POLICY-BASED COOPERATION OF SERVICES IN UBIQUITOUS ENVIRONMENTS

Toshio Tonouchi, Tomohiro Igakura, Naoto Maeda, Yasuyuki Beppu, and Yoshiaki Kiriha
Network Laboratories, NEC

Abstract: Various kinds of nodes, including cellular phones and information appliances, are to become popular and are expected to provide a variety of services. Cooperation of these services will result in more convenient services than keeping them isolated would. A ubiquitous network is characterized by changeable system configurations. Because of this and the fact that a node is so frequently connected to and disconnected from the network, the global cooperation of services is difficult to describe in flow languages such as Web Services Flow Language (WSFL). One of the solutions to this problem is a policy technology. A policy attached to a node can be added or removed when the node is connected or disconnected. The policies can re-configure a changed system.

Keywords: Management of Grid Computing, Clusters, Peer-to-Peer Applications, and Ubiquitous Computing Environments, Policy, Message-oriented system

1. INTRODUCTION

The ubiquitous network environment is maturing. Cellular phones with Internet access, personal data assistants (PDAs), and wireless local area networks (LANs) are becoming more and more popular. About 10 years ago, Weiser developed an original PDA called 'Tab' and invented a proprietary protocol for wireless communication [1].

Some ubiquitous nodes, especially information appliances, provide simple services. For example, an air conditioner with network access function can be turned on and off or can have the temperature set by a

remote user. Cooperation of these simple services provides a valuable service. For example, a cellular phone with a global positioning system (GPS) can automatically give the location of the user to the network-connected air conditioner, which is automatically set to turn on when the user (e.g. Tom) comes near his house.

One of the characteristics of ubiquitous networks is that some of the ubiquitous nodes constituting the systems are not always operational. A cellular phone may be off when the battery is dead or the network-connected air conditioner breaks down. We call this characteristic *fickle*. A fickle node may suddenly disappear, and the system suddenly stops due to this. For example, when the air conditioner breaks down, the cellular phone cannot communicate with the air conditioner. A fan should work instead of the air conditioner when the air conditioner breaks down.

We propose a policy-controlled message-oriented system that overcomes the fickle-node problem.

2. RELATED WORK

A partial solution to the fickle-node problem is a publisher-subscriber system[2]. A publisher-subscriber system has a message router that automatically forwards a message to some of the nodes registered with the message router. Stopped and disconnected nodes will be manually unregistered. They can forward a message to adequate nodes in normal cases but they cannot handle the message when an error occurs. It is, therefore, difficult for the publisher-subscriber system to realize the example of the broken air-conditioner, which replies an error message.

Web Services Flow Language (WSFL[3]) and XLANG[4] were interesting trials for specifying the workflow among Web services. However, these technologies encounter the fickle-node problem because the description in the control flow languages requires deterministic routing information. The unplanned appearance and disappearance of nodes totally affects the workflow. The programmer therefore must rewrite the workflow.

3. ARCHITECTURE

Our architecture is basically the same as that for publisher-subscriber systems. The architecture is shown in Figure 1. A message router called the *distributor* is connected to a network. All the messages that the ubiquitous nodes (e.g., personal computers, PDAs, and cellular phones) send go through the distributor. The distributor determines where the messages go next.

Just as for the publisher-subscriber message-oriented systems, nodes that receive messages must be registered with the distributor a priori. In our system, policies describing which messages the joined nodes accept are also registered with the distributor.

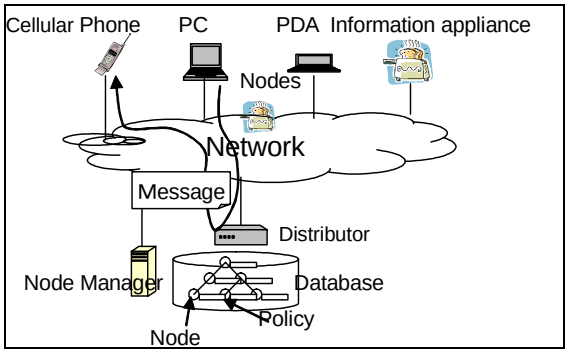

Figure 1. Architecture

Policies are the key to our architecture. We give an example of policies in Figure 2 (a). A policy is composed of a *matcher* (before "|") and a *generator* (after "|"). A matcher specifies what kind of message a node accepts. P_1 and P_2 accept any message because the matcher does not specify any condition. P_1 and P_2 could even accept the same message. However, the distributor non-deterministically chooses one of them.

The interesting syntax of our policy language is the generator. Generator "*" creates a copy of an accepted message and distributes it to the other policies. Suppose that P_1 accepts a message. P_1 forwards it to Node N_1. P_1 then generates an internal message copied from the original message with attribute "after=P_1". An internal message is a pseudo message that is used to explain the behavior of the policy processing of the distributor. Policy P_2 accepts the generated internal message because P_1 has already been used and only P_2 can be matched with the internal message. Next, suppose that P_2 is matched earlier than P_1. P_1 will match the message generated by P_2. In either case, N_1 and N_2 are chosen in the case of the example in Figure 2 (a).

N_3 is a 'fickle' backup server, whose policy, P_3, accepts messages that include "after=P_2". This means that a message to Node N_2 is copied to the backup server. N_1 and N_2 work well even if fickle node N_3 is removed. Only the backup function does not work. However, the backup of N_2 will work automatically when N_3 is connected to the distributor with Policy P_3. This shows that our policy approach solves the fickle-node problem.

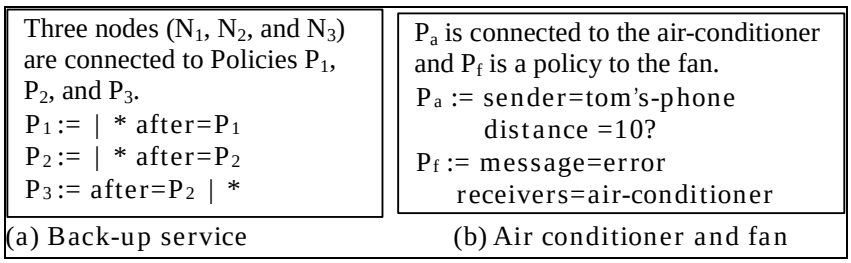

Figure 2. Examples of policies

Figure 2 (b) shows the policies of the example in Section 1. Policy P_a is fired when the distance between Tom's cellular phone and his house is less than 110 meters and more than 100 meters ("distance=10?"). If the air conditioner is broken, the error message is issued. The P_f is fired because it matches the error message. Notice that both P_a and P_f have no generator. These do not generate internal messages, and no more policy is fired.

4. CONCLUSION

We proposed a policy-based message system. We showed, using the examples, that this system solves the fickle-node problem. Nevertheless, the syntax and semantics of our policy language are inadequate. We are trying to improve the policy language without losing its simplicity. Another challenge is the effectiveness of policy processing. A distributor may have to handle a lot of ubiquitous nodes. In such a situation, fast policy processing is required. Therefore, we are now studying an optimization method for policy processing. The correctness of the optimization method is proved based on the operational semantics of our policy language.

REFERENCES

[1] Weiser, M.: *Some Computer Science Issues in Ubiquitous Computing*, Communication of the ACM, Vol. 36, No. 7, pp.74-84, July 1993
[2] Sun Microsystems, Inc: *Java Message Service*, 1999
[3] Leymann, F.: *Web Services Flow Language (WSFL Ver 1.0)*, May 2001
[4] Thatta, S.: *XLANG – Web Services for Business Process Design*, 2001