long and ambiguous inputs through an SMS. Malicious users on the other hand could attempt to overload the server's memory space through supplying long and ambiguous inputs, resulting a Denial of Service. This was a scenario that we did not test, and was open for further examination.

In addition, we tested the above implementation with IPC through a hardware implementation. Other aspects of work [7] [8] [9] were more focused on the software part of a distributed system and its performance. Another work [10] actually tested an MQ model through a hardware implementation, but it focused on network accessibility. Our work emphasized reliability mechanism. In this paper, we presented a novel approach that the reliability mechanism was implemented through parsing, and leveraging the reliability mechanism of an MQ.

VI. Conclusion

We have demonstrated that a parser can be used as a novel technique to implement a reliability mechanism between a client and a server over the Internet. Other experiments on parsers were to enhance the phrase finding performance itself [11] [12]. This can be incorporated as one of our future directions in order to reduce the shortcomings of our solution in the event that a malicious user supply a series of ambiguous inputs as discussed above.

Furthermore, we have shown that MQ effectively bridged the gap for the two ends in the pervasive computing to interoperate. It enabled things that were connected to a private IP address to be located by devices on the Internet. We demonstrated that a reliability mechanism could be implemented over MQ. The parser and the MQ have been presented as a novel fault detection mechanism.

VII. References

[1] K. Kang, J. Lee, K. Beak, S. Park, and J. Kim, "A mobile community service platofm promoting ubiquitous collaboration," in *IEEE 9th Int. Conf. Dependable, Autonomic and Secure Computing*, Sydney, 2011, pp. 939-946.

[2] E. Niemelä and J. Latvakoski, "Survey of requirements and solutions for ubiquitous software," in *Mobile Ubiquitous Computing Conf.*, Washington DC, 2004, pp. 71-78.

[3] R. Bastide, P. Palanque, O. Sy, and D. Navarre, "Formal specification of CORBA services: Experience and lessons learned," in *Proc. 15th ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications*, Minneapolis, 2000, pp. 105-117.

[4] J. Wan and H. Liu, "A Web/CORBA based architecture for network management with TCP connection," in *1st Int. Workshop on Education Technology and Computer Science*, Wuhan, 2009, pp. 463-466.

[5] Wikipedia contributors. (2012, January) Wikipedia, The Free Encyclopedia. [Online]. http://en.wikipedia.org/w/index.php?title=Inter-process_communication&oldid=470765737

[6] A.V. Aho, R. Sethi, and J.D. Ullman, *Compilers: principles, techniques, and tools*. Boston, USA: Addison Wesley, 1986.

[7] Z. Huang, Shengxi Zhou, and Shengru Tu, "Java communication interfaces for control systems," in *Proc. 21st Int. Computer Software and Applications Conf.*, 1997.

[8] M. Menth, R. Henjes, C. Zepfel, and S. Gehrsitz, "Throughput performance of popular JMS server," in *SIGMetric/Performance*, Saint Malo, 2006, pp. 367-368.

[9] R. Oechsle and T. Gottwald, "Disaster (Distributed Algorithms Simulation Terrain): A platform for the implementation of distributed algorithms," in *ITiCSE'05*, Monte de Caparica, 2005, pp. 44-48.

[10] V. Narayan, "Application integration environment for messaging/queueing model," in *Proc. 2nd Int. Symp. Autonomous Decentralized Systems*, 1995, pp. 169-174.

[11] K. Zhang, J. Wang, B. Hua, and X. Tang, "Building high-performance application protocol parsers on multi-core architectures," in *IEEE 17th Int. Conf. Parallel and Distributed Systems*, Taiwan, 2011, pp. 188-195.

[12] M. Lohuizen, "A generic approach to parallel chart parsing with an application to LinGO," in *Proc. 39th Annual Meeting on Association for Computational Linguistics*, Stroudsburg, 2001, pp. 507-514.

Erwin Adi has a Master degree in Telecommunications from University of Strathclyde, Glasgow, UK. His Bachelor degree was in Computer Science and Applied Mathematics/Statistics from State University of New York at Stony Brook, USA.

He has about 14 years of experience in computing technology. Early career includes being a Network Engineer in some telecom companies in Belgium. During the time he had gained experience in handling hands-on fiber network on the field, controlling European-wide network from the central operation under a wide range of platform, troubleshooting IP-related problems, and mitigating high-impact network failures. The complexity of the environment demanded him to finally learn some European languages (with some efforts).

He joined his family business in Indonesia for a couple of years and was responsible for marketing activities, while at the same time acted as the IT Solution and Infrastructure Manager. His passion in computing technology brought him to join Bina Nusantara University where he teaches, trains, and researches the network and security topics.