Integrated Service Navigation Framework for Ubiquitous Networking

By Toshio TONOUCHI,* Norihito FUJITA,† Naoto MAEDA,*
Tomohiro IGAKURA* and Yoshiaki KIRIHA†

ABSTRACT A huge amount of service components will soon be deployed to ubiquitous networks. Each will provide a simple service, but compositions of service components are expected to be highly valuable to users, because it is hard for users to find and combine adequate service components. We are developing a service navigation architecture in which a composite service is assembled from service components autonomously. We point out four issues to be addressed in the architecture: autonomy, efficiency, security, and reliability. In our architecture, these issues are handled by integrating management of applications and the management of networks. In this paper, we explain the technologies we are developing for this integrated management.

KEYWORDS **Service composition, Policy-based service management, Overlay network, Policy conflict**

1. INTRODUCTION

Ubiquitous networks are maturing. Cellular phones with Internet access, PDAs, and wireless LANs are becoming popular. About ten years ago, Weiser developed an original PDA called 'Tab' and invented a proprietary protocol for wireless communication[1].

Ubiquitous networks connect with a huge number of nodes including tiny sensors and large size of information appliances, and, by 2010, the number of nodes in these networks is expected to reach ten billion in 2010[2].

Each node, such as a sensor or a network-connected air-conditioner, provides only a simple service. For example, an air conditioner with network access function can be turned on and off, or set to hold a specific room temperature by a remote user. A collection of cooperating services is more useful than the set of independent services. A cellular phone with a GPS (Global Positioning System), for example, can autonomously give the location of the remote user to his network-connected air conditioner, enabling the air-conditioner to be turned on autonomously when the user comes near his house (**Fig. 1**). In this example, the requirement of the user is that "the room temperature should be 27°C whenever I am in the room." This requirement could be satisfied by having the air-conditioner run continuously and always keep the room temperature at 27°C, but this would be wasteful. The user would find it more economical to have his mobile phone, a home server at his house, and his air-conditioner collaborate with each other to satisfy the requirement.

The WSFL (Web Service Flow Language)[3] is a workflow language used for building functional collection of web-service components. The user who wants to receive a service satisfying his requirements must write a workflow in WSFL. That is, he needs to select

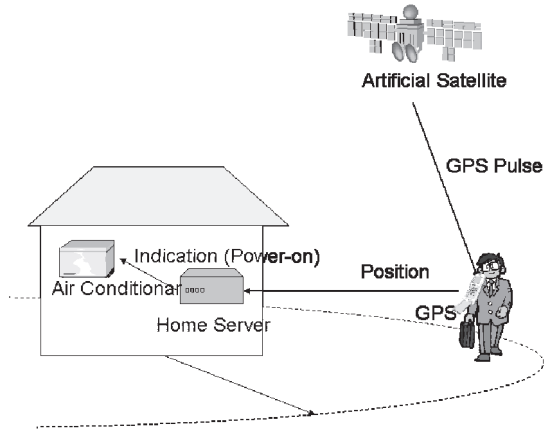

Fig. 1 Example of collaboration of service components.

*Internet Systems Research Laboratories
†System Platforms Research Laboratories

some of them and specify their interactions. It is difficult, however, for most users to specify workflows because a huge number of components are available and the most effective collection for any given purpose is not easy to find. An autonomic process assembling the appropriate components without human intervention would be extremely useful.

We are, therefore, developing a new service navigation framework in which a suitable workflow is autonomously generated in accordance with the user's requirements is generated automatically and in which the user is directed to the service provided with that workflow. This paper clarifies the issues of the service navigation framework and reports the current status of our work.

2. HISTORY OF DISTRIBUTED COMPUTING: FROM CENTRALIZED COMPUTING TO AUTONOMIC SERVICE COLLABORATION

Figure 2 shows the history and expected near-term future of the distributed computing architecture. By 1980, a centralized computing architecture had become popular because computing resources, such as CPUs, were very expensive. Users shared a huge expensive computer. At that time, the efficient use of computer resources was more important than the human operation's environment.

As computer resources became less expensive, programming costs became more significant. In other words, the cost of human resources became more important than the cost of computing resources. Object-oriented technology, easily mapping objects in real world to objects in the programming world, became popular because it reduced programming costs. It also virtualized computer resources because it separated the objects in the programming world from the computer resources. That is, the computer resources on which an object is running can be placed in a different computer, but the object running on those resources virtually placed at the same computing resource. In other words, a client/server computing architecture: the virtual object on the different computer is a server, and the invoker of the virtual object is a client. This is called an RPC (Remote Procedure Call).

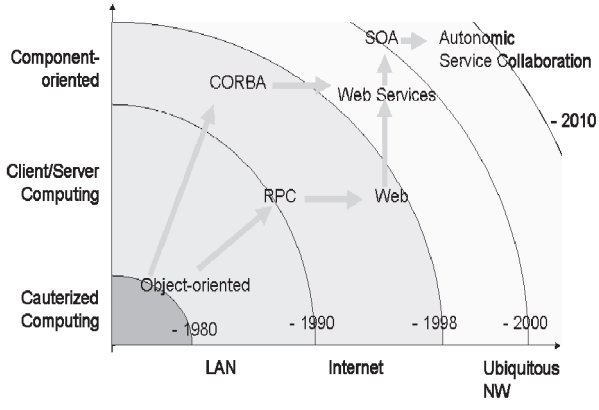

Fig. 2 Trend of distributed computing.

Definition methods of the interfaces of server objects and a protocol between a client and a server need to be standardized because these standardizations enhance the reusability of server objects. The most well known standardization of RPC is the CORBA (Common Object Request Broker Architecture), in which server objects are regarded as service components placed in a distributed environment.

Another popular client/server architecture is WWW. People noticed the necessity of RPC in the Internet because of the success of WWW. The conventional RPC, such as CORBA, implicitly assumes LAN because it does not consider the existence of firewalls. The IIOP, a standardized protocol of CORBA, cannot go through a firewall. The way to provide Internet-wide RPC is to use a web service technology. Its transport is HTTP, SMTP or some other protocols that goes through firewalls. In 1999 and 2000, SOA (Service-Oriented Architecture), in which services are implemented by collection of web-service components, was proposed.

These client/server architectures inherently assume that servers always stay and work. In the ubiquitous network environment, however, service components including sensors may be placed temporarily and may be temporarily removed. As a result, clients cannot expect that any given server is always running.

Suitable collection of service components may, therefore, change from one time to another. This means that we need an autonomic service navigation technology that assembles collections of service components in accordance with the load on or running states of service components and in accordance with network traffic.

An architecture providing the autonomic service navigation uses a broker server that has the specifications of service components and that has information about the running states of service components. When it receives a request packet with user's requirements, it checks its store of specifications and running states, identifies an appropriate collection of components, and generates a collaboration sequence satisfying the user's requirements.

3. ISSUES IN AUTONOMIOUS SERVICE NAVIGATION TECHONOLOGY

There are four issues in the autonomic service navigation technology.

The first issue in autonomic service navigation technology is finding a set of service components and generating a service collaboration that satisfies the user's requirements.

The service components in a collection assembled autonomously need to have machine-readable specifications because the broker assembling the collection has to know what the components are. Specifications should include the syntax of the interfaces of service components and the semantics of service components. The WSDL (Web-service Definition Language)[4] can define the interfaces of service components, but the semantics of a service component are beyond the scope of WSDL.

The standardizations of service components are progressing in some standardization organizations or some consortiums, and one example of their work is the NGOSS (New Generation Operations Systems and Software) proposed by TMF (Tele-Management Forum)[5]. In NGOSS, service components in a Tele-Management field are standardized. Service components will be more reusable when these standardizations become popular.

For standardized components, the specifications and the semantics of components are clear. We assume the broker will be able to know these specifications. We are currently using an in-house specification method, because, now, there is still no adequate and widespread specification method.

The second issue is the efficiency of the autonomous collection process. A huge number of service components will be distributed in coming ubiquitous networks. Better standardizations of service components will greatly increase the possibility of their combinations of service components in functional collection. Because the broker finds a collection dynamically, it must find it in a short time.

The third issue is the security of the communication between the user's node and the service components. Users could not trust composite services if others were able to tap this communication or issue counterfeit packets. In our service navigation framework, components are dynamically found, and the network path among them is also determined dynamically. We have been developing a technology to set a secure communication path dynamically.

The fourth issue is reliability. Autonomic service composition proceeds without human intervention, so reliability is more important in autonomic systems than in normal human-controlled systems. The composition is decided by a set of specifications of each service component, and the administrator of each component gives its specifications. Reliability thus depends on how to guarantee correct specifications. Specifications may change when the service component is updated, and they may be conflict with the existing specifications. A different administrator manages each service component, and it is difficult for one administrator to know what specifications another administrator defines. This results in potential conflicts, and decreases reliability.

A tool that finds pairs of potentially conflicting policies is, therefore, needed. Administrators can know potential conflicts with this tool, and they can correct policies.

4. SERVICE NAVIGATION PLATFORM ARCHITECTURE

4.1 Overview

Figure 3 shows our service navigation architecture, which has two technologies: an "Autonomic Service Control Technology" and a "Network Service Control Technology." The autonomic service control technology provides the function needed for the autonomic assembly of collaborations of service components (**Fig. 4**). The network service control technology provides network resource allocation suitable to those composite composed services.

In the autonomic service control technologies, there is an application router in the whole system. Each service component is assumed to have its service specification. A user issues his request packet including his requirements to the application router. The application router collects service specifications beforehand. When it receives the request, it compares the user's requirements with a set of service specifications. The application router has a policy engine running in the background. The application router decides how and what components are composed. The policy engine considers the requirements and the specifications as policies and processes them efficiently.

The network service control technology supports the service navigation platform architecture to satisfy nonfunctional requirements while the autonomic service management technology satisfies the functional requirement. An example of a nonfunctional requirement is a response time. A user requires that the response should return within five seconds. To satisfy these requirements, the service navigation platform

has to know the loads of service components and the network traffic between the components. It also has to choose service components and network paths in the view of resource usage. It has to choose idle components and a path with light traffic so that it avoids bottlenecks.

Another example of non-functional requirements is security. A lot of cases of service compositions require secret communication because security is the most important issue in some services. In service navigation platform, information about the network is gathered as resource information with a certain protocol. This information includes whether or not a network path is secure. For example, a path in the Internet is not secure or a path in a leased line is secure. The application router decides a path according to this

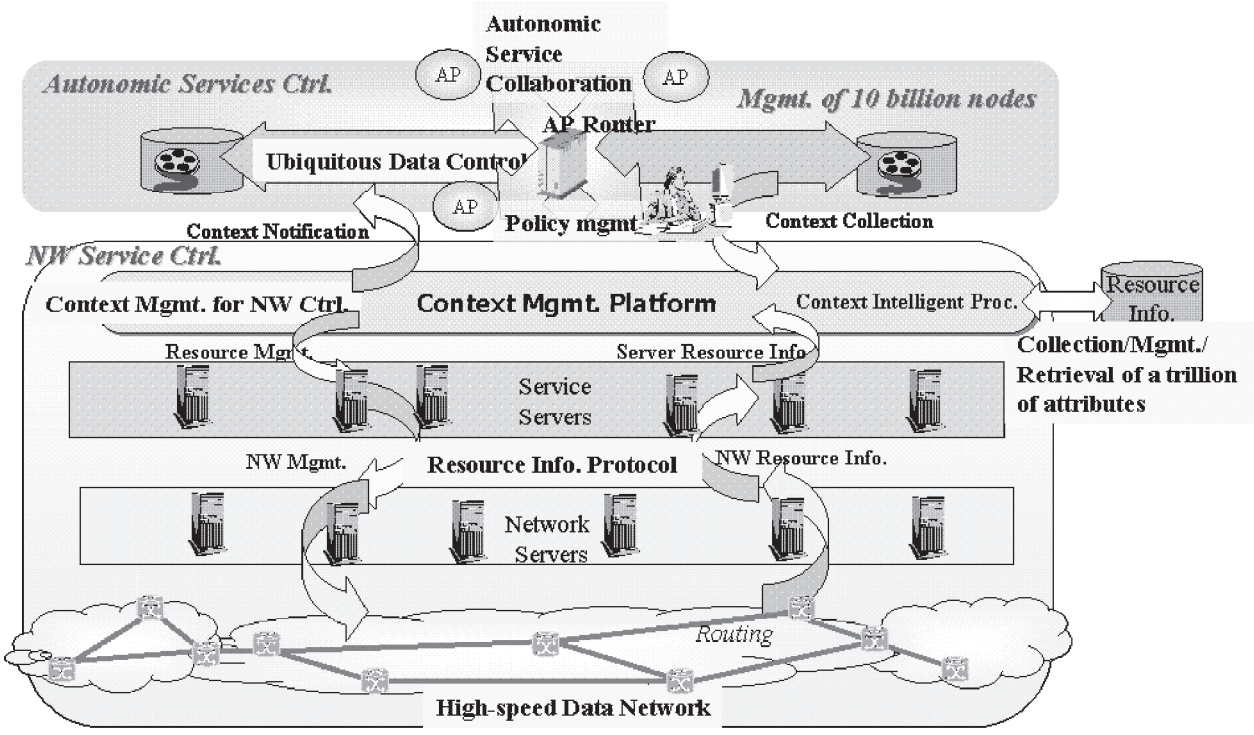

Fig. 3 Service navigation platform architecture.

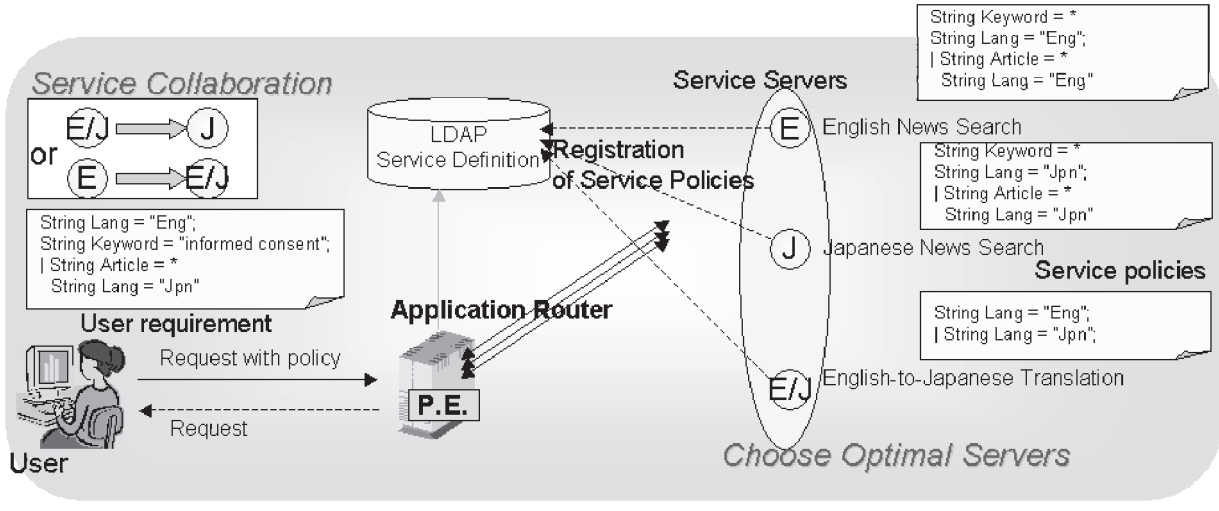

Fig. 4 Autonomic service collaboration.

information.

4.2 Solutions to Issues

This section gives solutions we are now studying to the four issues counted up in Section 3.

We explain our autonomic service collaboration, which is a partial solution to the first issue, through an example. Assume that a user wants to find news articles related to the English keyword "informed consent." A Japanese user wants articles written in Japanese. We can write this requirement as "Lang=Eng Keyword='informed consent' | Article=* Lang=Jpn."

We assume that there are an English article search service component, a Japanese article search service component, and an English-to-Japanese translation service component.

The specification of the English article search service component may be "Keyword=* Lang=Eng | Article='*' Lang=Eng." This means that the service accepts any keyword (Keyword=*) as an input. The input (i.e., Keyword) is written in English (Lang=Eng). The term on the right-hand side of the "|" specifies a reply message. The service component returns an "Article" (Article!=*) written in English (Lang!=Eng) as a search result. The specification of a Japanese article search service component is "Keyword=* Lang=Jpn | Lang=Jpn," and the specification of the translation service component may be "Lang=Eng | Lang=Jpn." The application router knows these specifications because it has collected them beforehand. The application router decides a suitable composition, for example "the English article search service component → the translation service component" or "the translation service → the Japanese article search component." We describe in Section 5.1 how these specifications and user's requirements are handled.

Consider now the second issue. As mentioned in Section 1, the number of nodes in ubiquitous networks is expected to reach ten billions by 2010. Because the number of possible collections of service components will be considerably huge, the policy engine must be a high-performance one. The Google search site manages six billions of items[6]. It can search them by a keyword search, and it can find out about thousands of indexes of contents. The application router may search these found indexes and results in a service composition. The policy engine running in the background of the application router must handle such a large number of contents in real time. The policy engine manages these contents by using the index mechanism described in Section 5.1.

A solution to the third issue is a mechanism for dynamic overlay network deployment technology. The service navigation framework has to decide dynamically which part of the network is to be secret because the composition of collaborative service is decided dynamically. Both nodes of an IPSec path must be configured consistently. This consistent configuration is bothersome. Our framework uses the DNS (Domain Name Server) protocol. Nodes on both sides of the IPSec path query the DNS server, and the DNS server replies to all of them with IPSec parameters. The DNS server can distributes consistent parameters to the nodes on the both sides. The dynamic overlay network technology is described in Section 5.2.

A solution to the fourth issue is the policy conflict detection technology we are now developing. We define conflict between policies as follows: when an event occurs, more than one policy is triggered and they perform inconsistent operations on one managed object. In the service navigation architecture, an event is a user request, and policies are specifications. Two services may be invoked when a user issues a request, and two policies may invoke the same service component with different arguments. This can result in inconsistency because the result depends on which policy is invoked first. The policy conflict detection technology can be adapted to an off-line command that finds possibility of conflicts. Our policy conflict technology is described in Section 5.3.

We have developed other technologies to complete the service navigation architecture. The network/computer resource information and user contexts are useful for efficient resource usage and find adequate service compositions. The context management platform technology collects contexts from various kinds of sensors, and translates them into a uniform syntax. For example, a position can be measured with RFID tag readers or with the GPS. The platform manages position data in the same syntax, and applications using position data need not care what kind of sensors are the source of that data.

The dynamic overlay network technology makes a closed network with the IPSec. If you put a cache in the closed network and if the cache is related to the users in the closed network, it will be efficient. The ubiquitous data management technology in Fig. 3 tries efficient cache management in ubiquitous network.

Because the number of nodes in ubiquitous networks is expected to increase greatly, the number of contexts is expected to become huge. Our large-capacity search technology provides an efficient search method for searching a huge number of

contexts. Sets of contexts are assumed to have inherently data dependency. Aware of the dependencies, you can divide the contexts into independent sets of contexts. And by focusing on the first set, the search can become more efficient.

Because a huge number of contexts go through networks, fast networks are required. High-speed network technology provides fast layer-2 switches.

5. TECHONOLOGIES

5.1 Autonomic Service Collaboration with Policy

In the service navigation architecture, the policy engine running in the background of the application router processes a user request as a router in accordance with the specifications of service components, which are considered as policies.

Figure 5 shows the architecture of a policy-based management system. A policy engine manages policies. It accepts events that represent a user request and then decides action or workflow in accordance with the policies represented by the specifications.

5.1.1 Policy Language

In our policy language, an event is assumed to be written as a set of properties such as "key=data."

The syntax of policy is composed of three parts, the first of which is an accept pattern. A policy only accepts events that have the properties required by the accept pattern. The second part is an action indicating a service component that is invoked when the policy accepts an event. The third part is a rule for modifying the output messages. An event accepted by the policy is modified in accordance with the modification rule, and the modified event is then accepted by another policy (if its accept pattern matches the event).

Consider an example in which "A=a, B=b | actionA | D=d." "A=a, B=b" is an accept pattern and "D=d" is a modification rule. If an event consisting of "A=a, B=b, C=c" is sent to the policy engine, the policy is activated. That is, actionA is executed, the event is then modified according modification rule, and the modified event of "C=c, D=d" is sent to the policy engine again.

Notice that the syntax of the policy language is similar to the syntax of the specifications in Section 4.2. The application router translates the specifications into policies, and sends the policies into the policy engine.

5.1.2 Techniques for Efficient Policy Processing

The larger a system becomes, the more policies are needed and the more events occur. In addition, the more policies there are, the more time is spent to search for a policy matching an event. So, we have therefore developed two techniques speeding up searching policies: "policy transition" and "event modification history."

Whether policies are activated depends on the state of the system. When some service stops, for example, policies that use the service must not be activated. That is, the policy engine can skip invocation check for some policies because they will be never activated in a certain state of the system. The "policy transition" is a mechanism to restrict the search area of policies. In the policy transition, certain policies become triggers changing a set of policies that can be activated. We call the set of these policies an active policy set. For example, first, a set of policy A, B, are C is an active policy set. If A accepts an event then policy B and C are no longer elements of an active policy set, and a set of policy D and E becomes an active policy set.

Policy's output messages are modified from the accepted events in accordance with the "modification pattern" of the policy, and the output messages are accepted by another policy whose accept pattern matches the message. A policy that did not accept a certain event cannot accept the event unless the properties of the event that does not match the accept pattern changes. That is, it is omissible to check whether the policy can accept the event. This technique is "event modification history."

In our test, the "event modification history" doubled the speed of policy searches.

5.2 ML-DNS: Dynamic Overlay Network Deployment

In ubiquitous networks, various services are

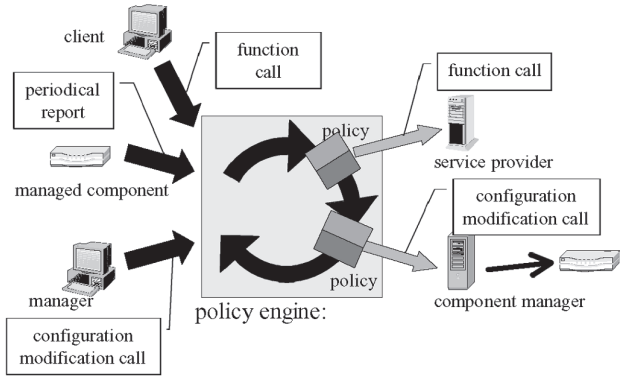

Fig. 5 Service collaboration with policy engine.

dynamically created by combining multiple service components. Because of security concerns, such as data/privacy protection, some services should be provided not on public networks but on closed networks. In such secure services, user nodes can use a service by participating in a closed network associated with the service. Thus, while the closed networks dynamically emerge/disappear, their participating nodes change on an ad hoc basis. We define a group forming such a closed network as a collaborative group. A promising approach to create a closed network is to use existing VPN (Virtual Private Network) techniques, which can deploy closed networks over public networks on an overlay basis. However, since previous approaches provide overlay networks in a preconfigured fashion[7], however, they cannot support the dynamic properties of collaborative groups.

We propose a scheme to deploy overlay networks for collaborative groups on an on-demand basis. **Figure 6** shows our system architecture. The system is composed of a private DNS (Domain Name System) server and a plug-in module installed on each member node terminal. An overlay network is provided by these two modules for each collaborative group. The private DNS server provides management functionalities for overlay networks. To deploy an overlay network for a group, all the service providers have to do is to configure the membership and domain name space of the group. Other miscellaneous configurations (e.g. IP address assignment) are generated automatically in the private DNS server. After this simple configuration, each member can ask to participate in the closed network through the login client in the plug-in module. If the login request is accepted, the private DNS server assigns to the member node an inner IP address, a private domain name and an X.509 certificate. The certificate is signed by the CA (Certificate Authority) server and is used for authentication both in establishing a control channel (TCP/SSL) with the private DNS server and in establishing IPsec tunnels with other member nodes. In the private DNS server, when a member logs in/logs out, the authentication server module dynamically updates the inner IP address for the domain name assigned to the member. The DNS server module provides a functionality to resolve the IPsec policy (notice that "IPSec policy" is not different from the "policy" in Section 5.1) in an overlay network. To achieve this functionality, the base IP address (i.e., the IP address assigned in the underlying network) is also registered as the resource record for the inner IP address through the dynamic update. The DNS proxy in the plug-in module performs IPsec policy resolution as well as normal DNS resolution for domain names associated with the group via the control channel established with the private DNS sever. Since the policy resolution is performed on an on-demand basis, the private DNS server does not have to distribute the updated IPsec policy for each membership change in the group. Each member node can communicate with other nodes by establishing IPsec tunnels based on the resolved IPsec policy. In communication with other nodes, the IPsec driver encapsulates and encrypts IP packets using the IPsec ESP mode[8]. Our

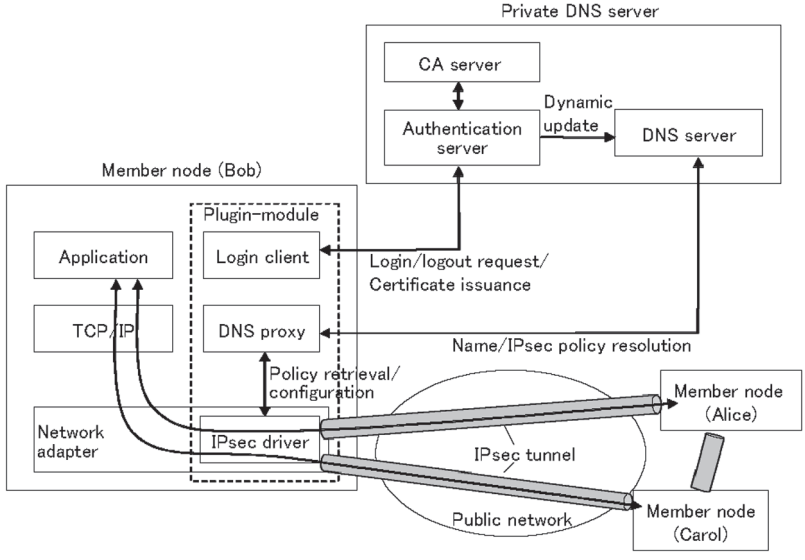

Fig. 6 Proposed system architecture.

approach thus creates a layer-3 overlay network.

To maintain an overlay topology created by IPsec tunnels, our system supports two operation modes: (i) mesh mode and (ii) graph mode, as shown in **Fig. 7**. The mesh mode is a straightforward approach, in which IPsec tunnels are directly established as virtual links between nodes. To improve system scalability, our approach creates not a fully-meshed topology but a partially-meshed topology in which IPsec tunnels are established only between nodes involved in actual communication. The on-demand-based IPsec policy resolution requires no tunnel initiation/teardown for changes in the number of participating nodes, which reduces the impact of membership changes. The mesh mode is well suited for a group either in which a small number of nodes participate or whose duration is relatively short because of its simplicity. On the contrary, for a large-scale group collaboration, in contrast, the graph mode illustrated in Fig. 7 (ii) provides good scalability because the number of tunnels maintained at each node is always independent of the number of participating nodes. In the graph mode, the graph-structured topology among member nodes is automatically created and is reconfigured for each change in the number of nodes. Packets are transferred over the graph on a hop-by-hop basis instead of over a directly established IPsec tunnel. In choosing a topology creation algorithm to be deployed, the following should be taken into account in terms: the number of hops between nodes (diameter) and the number of tunnels maintained by a node (degree). We have developed an algorithm that provides a minimized diameter with a fixed degree in Reference [9]. It also reconfigures a topology with a small number of initiated/torn-down tunnels for changes in the number of participating nodes.

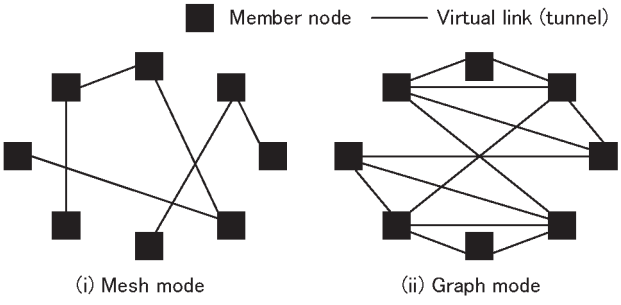

Fig. 7 Two topology operation mode topologies.

5.3 Detection and Resolution of Policy Conflicts

Without tools to check the consistency of policies, it is extremely difficult for system managers to create policies for controlling the behavior of managed distributed systems in the way that ensure reliability. We have developed a method for detecting and resolving the conflicts between policies whose actions may involve concurrent access to the same managed object, such as a router or a process running on a server. Such policies can result in an unexpected problem in a managed system.

Figure 8 shows an overview of our method. The static analysis detects all sets of policies that hold conditions below:

1) There is a shared target that appears in the action clause of the policies.
2) There are one or more actions that have side effects on the shared target.
3) The policies might possibly be executed at the same time.

These policies may conflict each other in terms of concurrent processing. Therefore, the policy engine, which interprets policies and controls the behavior of a managed object, should sequentially execute actions of the policies holding above conditions. In our method, given policies are first subjected to a static analysis. The analytical result then is sent to the policy engine, which enforces policies by referring to the analytical result. The policy engine concurrently executes actions of the policies that are not included in the same policy set in the result and sequentially

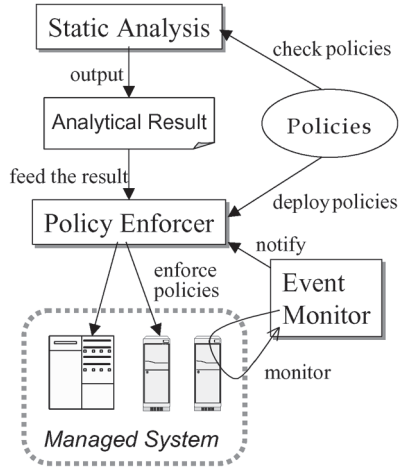

Fig. 8 Overview of our method for detecting and resolving policy conflicts.

executes the other policies.

Our method thus allows the policy engine to execute policies safely and efficiently.

6. CONCLUSION

This paper has described our service navigation architecture and shown technologies supporting it. These technologies are now under study, and we are trying to combine them to implement the service navigation architecture.

ACKNOWLEDGMENTS

This work is supported by Ministry of Public Management, Home Affairs, Posts and Telecommunications.

REFERENCES

[1] M. Weiser, "Some Computer Science Issues in Ubiquitous Computing," *Communication of the ACM*, **36**, 7, pp.74-84, July 1993.

[2] T. Shinohara, "Ubiquitous Network, Japanese-model Electronic Commerce and Challenges to Sustainable Development," OECD FORUM 2001, 2001, http://www1.oecd.org/forum2001/briefings/speeches/shinohara-01.PDF

[3] F. Leymann, "Web Services Flow Language (WSFL 1.0)," May 2001, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[4] E. Christensen, F. Curbera, et al., "Web Services Description Language (WSDL) 1.1," W3C Note 15, March 2001.

[5] TM Forum, "NGOSSTM Technology Neutral Architecture v4.0 - TMF053," Feb. 2004.

[6] Ubiquitous Forum, "Annual Report 2002," 2002.

[7] Google Press Release, "Google Achieves Search Milestone with Immediate Access to More Than 6 Billion Items," 17 Feb., 2004.

[8] D. Kosiur, "Building and Managing Virtual Private Networks," published by Wiley, ISBN 0-471-29526-4, 1998.

[9] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)," RFC2406, Nov. 1998.

[10] T. Koide, et al., "A Topology Construction Algorithm for Peer-to-peer-based Layer 2 Virtual Networks," IEICE Society Conference, Sep. 2004 (To appear).

Received July 26, 2004

* * * * * * * * * * * * * * * * *

Toshio TONOUCHI received his masters degree in information science at the Faculty of Science, University of Tokyo in 1992. He has worked in NEC Corporation since 1992. He worked on OSI management platforms and on development processes and tools for the platforms. He was a visitor at the Department of Computing, Imperial College, UK in 2000-2001, and he studied policy-based management, there. His research interest is now in applying policy-based management to ubiquitous networks and grid systems.

Norihito FUJITA was born in Tokyo, Japan, in 1973. He received his B.E. and M.E. degrees in electrical engineering from Kyoto University, Japan, in 1996 and 1998, respectively. He joined NEC Corporation in 1998 and is Assistant Manager at the System Platforms Research Laboratories, NEC Corporation, Kanagawa, Japan. His current research interests include dynamic deployment and management of virtual networks and application-aware routing control in computer networks.

Mr. Fujita is a member of the IEICE.

Naoto MAEDA received his M. Eng. degree in information science from Waseda University in 1999. He joined NEC Corporation in 1999 and worked mainly on the design and implementation of mobile agent systems from 1999 until 2002. His current research interests include the detection and resolution of policy conflicts in policy-based management, design and implementation of static rule checkers for Java programs and software verifications.

Tomohiro IGAKURA received his M. Eng. degree in electronics engineering from Tokyo University in 1999. He joined NEC Corporation in 1999 and until 2002 worked mainly on the research of network management from 1999 to 2002. His current research interests include policy-based management architecture and detection of wrong (against intention) policies.

Yoshiaki KIRIHA received his B.E. and M.E. degrees in electronic communication engineering from Waseda University in 1985 and 1987, respectively. He joined NEC Corporation in 1987, and is now a senior manager of the System Platforms Research Laboratories. He has been engaged in the research and development of network management systems, realtime database systems, distributed computing systems, active networking systems, and ubiquitous networking systems.