1998 International Computer Symposium
Workshop on Computer Networks, Internet, and Multimedia
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

A JAVA-IMPLEMENTED MESSAGE-ROUTING FRAMEWORK FOR WORLD WIDE MEET

Chien-Tsun Chen
Application Development Division
CanThink Co.
Taipei, Taiwan
Email : ctchen@can.com.tw

Jung-Sing Jwo
Department of Computer & Information Sciences
Tunghai University
Taichung, Taiwan
Email : jwo@s867.thu.edu.tw

ABSTRACT

We have introduced an idea called World Wide Meet (WWM) to support horizontal interaction ability for the netizens who are surfing on the webs [4]. With WWM support, users browsing the same web page can be notified automatically when anyone joins or leaves the page. Each page of a web is just like a virtual room and the netizens in the same room can interact with each other. In this paper we propose a generic message-routing framework to support the functionality required by WWM. We further implement the framework in Java technology. By using this Java-implemented framework, WWM can be seamlessly integrated with the webs without any additional effort. This framework can be further applied to various web applications including electronic commerce, and distance learning, etc.

1. INTRODUCTION

After *World Wide Web* (*WWW*), or simply *web*, has been introduced to the Internet world in 1989 [2], the Internet users (or called *netizens*) are increased exponentially year by year. Today, at least seventy-one millions people are said to have access to the Internet [7]. Undoubtedly, its hypermedia structure and the ability to support easy access of the multimedia information services are the two key issues that make the webs so popular. It becomes a kind of new media such that it can provide not only the traditional information services, but also can be treated as a platform for electronic commerce, and distance learning, etc [1, 12].

Usually, a web site provides its own special services and attracts the netizens to visit the site as frequently as possible. It can be said that the netizens surfing on the same site are belonging to the same *virtual community*. They may not meet each other physically, however they do interact with each other virtually through sharing the services provided by the same site. The way the netizens share the service in fact, as shown in Figure 1.1, is a *vertical relationship* between netizens and the webs. In other words, *horizontal interaction* among the netizens is not directly supported by the WWW architecture. As an example, considering a web site as a building and each page in this site as a room, then a netizen who is browsing in this site actually feels that he is completely alone no matter how many other netizens are browsing the same page at the same time with him. He can not know the appearance and vanish of the netizens in the same room. He can not interact with them, either. The lack of the horizontal interaction support in WWW makes the virtual communities not so realistic.

In order to overcome the drawback of the web mentioned above, we introduced an idea called World Wide Meet (WWM) in [4]. It is a solution that can integrate the horizontal interaction ability into WWW and it lets netizens *browse the web not alone*. Moreover, in order to keep the popularity of WWW, this solution should not replace the way that the current web servers and browsers work. Virtual place [13], sociable web [6], and Palfreyman's solution given in [10] are the early works related to this goal. However, these systems require specific browser technology. Since the architecture of WWW has been considered as the de-facto architecture for Internet services by most of the netizens, it is not convincing and not wise to obtain new functionality by just switching to other systems.

With the support of WWM, each page of a web site is treated as a WWM *room*. A netizen surfing from one page to another through hyperlink is considered as that he leaves the first WWM room and enter into the second WWM room. Inside each WWM room, the netizen can act exactly same as in a real room, i.e. he can either say hello to a new comer or chat with his old friends who are just bumping into the room.

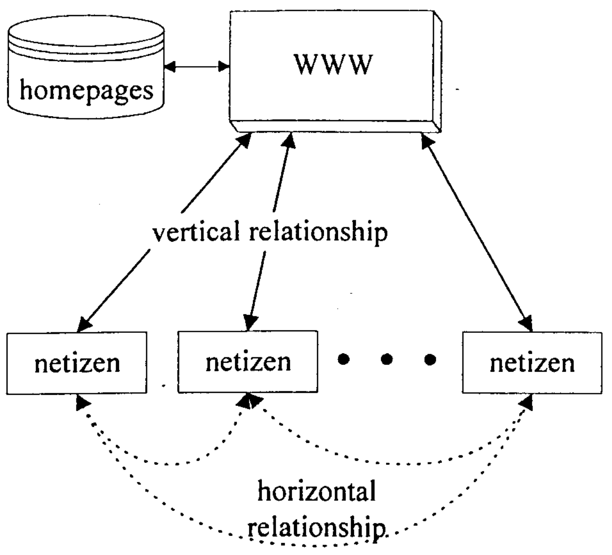


Figure 1.1. Vertical and horizontal relationships among netizens and WWW.

The major functionality that WWM should support includes at least the following items.

(1) A WWM room is automatically integrated into each web page when WWM is installed. As soon as a netizen browsing the page, he also enters the WWM room no matter what kind of browser is used. That is, WWM can be seamlessly integrated into WWW environment.

(2) A WWM room should not be related to the design of a web page. In other words, WWM can be installed or removed easily without affecting the existing web site.

(3) A WWM room should provide mechanisms for interaction. It can be either a VRML-based virtual reality scene, or multimedia-enriched communication tools. To netizens, WWM needs to provide (a) identification service, (b) messages of the appearance and vanish of the other netizens, (c) chatting service, (d) privately talking, and (e) message posting. WWM management services are also required for web masters.

(4) Tracking information of the netizens visiting the WWM rooms needs to be collected and analyzed to help web master construct a better virtual community.

With WWM support, users browsing the same page can be notified automatically when anyone joins or leaves the page. Each page of the web is just like a virtual room and the netizens in the same room can communicate with each other.

In this paper, we propose an object-oriented designed framework to fulfill the requirement of WWM. In fact, this framework can be considered as a *message-routing middleware*. Message-routing basically is a network service such that it handles all the messages, including those from users (or services) to other users (or services), by having them processed and dispatched to the corresponding destinations. Web applications, such as Intranet systems, electronic commerce and distance learning that require exchanging messages among clients or services, can be benefited from this architecture.

The rest of this paper is organized as follows. In Section 2 a generic framework for message-routing service is presented. Section 3 introduces the components related to the design of this framework. Section 4 discusses the implementation issues of the WWM. Results and conclusion remarks are given in Section 5.

2. A GENERIC FRAMEWORK FOR MESSAGE ROUTING

Figure 2.1 is the framework we propose to support message-routing in a web environment. This framework utilizes the concept of component model and object-oriented design patterns [3, 5, 8, 9, 11].

For the message routed in this framework, as shown in Figure 2.2, in addition to the data body itself, each message should contain the header information about its destination and required service. Messages from the clients are first accepted by the *Acceptor* [11]. Acceptor quickly transfers each of the messages to a free *Handler* which is selected from *HandlerPool*. We use HandlerPool to guarantee the safety for the usage of the system resources. Each Handler has two message boxes. One is *In-message box* and the other is *Out-message box*. Out-message boxes are further divided into two categories, i.e. for service and for client. Handler first adds its message into *ExternalMessageQueue*. *ExternalMessageDispatcher* checks the messages with the *GroupTable* and *ServiceRegister*. It then routes the messages in the queue to either Out-message boxes for service or Out-message boxes for client respectively. If a message is routed to a service's Out-message box, it will be processed first by the corresponding service and then new messages are added to the *InternalMessageQueue*. *InternalMessageDispatcher* is responsible for distributing the messages inside InternalMessageQueue to their final destinations.

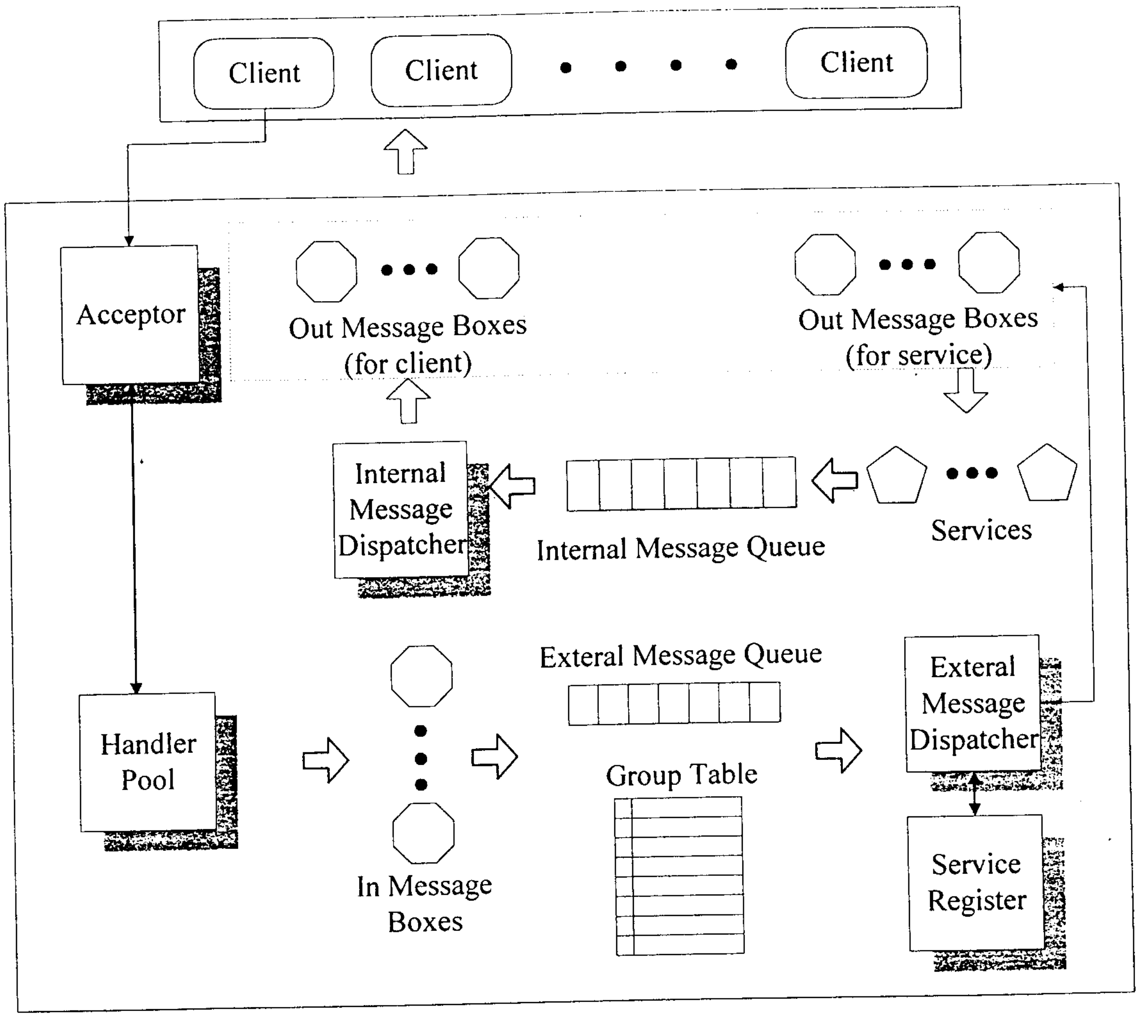


Figure 2.1. The generic framework for message-routing.

The message-routing framework described above obviously can be applied to WWM directly. Furthermore, by adding proper services, this architecture can do more than just routing message among clients. For example, we can add a broadcasting service to the middleware. Then, this framework can help a client broadcast its message to all the other clients. As another example, in electronic commerce an EDI service can be added into the framework. An E-form is processed and transferred into another format before it is routed to the destination.

3. COMPONENTS OF THE FRAMEWORK

In this section, we further discuss each of the major components of the generic message-routing framework introduced in the previous section.

(1) Acceptor :

The purpose of the Acceptor pattern is to listen on a TCP/IP port and to accept connection requested from a client. Acceptor then selects a free Handler from the HandlerPool to represent the client. If there is no available Handler, the connection request will be queued for the next free one. By inspecting the message header of the client, Acceptor adds the Handler into its corresponding GroupTable. A smart Acceptor can provide better system performance by asking HandlerPool to monitor and to limit the number of the Handlers. When special connection service is required, a filter can be added into the Acceptor and it can increase the flexibility of the system.

(2) HandlerPool :

The responsibility of HandlerPool is to create and manage the Handlers prepared for the Acceptors. Usually, in a frequent service-requesting environment, it is not efficient to create Handler for each request in realtime.

Therefore, HandlerPool automatically generates various Handlers as soon as the system is started. Since these Handlers are ready for use, the connection requests accepted by the Acceptors can be served without any delay. HandlerPool will dynamically increase or decrease the number of the Handlers by monitoring the resource usage and the number of the concurrent clients.

Header	***Body***
Service type	Protocol of the service
Service version	
Source	
Destination	
senderID	
receiverID	
Message encoding	
Time stamp	

Figure 2.2. Structure of a message

(3) Handler and In(Out)-message box :

Each handler has one In-message box and one Out-message box. All the messages issued by the client are first stored in the In-message box. Handler then transfers the messages to the ExternalMessageQueue. All the messages that need to be routed to the clients are first sent to the Out-message box. Out-message box then will distribute the messages to their corresponding destinations. Basically, In-message box and Out-message box are implemented as two concurrent threads. In-message box can filter out the abnormal messages, i.e. those messages that framework can not identify. On the other hand, by delaying non-emergent messages or destroying out-of-date messages, Out-message box has the ability to control the flow of the messages.

(4) External(Internal)MessageQueue :

ExternalMessageQueue is the bridge between In-message box and ExternalMessageDispatcher while InternalMessageQueue is between Services and InternalMessageDispatcher. The purpose of these queues is to decouple the dependency between the message boxes and the dispatchers or the services. This design can reduce the implementation complexity and increase the reusability of the components. External(Internal)MessageQueue can be implemented as a priority queue so that the framework can dynamically schedule each message's sending time.

(5) External(Internal)MessageDispatcher :

ExternalMessageDispatcher first interprets the message's header information. It then decides the Service that the message should be served by checking the ServiceRegister and routes the message to the Service's Out-message box. If no one is available, a default service is selected or the message is discarded. For InternalMessageDispatcher, the implementation is same as ExternalMessageDispatcher. However, after interpreting the header information of the message, InternalMessageDispatcher will distribute the message into the corresponding client's or Service's Out-message box.

(6) ServiceRegister :

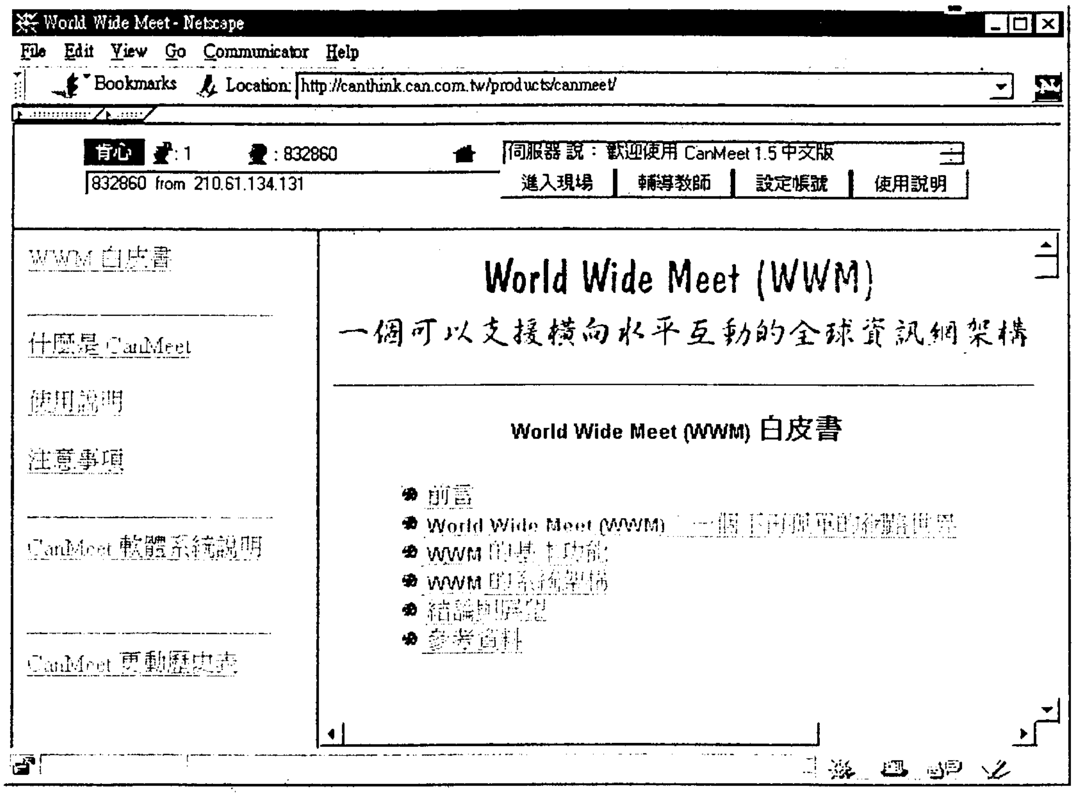
ServiceRegister is responsible for the management of available services. It can be implemented as a *prototype manager* pattern [8]. Services can be added into or removed from the ServiceRegister dynamically.

(7) Service :

Service is the component that really performs the requesting service. Basically, each Service implements its own service protocol. Service first takes a message from its corresponding Out-message box and then processes it. The output result is encapsulated as a new message and the message is sent to the InternalMessageQueue. In order to make a Service plugable to the framework, a standard interface for designing Service should be given.

4. IMPLEMENTATION ISSUES

By following the discussion of the previous section, we implement a message-routing system WWM by using Java technology. The reason for choosing Java is considering the portability and intrinsic multi-thread support. In this experimental version of WWM, we further add some more services. These services include chatting Service, concurrent netizens counting Service, and posting Service, etc. The implementation of the system is quite obvious. However, it is not so trivial for the client. Recall the functionality of WWM introduced in Section 1. Since we don't want to increase the burden, such as plug-in installation, of the netizens, it seems that Java applet is


World Wide Meet - Netscape

File Edit View Go Communicator Help

Bookmarks Location: http://canthink.can.com.tw/products/canmeet/

肯心 :1 : 832860

832860 from 210.61.134.131

伺服器說：歡迎使用 CanMeet 1.5 中文版

進入現場 | 輔導教師 | 設定帳號 | 使用說明

WWM 白皮書

什麼是 CanMeet

使用說明

注意事項

CanMeet 軟體系統說明

CanMeet 更動歷史表

World Wide Meet (WWM)

一個可以支援橫向水平互動的全球資訊網架構

World Wide Meet (WWM) 白皮書

- 前言
- World Wide Meet (WWM) 一個[illegible]的[illegible]世界
- WWM 的基本功能
- WWM 的系統架構
- 結論與展望
- 參考資料

Figure 4.1. An example of WWM implemented by CanThink Co.

the only choice for the client implementation.

Usually, a homepage designer will tell you that the total memory size for a good homepage design should not be more than 50 to 80 Kilobytes. The major concern of this design criteria is the bandwidth of the Internet service. If the contents of a page are about 60 Kilobytes, then the size of the applet client for WWM can not be more than 20 Kilobytes. This constraint is quite difficult to be satisfied if there are many services supported in the WWM. Fortunately, we can use network dynamic loading technique to overcome this problem. By subdividing the applet into several objects, these Java classes can be dynamically loaded through Internet only when they are needed. Size of each object in our WWM client is about 10 Kilobytes without compression. The size of the initial loading applet is also about 10 Kilobytes. Thus, our WWM solution will not consume the bandwidth too much.

Another issue about WWM client is from the design view of the homepage. Since the current user interface of the browser is based on window's concept, how to add the client applet into the page without affecting the original homepage design is not a simple problem. We can not answer this question here, however we do implement our solution as shown in Figure 4.1. In this implementation, we try to minimize the visual area of the WWM client. For example, if a netizen wants to chat with the others, she or he needs to enter the chat room inside an independent window which is invoked by pushing the chat button. The URL of our experimental WWM system is given in **http://canthink.can.com.tw/**.

5. CONCLUSION REMARKS

In this paper, we have proposed a framework for message routing. By applying this architecture, we further implement WWM related services with Java technology. With the support of WWM, each page of a web becomes a virtual room. Netizens surfing on a WWM enhanced WWW site can never feel alone. Currently, WWM is commercially available and has been applied to the fields of electronic commerce and distance learning. The success of these applications shows the superiority of the framework introduced in this paper.

REFERENCS

[1] Bently, R., T. Horstmann, K. Sikkel, and J. Trevor, "Supporting Collaborative Information Sharing with the World Wide Web : the BSCW Shared Workspace System," Proceedings of the 4th International WWW

conference: Boston, MS, USA, 1995.

[2] Bemers-Lee, T., R. Cailliau, A. Luotonen, H.F. Nielsen, and A. Secret, "The World Wide Web, " Communications of the ACM, 37(8), 1994, pp. 87-96.

[3] Brockschmidt, Kraig, "Inside OLE2", Microsoft Press Programming Serie, 1993.

[4] Chen, T. and J. Jwo, "World Wide Meet – Browsing the Web Not Alone", *International Conference on Multimedia & Telecommunications Management*, Hong Kong, 1998, Accepted.

[5] Davis, T. E., "Build your own ObjectPool in Java," http://www.javaworld.com/javaworld/jw-06-1998/jw-06-object-pool.html.

[6] Donath, J.S., and N. Robertson, "The Social Web," Proceedings of the 2nd WWW conference: Mosaic and Web, http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/donath/SocialWeb.html/; 1994.

[7] Fox, R., "News Track", Communications of the ACM, Vol. 40, No. 8, 1997, p. 9.

[8] Gamma, E., et al., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, 1994.

[9] Lavender, R. G. and D.C. Schmidt, "Active Object: An Object Behavioral Pattern for Concurrent Programming," Pattern Language of Program Design 2, Addison Wesley, 1996, pp. 483-499.

[10] Palfreyman, K., and T. Rodden, "A Protocol for User Awareness on the World Wide Web," ACM 1996 Conference on Computer Supported Cooperative Work, pp 130-139.

[11] Schmidt, D. C., "Acceptor and Connector," Pattern Language of Program Design 3, Addison Wesley,1998, pp. 191-229.

[12] Slaone, A., "Learning With the Web: Experience of Using the World Wide Web in a Learning Environment," *Computers Education*, Vol. 28, No.4, 1997, pp. 207-212.

[13] Virtual places, http://www.vplaces.com/vpnet/index.html.