HTTP-based Protocol for Internet Calendaring and Scheduling

by
Wingkong (Oliver) Yip

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering and Computer Science
Master of Engineering in Electrical Engineering and Computer Science

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1996

Author...
Department of Electrical Engineering and Computer Science
August 26, 1996

Certified by...
Dr. Jim Miller
Research Scientist, MIT Laboratory for Computer Science,
World Wide Web Consortium
Thesis Supervisor

Certified by...
Vinod Seraphin
Principal Engineer, Lotus Development Corporation
Thesis Supervisor

Accepted by...
rick R. Morgenthaler
n Graduate Students

HTTP-based Protocol for Internet Calendaring and Scheduling

by
Wingkong (Oliver) Yip

Submitted to the Department of Electrical Engineering and Computer Science
on August 26, 1996
in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science
Master of Engineering in Electrical Engineering and Computer Science

Abstract

An HTTP-based internet calendaring and scheduling (C&S) protocol, which can be used with standard Web browsers and Web servers, is designed and implemented. A C&S system with a Lotus Organizer backend has been implemented to verify the viability of such a protocol. HTML response templates and C&S ActiveX controls have been used to keep the protocol simple and efficient. A session manager has been developed to keep the state of C&S activities in the HTTP-based protocol. It is demonstrated that with the help from these components, problems encountered in designing such a protocol, such as heavy network traffic, lack of session support and limitations in user interface developments, can be solved.

Thesis Supervisor:	Dr. Jim Miller
Title:	Research Scientist, MIT Laboratory for Computer Science, World Wide Web Consortium
Thesis Supervisor:	Vinod Seraphin
Title:	Principal Engineer, Lotus Development Corporation

Acknowledgements

First of all, I would like to dedicate this thesis to my parents and my brother, Angus. Their support has been unceasing during my four years at MIT. Though they may not understand the technical content in this thesis, this dedication is to show my appreciation for their love and concern while we are miles away.

I am truly indebted to my thesis supervisor at Lotus, Vinod Seraphin. Without his vision and support, this thesis project cannot be made possible. He has also spent much time on figuring out the thesis topic, guiding me through the project, and reviewing the drafts of this thesis report. I would also like to thank my MIT thesis supervisor, Dr. Jim Miller, for his time, insights and guidance which all helped me to learn and to conduct the research.

Next, I would like to extend my gratitude to Candy Sidner and Professor John Guttag in the VI-A program, and Arvind Goyal in Organizer Group. Candy and Arvind helped me a lot in arranging the financial support for this thesis, and John has given me valuable advice on looking for a thesis and a faculty thesis supervisor.

The folks at Organizer Group were always helpful during my research there. Thanks Bruce, Jennifer, Vlad, Gary, Leo, Tom and Elie for their time and help while I got stuck and frustrated in debugging my prototype.

My sincere thanks also go to those who have been experiencing, learning, and sharing life with me at MIT. Thanks Felix, Andy, Patrick, Anita, Leo, Albert, Jenny, Vinci and Chris for their encouragement and support while we were taking classes together, working as lab partners and enjoying our lives outside classrooms. I would also like to thank Ernie, Khon, Jacky, Winnie, Ada, Kpang, June and Kelly for their concern and company during the days when I was working on this thesis project, and for having dinner with me and making me laugh after work.

Last but not least, thanks Lord Jesus Christ for His unfailing support and guidance through the thesis project and my years at MIT.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background

People use Personal Information Managers (PIMs) to store personal and group information, and to schedule meetings with each other. Traditionally users access PIM information stored on local machines or on file servers from their own workstations. This presently requires PIM users to always carry their PIM applications with them on their machines.

As the internet and the World Wide Web (WWW) technologies gain in popularity[1], the internet provides a new and open environment for storing and sharing calendaring and scheduling (C&S) information. With its intuitive and consistent interface, flexibility and linking capabilities, the Web provides an ideal application interface for C&S activities. The following benefits can be perceived in an internet C&S system:

- C&S information may be accessed through the Web, with any Web browsers running on any workstation and platform, from anywhere in the world.
- It's not necessary to always carry around a special C&S application to view personal calendar information or to set up group meetings. C&S functions can be

[1] The latest internet statistics survey [1] shows that the total number of internet hosts has reached 9.5 million by January 1996, nearly doubling the number in January 1995 when there were 4.9 million hosts. It is projected the number will reach 100 million in 1999. It is estimated that 30 to 40 million people have internet access now.

centralized on a Web server, instead of requiring individual users to have the C&S application installed on their workstations.

- Facilitate the publication of certain Organizer information to other people outside a LAN or WAN for more convenient information exchange. For example, the free time information, which indicates when an individual is free and busy, can be published on the Web and people from other corporations can access them easily.
- Make public scheduling possible, which allows the public to request the booking of available time slots of a professional across the internet [2]. For example, doctors or lawyers may be requested by their patients or clients to schedule a time with them over the Web. People could also directly book the times for resources across the net without any mediators. For example, the scheduling of using a tennis court or a conference room can be done with this idea.

Obviously internet calendaring and scheduling is a very useful internet application in real life, and it is desirable to many people.

1.2 Goals of Thesis

This thesis explores the various ideas and issues on how calendaring and scheduling can be done on the internet. A protocol for calendaring and group scheduling will be designed and implemented. This protocol will be based on the HyperText Transfer Protocol (HTTP), which is the native protocol of the Web.

In designing such a protocol, the following three problems can be perceived:

1. C&S applications require frequent interaction with calendar databases. As access of databases on the internet through the network is slow, the performance of internet calendaring and scheduling may be poor. How can an efficient HTTP-based protocol be designed, which is specialized for calendaring and scheduling, to minimize the traffic between the client and the server, such that reasonable speed and performance of the protocol can be ensured?
2. HTTP is known to be a "stateless" protocol [3]. The connection between the client and the server is terminated right after each response from the server. Information such as user name and password are not kept from one Web page to another, and general interactivity with a Web site involved in calendaring and scheduling can be difficult to achieve because of this "lack of state" property in HTTP. How can the HTTP-based protocol be designed to support "session-like" activities for calendaring and scheduling?
3. Traditionally C&S applications have very rich user interfaces, which exploit the native graphical operating systems such as Microsoft Windows. On the other hand, Web pages in internet applications are written in HTML, which is pretty limited in giving interactive and graphical intensive interfaces to the Web pages. This HTML limitation can greatly affect how the HTTP-based C&S protocol can be used, and can complicate the protocol implementation. How can we enhance the C&S user interface in HTML pages, so that the protocol can be kept simple and efficient to use?

In this thesis, solutions will be sought to solve these problems as the HTTP-based C&S protocol is developed. We can generalize the solutions and apply the ideas to other internet applications which have similar problems, i.e. internet applications that require frequent connections with Web sites for information retrieval, that need to keep state on the Web site while using the stateless HTTP, and that involve intensive graphical interaction with users.

1.3 Overview

The overall design of the internet calendaring and scheduling system built and used in this thesis is presented in Chapter 2. A big picture of the system architecture, the client-side user interface as well as the server processes are described there.

The design goals and design details of the HTTP-based C&S protocol are discussed in Chapter 3. Design details such as the use of response templates are explained. Chapter 4 explores the various issues in the protocol design for internet scheduling in more detail. Different aspects of an internet scheduling service and how to design a protocol to accommodate them are discussed. Chapter 5 defines the HTTP-based C&S protocol and talks about the protocol implementation details. Examples of how to use the C&S protocol are also included.

The use of the session manager process on a HTTP server to solve the problems caused by the "stateless" properties of HTTP is discussed in Chapter 6. The design and implementation details, such as the communication scheme between the session manager and the server CGI process, are presented.

Finally, the development and use of various ActiveX components in HTML in the internet C&S system to enhance the UI elements in HTML are shown in Chapter 7. Choices of technologies in the marketplace to develop the components, choices of methods in building the components, and how to use the components in the internet C&S applications are fully discussed in this chapter.

Chapter 2 The Base Framework

2.1 Overview

As part of designing a protocol for internet calendaring and scheduling, a prototype that uses this protocol was implemented. This was done to verify the viability of such a protocol, evaluate its performance and explore potential implementation issues. This prototype of an internet C&S system will be based on the design and features of Lotus Organizer, an award-winning C&S application in today's market. This base framework for the protocol is a "Web version of the Organizer application", and shall be referred to as the OrgWeb system hereafter.

A subset of the calendaring and scheduling features in Organizer are implemented in the OrgWeb system. The backend databases used to store C&S information are Organizer 3.0 files (.OR3 files) or Lotus Notes mail files (.NSF files). Moreover, the actual interactions with these databases are made through an Organizer API layer, in which database readings and writings can be made by means of API calls.

What needs to be constructed now is an interface between a Web client and the Organizer API layer on a Web server. This involves one or more processes on the Web server that can understand the OrgWeb C&S protocol, take requests from the Web clients who utilize the C&S protocol, access the Organizer databases through the Organizer API layer, and finally serves out calendaring and scheduling information to the clients. Section 2.2 will show the big picture of the base framework and how different components are connected in the system. Section 2.3 and 2.4 will describe the design and components on the client side and the server side respectively.

2.2 System Architecture

The architecture of the OrgWeb system is illustrated in Figure 2.1.

The system starts with the Web browsers on the right hand side of the figure. End users, who desire to access C&S functionality, access the OrgWeb system through standard Web browsers which communicate with Web sites over the internet (or intranet) via HTTP. In the OrgWeb system the clients need to talk to a Web site that has installed the OrgWeb components which can access the users' Organizer database and serve C&S information to the clients. Thanks to the dynamics of the Web, OrgWeb clients may be anywhere in the world, using any kind of platform and operating system, to access the OrgWeb Web server.

In the current implementation of the OrgWeb system, a Netscape Communication Server on a Windows NT 3.51 workstation and an Internet Information Server on a Windows NT 4.0 (Beta 2) server were used as the Web servers. Since the base protocol is still HTTP, the standard protocol spoken by all Web browsers and servers, no additional software is required on the Web client side[2]. On the Web server side, a CGI program was written to communicate with Web clients using the HTTP-based C&S protocol. In addition, a separate process was implemented on the server to support session

[2] When the user interface of the OrgWeb system is enhanced (See Chapter 7) Organizer-specific ActiveX controls and Organizer DLLs required to execute the ActiveX controls should be downloaded from the server side and installed on the client side.

management for the C&S CGI program. This process accesses the Organizer databases through the Organizer API layer. These two processes communicate with each other by means of named pipes in Windows NT, which will be discussed in more details in Section 6.3.2.

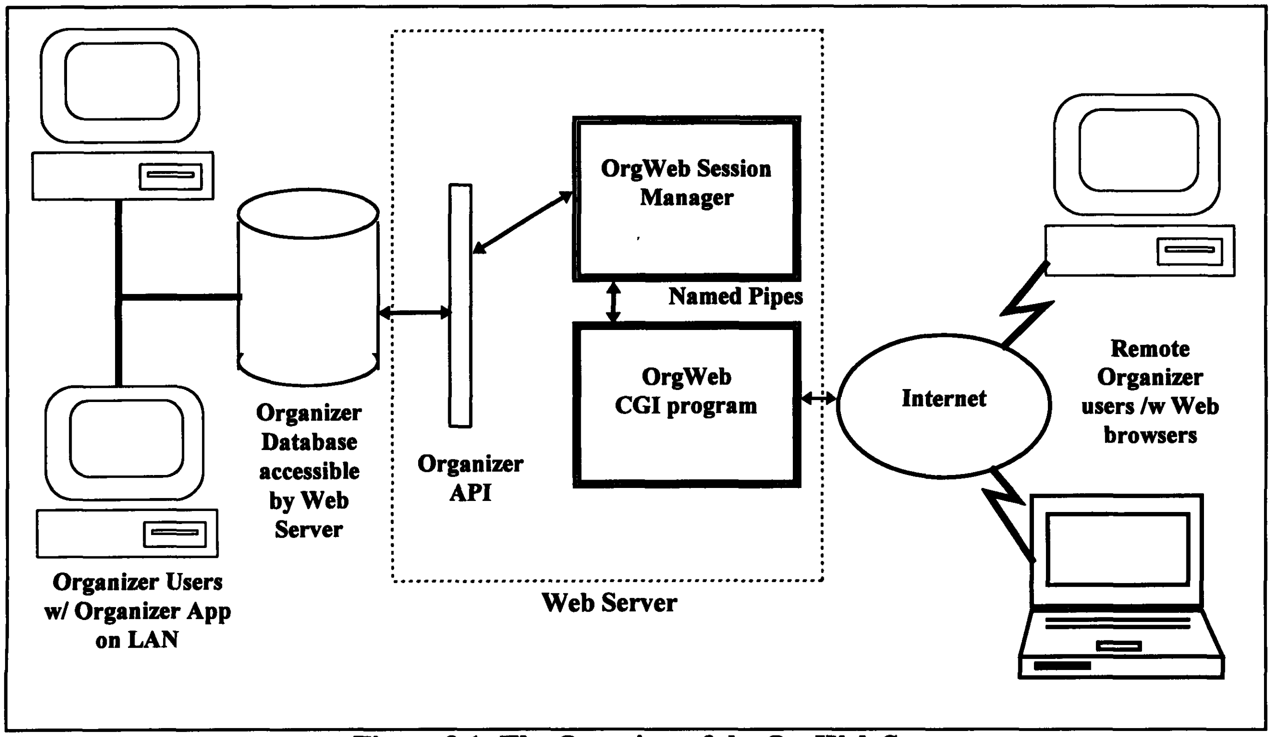

OrgWeb Session Manager
Named Pipes
OrgWeb CGI program
Organizer Database accessible by Web Server
Organizer API
Web Server
Internet
Remote Organizer users /w Web browsers
Organizer Users w/ Organizer App on LAN

Figure 2.1: The Overview of the OrgWeb System

Whenever a Web client makes a request in the OrgWeb HTTP-based C&S protocol, the CGI program at the server is invoked. This CGI process then talks to the session manager process, which keeps the information for different C&S sessions, authenticates the client requests, accesses the Organizer databases through the Organizer API layer, and finally returns the C&S information requested back to the CGI process. The CGI process then passes the results back to the Web client in the form of standard HTML responses, which can be readily displayed by the Web browsers on the client side.

In the Organizer API layer, a number of C procedures are provided to access the information in Organizer database files. All Organizer files that are accessible by the Web server through the API layer comprise the backend databases for the OrgWeb system.

2.3 Client Side

2.3.1 User Interface

The user interface of the OrgWeb system is in the form of HTML pages within a Web browser. When a user wants to use the C&S service, he or she first goes to a logon page at an OrgWeb Web server (See Appendix A.1). Then the user fills in the logon form with all the login information required, and submits it to login a C&S session.

After logging in, the week view (See Appendix A.2) of appointments for the current week is displayed by default. There are links to go to a month view of the appointments in the current month (See Appendix A.3), and to a day view (See Appendix A.4) of today's appointments. On every date of the appointment view, there is a link to a "Create Appointment" page (See Appendix A.5) which allows the user to create a new appointment on the date associated with that link. The appointment attributes may be set in a form on that page and submitted to create an appointment in the database. For creating a group meeting, a user can follow a link on this "Create Appointment" page to a "Create Meeting" page. (See Appendix A.9) There the user can set the meeting attributes and the attendees information on another form, and submit it to create a new meeting. After submission of these forms a possibly updated calendar view will be displayed.

On every appointment or meeting entry in the calendar view, there is a link to an "Edit Appointment" (See Appendix A.6) or "Edit Meeting" page for that appointment or meeting. The form element values are preset with the current attributes of that appointment or meeting. The user may change the values in these form elements to change the attributes. The user can submit the form to update the appointment or meeting. There is also a link on the "Edit Appointment" and "Edit Meeting" page which facilitates deleting the appointment or meeting from the database. After updating or deleting the appointment or meeting, the user will return to the preceding calendar view.

For group scheduling in the OrgWeb system, different users make meeting invitations and replies by means of exchanging meeting notices through Notes Mail or cc:Mail. These meeting notices are stored in a "mailbox" in the users' databases. To view the meeting notices in the mailbox, a user can follow a link on any calendar view to a "Meeting Notice" page (See Appendix A.7). A short description is displayed for every meeting notice on this page. On this short description there is another link, which leads to the "Meeting Notice Details" page (See Appendix A.8) for that particular meeting notice. The users can view all the important attributes and attendee status for that meeting on this page, and additional links are also displayed there for the user to process the meeting notice. When the user follows one of these links the meeting notice will be processed, hence possibly making changes in the user's database and sending out more meeting notices. The user returns to the "Meeting Notice" page afterwards, where the user can view and process other meeting notices, or simply go back to the calendar view.

When a user is done using the C&S service, the user may logout. Links to logout a C&S session appear on all Web pages throughout the C&S session.

Examples of the Web pages seen in a C&S session can be found in Appendix A.

2.3.2 Calendar View Navigation

The ability to easily navigate and read calendar information in different views and on different dates is very important in a C&S application.The user interface is designed to have three kinds of calendar views for appointments: day view, week view and month view. On every page of a particular calendar view in the system, there are links to go to the other two calendar views for the current date.

Moreover, for a page in a particular calendar view, there are links to go to the pages for the previous and next day, week or month, depending on which calendar view the user is currently in.

Figure 2.2 illustrates the flow of the Web pages to view the appointment information in different calendar views.

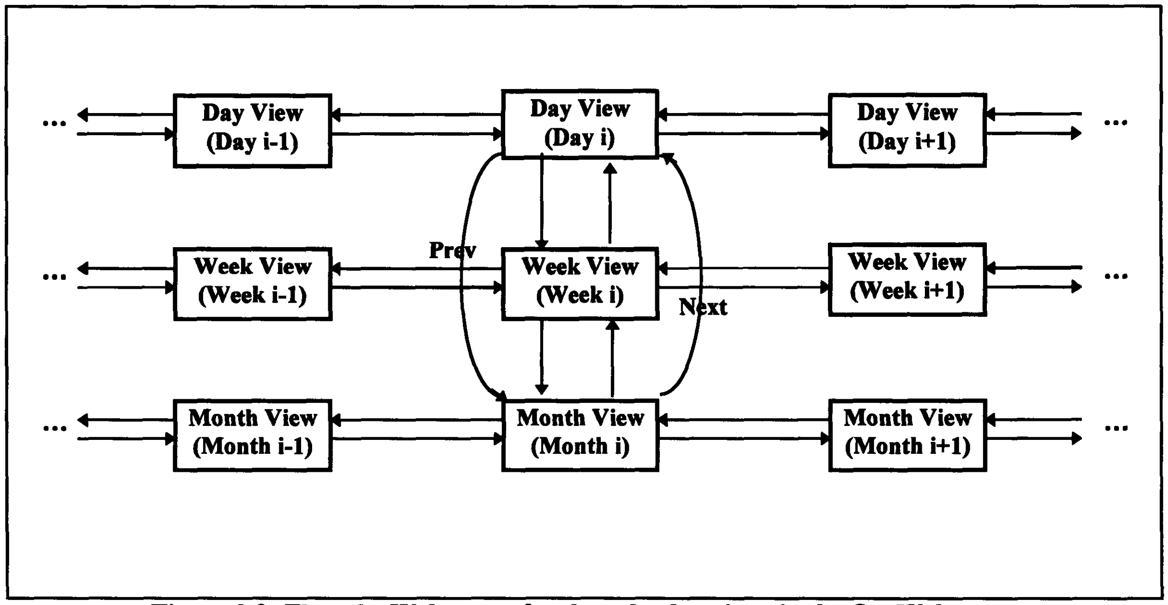


Figure 2.2: Flow the Web pages for the calendar views in the OrgWeb system

2.4 Server Side

2.4.1 The CGI Process

The CGI process on an OrgWeb server receives requests from the Web clients with the C&S protocol. There is a request parser in the CGI process, which takes the OrgWeb requests from the environment variables (via an HTTP "GET" request) and from the standard input of the CGI process (via an HTTP "POST" request) [4][5]. It creates a request object, in which the parameters and the environment variables of the request are stored.

The request object is then passed to a command parser, which parses the request strings into a list of command objects. In each command object a request command and the associated arguments can be found. This command object is then stored in the request object as well.

Finally the request object is passed to a command processor. The command processor takes the list of command objects from the request object, and executes the commands in the list sequentially. After all the commands in the list have been executed, the results are passed to the standard output of the CGI process as an HTML document, which will be displayed in the Web browser on the client side.

2.4.2 The Session Manager

The session manager acts as a mediator between the CGI process and the Organizer API layer. It is always running on the C&S Web server. When a CGI process is started by a C&S request, it will talk to the session manager for request authentication and session information retrievals. The session manager will actually perform the C&S operations by accessing the Organizer databases through the Organizer API layer. After successful database operations, the responses will be transmitted to the CGI process, which will pass the responses to the Web clients.

The design and implementation details of the session manager will be fully discussed in Chapter 6.

Chapter 3 Basic Protocol Design

In the internet calendaring and scheduling base framework, a Web client talks to a Web server to ask for C&S information. The way in which the two parties communicate is the internet C&S protocol. The basic design of this protocol will be discussed in this chapter.

3.1 Design Goals

In designing the internet C&S protocol, there are a number of design goals and requirements.

3.1.1 Ready and Easy to Use in Existing Environment

The first requirement for the protocol is that it can be readily used in the existing environment. This can reduce the costs of using the protocol, as few changes need to be made for the end-users and the information providers.

The World Wide Web is presently the most popular way of accessing information on the internet [4]. The Web's open and flexible way of accessing and publishing information has led to its explosive growth [1]. Having identified the Web as the prevalent way to use the internet, a key design goal is to make the C&S protocol ready and easy to use with the Web. Ideally, end-users should be able to use the protocol to access C&S information with their Web browsers, and the information providers should be able to use the protocol to provide the information on a standard Web server. A very important design goal shall be minimizing the changes that need to be made in the existing environment.

3.1.2 Minimize Traffic between the Client and the Server

C&S applications usually require frequent interactions with calendar databases. When the users "turn pages" in their calendar view and read their appointment records, a lot of appointment records need to be read from the databases. If only a single appointment record or a day of appointment records can be accessed in each request, a lot of separate requests have to be made when the user tries to read his appointments for a particular month.

An inefficient way to access a large amount of appointment information in the protocol can greatly affect the performance of the internet C&S system. As a result, the protocol must provide a flexible way for the client to get calendar information, so the client can access a day, a week, or even a month of appointment information with a single request. This minimizes the overhead required in making requests in the protocol.

3.1.3 Support Consistent Behaviors between Requests

In accessing calendar databases, users typically need to identify themselves (e.g. with a username and a password) before they can gain access to the information within the databases. If they are not listed within the access control list (ACL) of a database with the proper level of access, their requests will be rejected.

In C&S activities, users are likely to access the same databases from one request to another. For example, when a user reads his calendar information, he is making different requests from page to page on the same database. It is inefficient to supply a password to identify himself when he makes a request every time before he can actually access the database. As a result, the protocol should support *consistent behavior* between different requests. In other word, the protocol should support the concept of a "session", in which a user identifies himself at the beginning of the session, and doesn't need to identify himself again throughout the session.

3.1.4 Flexibility and Scalability

Last but not least, flexibility and scalability are required in the protocol. Making changes to the existing set of commands in the protocol should be easy. Also it should be possible to add and delete commands in the protocol without much difficulty. This helps maintain the C&S system in the long run.

The issue of the ready and easy use of the protocol in existing environments leads to the idea of developing this new protocol based on the HTTP protocol. The issues of minimizing network traffic, and promoting protocol flexibility and scalability will be reviewed while the design of the protocol is discussed in this chapter. The use of HTTP as the underlying protocol and the requirement of consistent behavior leads to the development of the session ID parameter in the protocol, and also the development of the session management process, which will be discussed in Section 6.

3.2 Basic Design

3.2.1 HTTP-based

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems [6]. It is a generic, stateless and object-oriented protocol which can be used for many tasks. It has been in use as the native protocol for the Web since 1990. All standard Web browsers and Web servers can understand and talk in HTTP.

As pointed out in Section 3.1.1, it is important to design the C&S protocol so that it can be readily used by end-users and information providers without making many changes to the existing environment. As the Web is currently the prevalent way to access internet information, and since HTTP is the protocol used in the Web, the C&S protocol shall be built on top of HTTP. By doing this, all Web users can immediately use the new protocol as no new client software[3] is needed to support this protocol.

On the other hand, the C&S information suppliers do not need a special server, which can talk in a completely new protocol, to serve C&S information. They can keep using the standard Web servers they are using now, since HTTP is still being used in the protocol and all standard Web servers can understand HTTP. Only some software is needed on the Web server to provide the C&S functionalities. As a result, building the C&S protocol upon HTTP can greatly increase the ease of using the protocol and reduce the costs involved.

[3] Same as footnote 1 in Chapter 2.

Furthermore, HTTP is light-weight and fast [6]. By using HTTP as the base protocol, the C&S protocol can take these advantages which are certainly desirable in C&S applications.

3.2.2 Requests

Due to the generic nature of HTTP, it is possible to make application-specific requests in HTTP. HTTP may be used to perform specific tasks through the extension of its request methods. Web clients can pass additional information to Web servers through the Universal Resource Location (URL) of the request, or through HTML forms via the stdin of an executable on the Web server.

The HTTP-based C&S protocol shall keep using the standard HTTP "GET" and "POST" methods in protocol requests. This is because these methods are defined as the common methods for HTTP/1.0 [6], and it can be assumed that all Web servers that support HTTP can understand these methods. The difference between the two methods lies in the way that the server passes the request information to the server program that processes the request. In the "GET" method, request data is passed via the environment variables. In the "POST" method, the request data is delivered to the server program using the standard input or *stdin* [5].

Both "GET" and "POST" methods are used in protocol requests, because there are two different ways to supply commands and parameters to the Web servers on the client side. In the first way, Web clients specify the commands and arguments in the URL. The "GET" method is used in this case, and the commands and arguments are passed to the server request processing program via the environment variables. In the current implementation, only the environment variable QUERY_STRING is used to store the commands and arguments in the requests. QUERY_STRING includes everything following the "?" character in an URL [4]. This allows clients to make C&S requests by simply typing in the URL with the request commands and arguments. As the requests are in the form of a URL, it will later be shown that request URLs may be embedded within a C&S response to allow users to generate further requests from that response.

The other way to supply C&S commands and arguments to server CGI program is by means of the program stdin with the "POST" method. This mechanism is typically used when a form is submitted in the request (e.g. form to create a new appointment or form to edit an existing appointment).

It is possible to pass the form data with either the "GET" or the "POST" method, so why isn't the form data also passed with a "GET" method just like the requests made in direct URL? This is because when the "GET" method is used, the form data must then be passed to the server program via the environment variable QUERY_STRING. However, the user can also specify a separate request in the URL at the "action" parameter within the <FORM> tag, which is also passed to the server via the QUERY_STRING environment variable. In this case, it turns out that the form data overwrites the commands passed in the "action" URL. As a result, using the "GET" method in passing form data restricts requests so they cannot be specified in the form "action" URL. Moreover, by submitting form data via the "GET" method, the data in QUERY_STRING is encoded in a standard format, in which data is put in a stream of name and value pairs separated by the character '&' [4]. For example, suppose a form has input names “username” and “password” and corresponding values as “John” and “12345678”. The data are passed to the server program in QUERY_STRING as

“username=John&password=12345678”

This is a standard encoding scheme for passing form data to CGI programs. As a result, unless those requests made in direct URLs also follow this format, the request data passed in the QUERY_STRING environment variable can have two different formats -- one used in requests in direct URL (with application specific format), and the other used in requests with form data (with standard encoded form data format). This non-uniform formatting of input data in QUERY_STRING may confuse the server program, and the server program must be intelligent enough to interpret request data in both formats.

To avoid these problems, both “GET” and “POST” methods are supported in the C&S requests. Yet it is required that “GET” is only used in requests made in direct URLs, and “POST” is only used in requests made with forms. By this, the server processor can differentiate the two different types of requests easily. The two types of requests can be in different formats without bringing confusion to the server process, since they come into the process through different channels (one from the QUERY_STRING; the other from the stdin). Also with the “POST” method, data can be passed in the form “action” URL at the same time now, since the form data will not overwrite the QUERY_STRING environment variable.

3.2.3 Responses

The responses of the C&S protocol are in the form of standard HTTP responses, since the protocol is HTTP-based. Different MIME types may be returned to the clients as responses, but in the first stage the standard "text/html" content type shall be utilized, as most standard Web browsers can parse and display this type of responses without any helper applications[4]. This is a big advantage of using HTML pages as responses in the C&S protocol.

However, HTML responses have a lot of limitations. They do not support those rich user interface elements that are normally seen in a C&S application. Thus it is difficult to map the rich UI elements of the existing C&S applications to the limited UI elements in HTML, such as HTML pages, hyperlinks, HTML forms, tables and image maps. Also users must learn how to interact with the new UI. This is a big disadvantage of using HTML pages as responses.

Later in this thesis project, the idea of defining HTML responses with C&S application specific objects inserted will be explored. These objects may be C&S components that are understood and executed by certain Web browsers to implement much richer UI elements for C&S activities. This will be discussed in more details in Chapter 7.

Another noteworthy feature of responses in the C&S protocol is that within the HTML responses, hyperlinks with URLs for making further C&S requests may be included. Following these hyperlinks users can make additional C&S requests via a

[4] It is possible to define new MIME responses sepecific to the C&S system. In this case, special softwares (e.g. Navigator plug-ins) should be installed on the client side to view them. This will be further discussed in Chapter 7.

simple mouse click, rather than typing the crytic request commands. This is the beauty of using URLs to pass the requests in the protocol, and this makes the navigation in the calendar views much easier and much more convenient.

3.3 Response Template

The response template is an important component in the HTTP-based C&S protocol. It helps to achieve the design goal of minimizing network connections in the protocol, and provides the users a flexible way to define and control the appearance of the user interface.

3.3.1 The Idea

Suppose a user of the C&S service wants to get the times and descriptions of all the appointments in August 1996. A protocol could be designed which can only return a day of appointment information in a single request. In the example, 31 requests are needed to get the appointment information in the 31 days of August 1996. Nevertheless, this can be done in a more efficient way. The protocol can be extended to perform more tasks and return more information in a single request. Since most C&S products support displaying a month's worth of appointment information on the same page, it makes sense to add a new "month view" command into the protocol. This command takes in a date as the argument and the response will return all the appointments in the month of the date, the username of the current C&S session, as well as the request strings for getting appointment information in adjacent months of the date. This "month view" command can be perceived as a package of requests, where the requests are related and always come together at the same time.

We can look at response templates in the C&S protocol in the same way. Response templates are just templates of the HTML documents to be returned in the response, within which application specific command tags can be found. Using the same example above, we can just tell the server to use the response template "month view" in the response, and within the "month view" template there are command tags to list the appointments in a month on the HTML document. Other useful information can also be displayed at the same time. For example, the username of the current C&S session and the request links to access the appointments in previous and next months can also be displayed. Notice that now a single request in the protocol informs the server which response template to use, and that can result in a package of commands to be executed and multiple tasks to be performed.

The idea of response template and packing multiple requests into one not only reduces the number of requests we need to make in the C&S protocol, but also reduces the number of commands the client needs to know and remember. This also minimizes the number of network connections the client needs to make with the server, and makes the protocol simpler and more efficient. Later the importance of response templates in providing user interface flexibility will also be discussed.

Note that the idea of the response template is partly driven by the constraints in using the HTTP and simple Web browsers as the clients. The Web browser receives and displays a response immediately after it has made a request. It cannot store responses from multiple requests and display them altogther at a later time. As a result, only a single request may be made from the Web browser in each C&S activity. This request must perform all the desired tasks, and get a well formatted response to display within the browser. Response templates are a very good way to make this work.

3.3.2 Design Details

A response template is simply an ASCII text file that includes standard HTML tags and special command tags specific to the HTTP-based C&S protocol. These templates are used to generate real HTML documents which are the responses in the protocol. All the command tags begin with "<!--ORG-" and end with "-->". For example, the command tag "icon" appears as "<!--ORG-icon-->" in the response templates. Note that in HTML the tags "<!--" and " -->" allow comments to be included between them [4], so the C&S command tags in the templates are just comments to Web browsers. Even if the server CGI process cannot parse a particular command tag on a template and leaves it in the HTML response, the tag is just recognized as a comment and is ignored by the browsers.

For those components in the HTML responses that are the same for all users and all possible cases of a request, they simply appear as HTML tags and contents in the templates, e.g. the title of the HTML page, the image sources for the icons, and some of the form elements in a form to create a new appointment. Contents that are variable from one response to another will be generated on the fly when the server processes a request. These contents are generated by the special command tags in the templates. The positions of the command tags in the templates specify where the variable contents should be displayed within the HTML documents.

All the command tags are registered in the CGI process on the server. When the client makes a request, the CGI process will fetch the associated response template and process the template. Those standard HTML tags and non-variable contents are simply passed to the standard output to form the HTML document for response. When special command tags are encountered they are looked up in a command tag table in the CGI process, and the corresponding commands will be executed. These commands may involve the access of calendar information from a calendar database, manipulations on the parameters to form request anchors in the response, or fetching some session and configuration information. The results will be formatted in standard HTML tags and contents, and will be passed to the standard output. After the whole template has been processed, all the command tags within it are parsed and executed, and the resulting HTML document is generated and sent to the standard output (stdout) for response.

Table 5.1 in Section 5.1.4 contains a list of response templates, their corresponding uses and embedded commands.

3.3.3 Features

Packing Multiple Commands

As pointed out earlier, response templates pack multiple commands into one protocol request and allow clients to perform multiple tasks in that single request. This reduces the number of requests the clients need to make in the C&S activities, and reduces the number of request commands the clients need to know. Hence the performance of the C&S protocol is improved and the command set of the protocol is simplified.

Flexible User Interface Modification

Besides allowing the C&S protocol to pack multiple commands into a single request, the response template also provides a flexible way to manage and modify the user interface in the C&S system. As the templates are simply ASCII text files with standard HTML tags and contents plus the special C&S command tags, these text files can be modified easily to change the appearance of the responses, without any changes to the source code of the server CGI process.

This feature makes the life of the UI designers of the C&S system much easier, as they can just edit the template text files to modify the UI design, without writing or changing any source code. Moreover, when new HTML elements come up in the future, the UI designers can simply integrate these new elements in the response templates by editing the template text files, again without any changes to the source code. This gives more flexibility for UI designers to control and maintain the appearance of the user interface.

Furthermore, with the response templates, people who are using the C&S system can customize the user interfaces according to their preferences. For example, users in different corporations can customize the appearance of the responses on their own. They may add logos of their companies in the responses, add new links to Web sites related to their business, or broadcast company-wide announcements in the responses so that their employees can receive the messages while they do their C&S activities. All they need to do to make such customizations is to simply modify the response templates on the Web server where the OrgWeb system is installed.

3.3.4 Further Developments

Further developments can even make the response templates a more flexible tool to use. Some of the ideas are presented in this section.

Arguments in Command Tags and Flexibility in Appointment Listings

In the current design of the response templates, arguments may not be specified for the commands in the command tags. The commands use the arguments passed in the protocol requests to generate the information the users desire. This restricts the way appointment listings in the calendar view may be generated. For example in a weekly calendar view, one command tag "wvlistappt" is used to generate all the appointments in the current week in a predefined HTML table format. Users cannot alter the format of the appointment listings as they have been hardwired within the source code of the server CGI process. Alternatively, the "wvlistappt" command can be broken down into seven simpler commands "mon_listappt", "tue_listappt", "wed_listappt" and so on, which allow defining the format of the weekly appointment listings as well as positioning the listings in the HTML responses by editing the templates. However, the large number of new commands needed prevent the current implementation of the OrgWeb system from doing so.

Nevertheless, enabling arguments to be specified within the command tags would solve the problem of too many new commands. Instead of using different commands to list the appointments on different days of the week, the same command with different arguments could be used. For the above weekly calendar view example, we could have "wvlistappt='mon'", wvlistappt='tue'", "wvlistappt='wed'" and so on. By using this new scheme, the UI designers would have even more flexibility and control over the response appearance within the response templates.

Personalization and Themes in Responses

Individual users are allowed to have their own sets of response templates so that personalized C&S responses can be provided based on individual users' preferences. This involves modifying the template fetching process in the server CGI program. Instead of fetching the templates from a standard set, the process can fetch the templates from a set which is devoted to a particular user.

Alternatively, the users may choose from a number of pre-defined response sets to be the C&S responses. Different "themes" of responses can be selected according to the user's preference. This idea parallels the desktop themes in Microsoft Windows 95 and NT 4.0, and this can easily be implemented by providing users with groups of templates with the same UI themes and color schemes.

Chapter 4 Protocol Design for Scheduling

In the HTTP-based C&S protocol, users can set up group meetings among themselves over the internet. This involves choosing the right attendees for a meeting, selecting a convenient time for all the attendees, and sophisticated time negotiations among the attendees. These processes are much more complicated than simple viewing and editing of personal appointment entries over the internet, so the protocol design for internet scheduling is discussed in more details in this chapter. As a source of background information, the architecture of the scheduling system used in the C&S protocol is first presented in Section 4.1. Various design issues in the protocol for internet scheduling will be discussed in Section 4.2.

4.1 Scheduling System Overview

4.1.1 Basic Model

The scheduling processes and architecture used in the C&S protocol are modeled after the group scheduling features in Lotus Organizer 2.1 [7]. In this architecture, scheduling users set up meetings by trading meeting notices. These meeting notices are messages carrying meeting information, such as the meeting time and place, the person scheduling the meeting (the "chairperson"), and the attendees in the meeting. By exchanging meeting notices, users of the C&S protocol can communicate with each other, inviting others for meetings and negotiating on meeting time. With an Organizer backend, these meeting notices are transmitted through Notes mail or cc:Mail, and are deposited in Notes mail files or Organizer files. Upon response to the meeting notices, meeting entries are possibly inserted or updated in the scheduling users' calendar databases.

The best way to understand how the processes work is to see a typical scheduling example. Suppose a manager would like to schedule a meeting with his group. He first accesses the OrgWeb system and creates a new meeting in which he chooses the meeting time and the attendees. When the meeting is created, two things happen. First, a meeting entry is inserted in the manager's calendar database. Second, *invitation* meeting notices, carrying information about the meeting, are sent to all the attendees chosen in the meeting creation. These notices are deposited in the "mailboxes" of the attendees' calendar databases.

A few moments later, one of the group members accesses the OrgWeb system. He notices that there are new meeting notices in his "mailbox", so he opens the "mailbox" and reads the invitation meeting notice about the new meeting. Here he can see the chairperson, the time and the attendee information of the meeting. If he decides that he will join the meeting and accept the invitation, a meeting entry is inserted in his calendar database, based on the meeting time and other information carried by the meeting notice. At the same time, an *invitation acceptance* meeting notice is sent back to the manager, who is the chairperson of the meeting. Alternatively the invitee can decline the meeting invitation. In this case, an *invitation decline* meeting notice is returned to the manager, but no changes are made in the invitee's calendar database. When the manager checks his "mailbox" later, he will find and process these invitation acceptance or invitation decline notices. The attendee status of the meeting entry in his calendar database will be updated from “invited” to “accepted” or “rejected”. By this the manager is able to see who is able to attend the meeting and who is not.

In the scheduling architecture, only the chairperson can edit an existing meeting. Once the manager has created the meeting entry, he can change the meeting time, change the attendee list by adding more attendees or removing some attendees, or even delete the meeting. When the manager decides to change the meeting time, the meeting entry in his database will be updated, and *reschedule* meeting notices are sent to all attendees. The attendees can accept or decline the meeting reschedule, and the associated meeting entry in the attendee’s database is either updated or deleted respectively. Appropriate meeting notices are also sent back. If the manager adds new attendees or removes some of the existing attendees, those attendees who are affected will receive appropriate meeting notices (invitation notice for those added, and cancellation notice for those removed).

On the attendee side, an attendee cannot alter the attendee list of a meeting, but he can propose a change in the meeting time. In this case, he has accepted the meeting but would like to change the meeting time. A *reschedule proposal* meeting notice is sent to the chairperson. When the chairperson receives the reschedule proposal, he can decide to accept it and make an actual meeting reschedule, or simply decline that proposal.

This gives a rough idea of the group scheduling features in the OrgWeb system. A more detailed specification of the scheduling features can be found in Appendix C.

4.1.2 Scheduling Services

To facilitate more efficient group scheduling, two supporting services are provided in the scheduling system: the free time search service and the attendee directory service.

Free Time Search Service

When setting up a meeting, a chairperson needs to know other people's available times, so that he or she can determine the best time for the meeting. This can avoid a series of reschedule proposals and meeting reschedules after invitations for a new meeting are sent out, minimizing the flow of meeting notices and the time needed to reach a suitable meeting time for all the attendees.

With this service, when the chairperson chooses a person as the attendee of a meeting, the OrgWeb client will make a request to the OrgWeb server for a free time search for that person. The server will access the invitee's calendar database, and check for all the busy time periods for that invitee within a time period around a tentative meeting time. Note that the chairperson is only able to see whether that person is free or busy, without knowing what the person is actually doing. This protects the security of the invitee while giving the chairperson the information needed to schedule a meeting. The server returns these busy time periods back to the client side, and the busy periods are displayed in the Web browser. With the free time information for all the invitees, the chairperson can select a meeting time that can minimize time conflicts among the invitees' schedules.

Attendee Directory Service

For group scheduling in the OrgWeb system, the scheduling users are identified by their names used in Notes mail or cc:Mail. If the meeting chairperson types in a name for an invitee incorrectly when he chooses the invitees, meeting notices will not be delivered to the desired invitee, and all other invitees will get the wrong attendee information in the meeting notices.

In view of this problem, the scheduling system should provide the chairperson a list of names, from which the chairperson can choose to invite to a meeting. This cannot only tell the chairperson whom he or she can invite to a meeting, but also prevent the chairperson from misspelling the invitees' names. This is the attendee directory service in group scheduling. The OrgWeb server can obtain this attendee directory from the Notes Name & Address Book of the chairperson, and the list basically includes all the OrgWeb users whose databases can be accessed by the OrgWeb server. The list is passed to the Web client when the OrgWeb user wants to create a new meeting.

4.2 Scheduling Protocol Design

4.2.1 Design Goals

Specific to group scheduling processes and services in the OrgWeb system, there are some design goals to keep in mind:

Keeping Protocol Simple and Scaleable

As pointed out in Section 4.1, the group scheduling processes are pretty sophisticated. Although manipulating a meeting entry is very similar to manipulating a personal appointment entry, working on meeting notices can be a complicated task. There are many different types of meeting notices (there are 11 different types of notices in Organizer 2.1). Even worse, for each notice type users can respond to it differently. For example, in Organizer 2.1 a user can respond to a meeting invitation by accepting it, rejecting it, proposing a reschedule or delegating the invitation to another user. The sophistication of the meeting notices and the responses provide much flexibility in scheduling, but it can also make the internet C&S protocol too complicated to use. As a result, in designing the scheduling extension of the C&S protocol, keeping the protocol simple is one of the top priorities.

Moreover, as group scheduling involves lots of different user actions and responses, the scheduling protocol should be flexible enough that new meeting notices and user responses may be added easily. Making the protocol scaleable saves time and efforts in the future maintenance of the protocol.

Efficient Free Time Search

As seen in Section 4.1.2, free time search service keeps the group scheduling process efficient. However, this service may lead to very frequent interactions between the client and the server. For every attendee added in a meeting, the OrgWeb client needs to make a request to the server to ask for a free time search on that attendee.

Updating the free/ busy time information when the user changes the date and time of a meeting is worse. In every request for a free time search, all the busy periods in a user's entire calendar schedule are not returned, because this would be too much information to be returned in a single request. Instead, a time range is specified on the request so that the server needs only search for the busy time periods within that range.

Usually this time range starts a small time period before a tentative meeting time, and ends a small time period after that meeting time. Thus a scheduling user can view the free time information for all the attendees around that tentative meeting time. If there are time conflicts and the user moves the meeting time to a time outside the time range used in the previous free time search request, a new free time search request is required for every attendee who has been chosen, with a time range around the new tentative meeting time. This process goes on until the user finds a suitable meeting time for all the attendees. As transactions over the network can be slow, making free time search requests so frequently in this process certainly affects the performance of the free time search service.

Therefore, the scheduling protocol needs to be designed so that the free time search service is as efficient as possible, minimizing the number of requests required in scheduling a meeting.

How to design the scheduling protocol to achieve these design goals will be discussed in the next section. How to *use* the protocol to make the free time search service efficient will be discussed in Section 7.3.2.

4.2.2 Design Details

Meeting Entry Creation, Editing and Deleting

Creating, editing and deleting a meeting is very similar to creating, editing and deleting a personal appointment, except that the user needs to work on the attendee information, and uses the free time search and attendee directory services in working on meetings. A number of user interface enhancements (See Chapter 7) can make the editing of the attendee information and the use of the services intuitive and simple to a user, and these enhancements also keep the protocol simple. Adding the functionality of meeting entry manipulations to the protocol is simply adding request commands analogous to those request commands for appointment entry manipulations.

Meeting Notices Viewing and Processing

Meeting notices are created when a scheduling user creates a new meeting or responds to an existing meeting notice. Viewing the details in a meeting notice can be made similar to reading an appointment record or a meeting record[5].

To process a meeting notice, instead of having a separate protocol request command for each type of meeting notice and each possible notice response, the C&S protocol defines one request command to process all types of notices with all kinds of responses. In this request command, the record ID for the meeting notice to process and the user response to the notice are supplied as the command arguments. With the record ID, the OrgWeb server can retrieve the meeting notice from the database, and checks which type the meeting notice is. Once the notice type is returned, the user response argument is used by the server to determine how to process the meeting notice. Under this scheme, the scheduling protocol can stay simple. This is also a scaleable solution to accommodate the meeting notice processing feature, because the addition of new user actions and responses only adds the possible notice types for the notice records and the possible choices used in the user response argument of the notice processing command. No new request command will be required.

[5] In Organizer databases, meeting notices, appointment entries and meeting entries are all "record" entities. The type of that record is detemined by the section in which the record is located and some of the field values in the record.

Free Time Search

Since the scheduling users choose the attendees to invite after they have requested for a form for creating a meeting, in order to request and display the free time for the attendees, an extra connection to the server cannot be avoided when each attendee is added to the attendee list. Yet we can make the free time search more efficient by allowing the protocol to search free time periods for *multiple* users within the same time range. With the capability to search free time for multiple users in the protocol, the Web client only needs to make a single request to update the free time display for all the attendees chosen. This definitely saves a lot of server connections in the case of changing the tentative meeting time when the scheduling user tries to choose a good final meeting time. As the free time request can handle multiple users, only one request is needed to fetch the free time for all the attendees in the attendee list.

Chapter 5 Protocol Definition

5.1 Parameters

5.1.1 Commands

Commands are the central components in C&S requests in the protocol. They tell the server what operations to perform and with what information to respond.

Request commands are used by a client in making a C&S request in the protocol. They can be specified in the URLs after the "?" character, or in an HTML form. Commands in the URLs are submitted to the server CGI process through the QUERY_STRING environment variable, and this can happen when either "GET" or "POST" method is used[1]. Commands in an HTML form are submitted to the server CGI process through the stdin, and this can only happen when the "POST" method is used. Clients can specify arguments in request commands.

There are also commands in the command tags embedded within the response templates, and they are executed by the CGI process when the response templates are parsed to form a response. These commands take no arguments, but use the arguments that are passed in the request commands when protocol requests are made. Note that the clients do not normally see these template embedded commands.

[1] A URL with a request can be specified in the "action" parameter of the <FORM> tag when an HTML form is submitted with the "POST" method.

5.1.2 Session ID

A session ID identifies the C&S session for a particular user. It is a 36-character string that is transformed from a Universally Unique Identifier (UUID), defined by the Open Software Foundation (OSF) as part of their Distributed Computing Environment (DCE) [8]. It is a string consisting of 8 hexadecimal digits, followed by a hyphen, then three groups of 4 hexadecimal digits each followed by a hyphen, then 12 hexadecimal digits. It virtually guarantees that a computer-generated UUID and hence the session ID is unique in the world across time and space.

e.g. “6B29FC40-CA47-1067-B31D-00DD010662DA”

5.1.3 Date

The date parameter tells the server the date that the current C&S command request should work on. It is an 8-character string, with the following format:

Character	0	1	2	3	4	5	6	7
Meaning	Year				Month		Day	

e.g. “19960826” stands for August 26, 1996.

5.1.4 Datetime

The datetime parameter is used in the free/ busy time searching in group scheduling. It tells the server in what time range it should search for the busy time periods for a user. It is a 12-character string, with the following format:

Character	0	1	2	3	4	5	6	7	8	9	10	11
Meaning	Year				Month		Day		Hour		Minute	

e.g. “199608261715” stands for 17:15 on August 26, 1996.

5.1.5 Response Template

The idea and design of the response template has been discussed in Section 3.3. Table 5.1 below lists the response templates in the current implementation of the protocol. The meanings of the embedded commands within the templates are listed in Appendix B.

Name	Uses	Commands embedded within Template
logon	Logon page for the user to login	icon, script, today, login
dview	Calendar Day View	name, script, icon, dprev, dfwd, sessionid, date, dvlistappt, pendingnotice
wview	Calendar Week View	name, script, icon, wprev, wfwd, sessionid, date, wvlistappt, pendingnotice
mview	Calendar Month View	name, script, icon, mprev, mfwd, sessionid, date, mvlistappt, pendingnotice
newappt	Page for creating new appointment	script, icon, back, sessionid, date, daystr, createappt, categorynames
editappt	Page for editing or deleting an existing appointment	script, icon, back, sessionid, curruid, date, daystr, updateappt, editapptform
newmtg	Page for creating new meeting	scripts, icon, date, sessionid, createmtg, categorynames, name, dtrackdate
editmtg	Page for editing or deleting an existing meeting	script, icon, date, back, sessionid, updatemtg, editmtgform
notice	List of the meeting notices in mailbox	name, script, icon, back, date, sessionid, listnotices
mtgmsg	Page for viewing the details of a meeting notice	script, icon, back, date, sessionid, msgdetails

Table 5.1: List of Response Templates

5.1.6 Record ID

A record ID identifies an appointment record, a meeting record or a meeting notice record in an Organizer file. The record ID is in the form of a pair of numbers separated by the character "-". The pair of numbers are used to form a unique ID for a record in an Organizer file. For requests dealing with a particular appointment, meeting or meeting notice in the protocol, the record involved is identified by the record ID.

5.2 Requests

In the C&S protocol, requests can be made with either the "GET" method or the "POST" method in the underlying HTTP requests.

5.2.1 With "GET" Method

The request string is passed in the URL. The URL can be decomposed as follows:

protocol://host/script?request string

protocol	Protocol used between the server and the client. In the C&S protocol it is always "http" or "https".
host	The host machine where the Web server sits, and where the C&S server CGI process is located. e.g. "torch.lotus.com"
script	The filespec for the C&S server CGI executable on the host machine. e.g. "orgweb/test/orgweb.exe"
request string	Contains the request command and associated arguments.

The format used in the request string is defined as follows:

request string ::= command='argument, ...'

In the request string, **command** is the request command that the client desires to perform, and **argument, ...** are the arguments associated with the request command. The number of arguments depends on the command. Multiple arguments for any command are delimited by the character ','. For example, in request string "foo1='bar1,bar2,'", "foo1" is the command in the request. It has three arguments in this case: "bar1", "bar2", and "" (empty string). The request string in the URL will be passed to the server CGI process via the QUERY_STRING environment variable.

5.2.2 With "POST" Method

With the "POST" method, the C&S requests can be made in two ways simultaneously. First, a request can be made through a HTML form, and the request string will be passed to the server CGI process via the standard input (stdin). Second, the request can be made through the URL in the "action" parameter of the FORM tag, just in the same way as it is in the "GET" method. The request string will be passed to the server CGI through the QUERY_STRING variable.

Within a form, the first hidden input is the request command. The names and values of the form input elements following the request command are the associated arguments. The values in the form input elements shall be set to whatever values the user enters within the HTML form. It is required in the C&S protocol that the request command must have the hidden input name "cmd", and the command itself as the corresponding hidden input value. This helps the server CGI process to differentiate the command inputs from the argument inputs. Also in a request from a form, all arguments have an argument name, while in a request through the URL arguments have no names and only argument values are passed.

An HTML form which allows a user to login is shown in Figure 5.1. Within the form the command "login" can be found, which appears in the first hidden input within the form. Note that in the hidden input the input name is "cmd", and the input value is "login". The form input elements following this are command arguments associated with the command "login".

```
<FORM method=post action="/orgweb/test/orgweb.exe?wview=',19960826'">
<input type="hidden" name="cmd" value="login">
<TABLE border=0>
<TR><TD><B>User Name: </B></TD><TD><input type="text" name="username" value=""
maxlength=126></TD></TR>
<TR><TD><B>Password: </B></TD><TD><input type="password" name="passwd"
maxlength=10></TD></TR>
<TR><TD><B>Organizer File: </B></TD><TD><input type="text" name="orgfile" value=""
maxlength=126></TD></TR>
<TR><TD><B>Organizer Password: </B></TD><TD><input type="password" name="orgfilepasswd" value=""
maxlength=126></TD></TR>
</TABLE>
<input type="submit" value="  OK  "><p>
</FORM>
```

Figure 5.1 The login form in HTML

After the form is submitted, the server CGI executable "orgweb.exe" will get the request string

"cmd=login&username=Oliver&orgfile=c:\org3\work\organize\oyip.or3&orgfilepasswd =password"

through the stdin. This encoding is done by the Web server before passing the data to the CGI process. The formats that different fields are delimited by '&' and field names and values are separated by '=' are defined in the HTTP protocol specification [4].

In the example above there is one command "login" in the form. There are four arguments, with argument names "username", "passwd", "orgfile" and "orgfilepasswd", and with values "Oliver", "password", "c:\org3\work\organize\oyip.or3", "password" respectively.

Why do the argument names need to be specified in requests with forms, but not in requests through URLs? This is because in some of the forms in the C&S requests, it is possible to specify multiple values in the same input. For example, a user may select multiple categories within the "Categories" SELECT input element when creating a new appointment. Selecting multiple entries in the SELECT box will result in more than one field in the encoded request string having the same field name. For example, if items "Calls", "Expenses" and "Meetings" are picked in a SELECT element named "category", the encoded request string will appear as "...&category=Calls&category=Expenses&category=Meetings&....". That means in the request some arguments can occur more than once in the request string, and the number of times they occur may be variable. In order to recognize the arguments correctly, the rule that each argument must have a name in a request submitted through an HTML form is required.

5.2.3 Specification of Request Commands

The specification of the request commands in the C&S protocol are shown in this section.

`logon`

Request Channel: URL
Synopsis: `logon`

Returned value: An HTML response with a form for the user to login to a new C&S session.

`login`

Request Channel: HTML Form

Argument	Value
`username`	Organizer username
`passwd`	Password for the Organizer user specified in `username`
`orgfile`	Organizer file to open
`orgfilepasswd`	Password for the Organizer file specified in `orgfile`

The `login` command starts a new session in the C&S protocol.
Returned value: A session ID that identify the new C&S session.

`logout`

Request Channel: URL
Synopsis: `logout='SessionID'`

Argument	Value
`SessionID`	Session ID for the C&S session to logout

The `logout` command terminates an existing session in the C&S protocol.

`dview`

Request Channel: URL
Synopsis: `dview='SessionID,Date'`

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Date`	Date of calendar information to view

Returned value: An HTML response with the view of the appointment information on the day specified by `Date`.

`wview`

Request Channel: URL

Synopsis: `wview='SessionID,Date'`

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Date`	Date of calendar information to view

Returned value: An HTML response with the view of the appointment information in the week containing `Date`.

`mview`

Request Channel: URL

Synopsis: `mview='SessionID,Date'`

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Date`	Date of calendar information to view

Returned value: An HTML response with the view of the appointment information in the month containing `Date`.

`newappt`

Request Channel: URL

Synopsis: `newappt='SessionID,Date,ReturnCalendarView'`

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Date`	Date on which the new appointment will be inserted
`ReturnCalendarView`	Calendar view from which the `newappt` command is called. It can be *dview*, *wview* and *mview*. When the create appointment activity is done or canceled, the system will return to this calendar view.

Returned Value: An HTML response with a form for the user to create a new appointment on the date specified by `Date`.

`newmtg`

Request Channel: URL

Synopsis: `newmtg='SessionID,Date,ReturnCalendarView'`

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Date`	Date on which the new meeting will be inserted
`ReturnCalendarView`	Calendar view from which the `newmtg` command is called. It can be *dview*, *wview* and *mview*. When the create meeting activity is done or canceled, the system will return to this calendar view.

Returned Value: An HTML response with a form for the user to create a new meeting on the date specified by `Date`. The date can be changed by the date control on the form to create a new meeting on a date other than `Date`.

`createappt`

Request Channel: HTML Form

Argument	Value
`sessionid`	Session ID to identify the C&S session
`date`	Date on which the new appointment will be inserted
`starthr`	The hour of the starting time of the new appointment. Range from 0 - 23
`startmin`	The minute of the starting time of the new appointment. Range from 0 - 55
`durhr`	The hour of the duration of the new appointment. Range from 0 - 23
`durmin`	The minute of the duration of the new appointment. Range from 0 - 55
`description`	Non-empty string that describes the appointment
`category`	Categories that the new appointment belong to. Can be absent or occur more than once.
`conflict`	Flag to indicate whether to warn the user if there is a conflict in the schedule. Has value equal to 'C' (checked) if the flag is set. Argument is absent if the flag is not set.
`pencilin`	Flag to indicate whether to pencil in the new appointment. Has value equal to 'C' (checked) if the flag is set. Argument is absent if the flag is not set.
`confidential`	Flag to indicate whether the new appointment is confidential. Has value equal to 'C' (checked) if the flag is set. Argument is absent if the flag is not set.

The `createappt` command creates a new appointment in the date specified by `Date`, with attributes specified by the arguments in the command.

`createmtg`

Request Channel: HTML Form

Argument	Value
`sessionid`	Session ID to identify the C&S session
`attendeeinfo`	Attendee Information for the meeting. Information for each attendee is delimited by the ':' character. For each attendee, the attendee information is in the form: `AttendeeName,AttendeeStatus,AttendeeRequired` where `AttendeeName` is the name of the attendee, `AttendeeStatus` is the status of the attendee in the meeting and `AttendeeRequired` specifies whether the attendee is a required attendee in the meeting. `AttendeeStatus` can be *1*, *2*, *3*, ... *10*, meaning: *1 - Invited* *2 - Rejected* *3 - Accepted* *4 - Delegated* *5 - Penciledin* *6 - Invited Delegation* *7 - Accepted Delegation* *8 - Rejected Delegation* *9 - Chairman* *10 - Room* `AttendeeRequired` can be either *1* or *0* to indicate the attendee is required in the meeting or not respectively.
`date`	Date on which the new meeting will be inserted
`starthr`	The hour of the starting time of the new meeting. Range from 0 - 23
`startmin`	The minute of the starting time of the new meeting. Range from 0 - 55
`durhr`	The hour of the duration of the new meeting. Range from 0 -23
`durmin`	The minute of the duration of the new meeting. Range from 0 - 55
`description`	Non-empty string that describes the meeting
`category`	Categories that the new meeting belong to. Can be absent or occur more than once.
`conflict`	Flag to indicate whether to warn the user if there is a conflict in the schedule. Has value equal to 'C' (checked) if the flag is set. Argument is absent if the flag is not set.
`pencilin`	Flag to indicate whether to pencil in the new meeting. Has value

	equal to 'C' (checked) if the flag is set. Argument is absent if the flag is not set.
confidential	Flag to indicate whether the new meeting is confidential. Has value equal to 'C' (checked) if the flag is set. Argument is absent if the flag is not set.

The `createmtg` command creates a new meeting in the date specified by `Date`, with attributes specified by the arguments in the command.

`editappt`

Request Channel: URL

Synopsis: `editappt='SessionID,Date,id,ReturnCalendarView'`

Argument	Value
SessionID	Session ID to identify the C&S session
Date	Date on which the appointment to edit is found
id	Appointment ID for the appointment record to edit
ReturnCalendarView	Calendar view from which the `editappt` command is called. It can be *dview*, *wview* and *mview*. When the edit appointment activity is done or canceled, the system will return to this calendar view.

Returned Value: An HTML response with a form for the user to edit an existing appointment on the date specified by `Date`.

`editmtg`

Request Channel: URL

Synopsis: `editmtg='SessionID,Date,id,ReturnCalendarView'`

Argument	Value
SessionID	Session ID to identify the C&S session
Date	Date on which the meeting to edit is found
id	An meeting ID for the meeting record to edit
ReturnCalendarView	Calendar view from which the `editmtg` command is called. It can be *dview*, *wview* and *mview*. When the edit meeting activity is done or canceled, the system will return to this calendar view.

Returned Value: An HTML response with a form for the user to edit an existing meeting on the date specified by `Date`.

`updateappt`

Request Channel: HTML Form

Argument	Value
`sessionid`	Session ID to identify the C&S session
`date`	Date on which the appointment to update is found
`id`	An appointment ID for the appointment record to update
`starthr`	The hour of the starting time of the appointment. Range from 0 - 23
`startmin`	The minute of the starting time of the appointment. Range from 0 - 55
`durhr`	The hour of the duration of the appointment. Range from 0 -23
`durmin`	The minute of the duration of the appointment. Range from 0 - 55
`description`	Non-empty string that describes the appointment
`category`	Categories that the appointment belong to. Can be absent or occur more than once.
`Conflict`	Flag to indicate whether to warn the user if there is a conflict in the schedule. Has value equal to 'C' (checked) if the flag is set. Argument is absent if the flag is not set.
`Pencilin`	Flag to indicate whether to "pencil in" the appointment. Has value equal to 'C' (checked) if the flag is set. Argument is absent if the flag is not set.
`Confidential`	Flag to indicate whether the appointment is confidential. Has value equal to 'C' (checked) if the flag is set. Argument is absent if the flag is not set.

The `updateappt` command update an existing appointment identified by the appointment ID `id`, with attributes specified by the arguments in the command.

`delappt`

Request Channel: URL

Synopsis: `delappt='SessionID,Date,id,ReturnCalendarView'`

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Date`	Date on which the appointment to delete is found. It also sepecify the date of calendar view to show after the appointment deletion
`id1`	An appointment ID for the appointment record to delete
`ReturnCalendarView`	Calendar view to show after the appointment deletion. It can be *dview*, *wview* and *mview*.

The `delappt` command delete an existing appointment identified by the appointment ID `id,` and return to the calendar view specified by `ReturnCalendarView.`

Returned Value: An HTML response with the calendar view specified by `ReturnCalendarView` for the date specified by `Date`.

`notice`

Request Channel: URL

Synopsis: `notice='SessionID,Date,ReturnCalendarView'`

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Date`	Current Date. When the meeting notice viewing is done, the system will return to a calendar view (specified by `ReturnCalendarView`) on this date.
`ReturnCalendarView`	Calendar view from which the `notice` command is called. It can be *dview*, *wview* and *mview*. When the meeting notice viewing is done, the system will return to this calendar view.

Returned Value: An HTML response showing a list of meeting notices in the user's mailbox. The senders, the types and the subjects of the meeting notices are displayed.

`mtgmsg`

Request Channel: URL

Synopsis: `mtgmsg='SessionID,Date,id,ReturnCalendarView'`

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Date`	Current Date. When the meeting notice viewing is done, the system will return to a calendar view (specified by `ReturnCalendarView`) on this date.
`id`	A meeting notice ID for the meeting notice record to view
`ReturnCalendarView`	Calendar view from which the `mtgmsg` command is called. It can be *dview*, *wview* and *mview*. When the meeting notice viewing or processing is done, the system will return to this calendar view.

Returned Value: An HTML response displaying the details of the meeting notice.

`noticeproc`

Request Channel: URL

Synopsis: `noticeproc='SessionID,Date,id,NoticeResponse,`
`ReturnCalendarView'`

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Date`	Current Date. When the meeting notice processing and viewing is done, the system will return to a calendar view (specified by `ReturnCalendarView`) on this date.
`id`	A meeting notice ID for the meeting notice record to process
`NoticeResponse`	The user response to the meeting notice. It can be *acc* (accept) or *decl* (decline).
`ReturnCalendarView`	Current calendar view. It can be *dview*, *wview* and *mview*. When the meeting notice viewing and processing is done, the system will return to this calendar view.

The `noticeproc` command processes a meeting notice identified by the notice IDs `id`, with the user response specified by `NoticeResponse`.

`busytime`

Request Channel: POST method in WinInet API Calls

Argument	Value
`SessionID`	Session ID to identify the C&S session
`Startdt`	Starting Datetime of the datetime range to serach for the busy periods
`Enddt`	Ending Datetime of the datetime range to search for the busy periods
`User`	The user from whose calendar schedule to search for the busy periods. Can occur more than once to search for busy periods for multiple users.

Returned Value: An HTML response displaying the busy time periods for the users specified by `User`, within the datetime range specified by `Startdt` and `Enddt`. The response is in the format:
userbusyinfo_1#userbusyinfo_2#....#userbusyinfo_n##
where
userbusyinfo_i ::=
*username:busyperiod_1*busyperiod_2*...*busyperiod_n*
where *username* is the name of user to search the busy periods for, and
busyperiod_i ::= *periodstartdt_i-periodenddt_i*

where *periodstartdt_i* and *periodenddt_i* are the starting datetime and ending datetime of the busy period i respectively.

e.g. A possible response for the request

"cmd=busytime&startdt=199608260000&enddt=199608262359&user=Alpha Tune&user=Beta Tune&user=Theta Tune"

can be:

```
Alpha Tune:
199608260900-199608261000*
199608261500-199608261630*
#
Beta Tune:
199608261100-199608261230*
199608260200-199608260215*
199608261900-199608262200*
#
Theta Tune:
199608261500-199608261630*
#
#
```

5.3 Responses

The responses in the C&S protocol are standard HTML responses, with content type "text/html". These responses are generated from a number of response templates on the server side (See Section 3.3). In the response templates, there are command tags that are parsed and executed by the server CGI process to generate the actual response. A summary of these command tags can be found in Appendix B.

5.4. Examples

5.4.1 Example on Accessing Personal Appointment Data

In this section, an example of how to make requests in the C&S protocol within a session is presented. The expected responses for the requests are also shown.

- **Go to the OrgWeb logon page**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?logon ("GET" method)
Response: The logon page with the login form is returned.

- **Submitting the login form to login**

Request: action = "/orgweb/test/orgweb.exe?wview=',19960826'" ("POST" method)
Logon form has hidden command "login", which will be passed to the server through the stdin, with other arguments set by the input elements in the form.
Response: User login a C&S session. The calendar week view for 8/26/1996 is shown after successful login.

Note in the logon form the current date is used in the command `wview` by default.

- **See calendar information of previous week**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?wview='6B29FC40-CA47-1067-B31D-00DD010662DA,19960819' ("GET" method)
Response: The calendar week view for 8/19/1996 is shown.

Similar requests can be made by specifying different combinations of calendar view command and date argument to view calendar information of a particular day in a particular view.

- **Create a new appointment on 8/20/1996**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?newappt='6B29FC40-CA47-1067-B31D-00DD010662DA,19960820,wview' ("GET" method)
Response: The create appointment page and form is returned.

- **Submitting the create appointment form**

Request: action="/orgweb/test/orgweb.exe?wview=’6B29FC40-CA47-1067-B31D-00DD010662DA,19960820’" (“POST” method)
Form has hidden command “createappt”, which will be passed to the server through the stdin, with other arguments set by the input elements in the form.

Response: Create a new appointment in current date. Return to Week View for 8/20/1996.

- **View appointment details/ edit an appointment on 8/20/1996**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?editappt=’6B29FC40-CA47-1067-B31D-00DD010662DA,19960820,618-10644986,wview’ (“GET” method)

Response: The edit appointment page and form is shown, with current appointment attributes preset in the form.

- **Delete the appointment**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?delappt=’6B29FC40-CA47-1067-B31D-00DD010662DA,19960820,618-10644986,wview’ (“GET” method)

Response: Appointment deleted. Return to Week View for 8/20/1996

- **Submitting the edit appointment form**

Request: action="/orgweb/test/orgweb.exe?wview=’6B29FC40-CA47-1067-B31D-00DD010662DA,19960820’” (“POST” method)
Form has hidden command “updateappt”, which will be passed to the server through the stdin with other arguments set by the input elements in the form.

Response: Update the current appointment with the attributes set in the form. Return to calendar week view for 8/20/1996.

- **Logout the C&S session**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?logout=’6B29FC40-CA47-1067-B31D-00DD010662DA’ (“GET” method)

Response: Logout the current C&S session.

5.4.2 Example on Group Scheduling

In this section, an example of how to make group scheduling requests in the C&S protocol is presented. In this example, there are three scheduling example, and their names are “Alpha Tune”, “Beta Tune” and “Theta Tune”. “Alpha Tune” wants to schedule a new meeting with other two people, and he is the chairperson of the meeting.

- **Create a new meeting on 8/20/1996**

Requests: http://torch.lotus.com/orgweb/test/orgweb.exe?newmtg='61F4D460-EFB4-11CF-9FC9-00805FC24DFE,19960820,wview' (“GET” method)
Response: The create meeting page and form is returned.

- **Submitting the create meeting form**

Request: action="/orgweb/test/orgweb.exe?wview='61F4D460-EFB4-11CF-9FC9-00805FC24DFE,19960820'" (“POST” method)
Form has hidden command “createmtg”, which will be passed to the server through the stdin, with other arguments set by the input elements in the form.
Response: Create a new meeting on current date. Invitation meeting notices sent to “Beta Tune” and “Theta Tune”. Return to Week View for 8/20/1996.

- **View meeting details/ edit meeting on 8/20/1996**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?editmtg='61F4D460-EFB4-11CF-9FC9-00805FC24DFE,19960820,618-10644986,wview' (“GET” method)
Response: The edit meeting page and form is shown, with current meeting attributes preset in the form.

After a while “Beta Tune” logs in an OrgWeb C&S session. He notices that there are new meeting notices in the “mailbox” of his database.

- **View meeting notices in mailbox**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?notice='7AADDE50-EFB7-11CF-9FC9-00805FC24DFE,19960820,wview' (“GET” method)
Response: The meeting notice page is shown. Short descriptions and sender information of all the meeting notices in the calendar mailbox are displayed on the page. In this example, an invitation meeting notice from “Alpha Tune” can be found.

- **View the details of a meeting invitation**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?mtgmsg=' 7AADDE50-EFB7-11CF-9FC9-00805FC24DFE,19960820,9206-17739540, wview' (“GET” method)
Response: The details of the invitation meeting notice is displayed. Information such as the chairperson of the meeting, meeting time, and people in the meeting are shown.

- **Accepting the meeting invitation**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?noticeproc=' 7AADDE50-EFB7-11CF-9FC9-00805FC24DFE,19960820,9206-17739540,acc, wview' ("GET" method)

Response: A meeting entry is inserted in the calendar database. Meanwhile an invitation acceptance meeting notice is sent to the chairperson of the meeting.

- **View meeting details as an attendee**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?editmtg='61F4D460-EFB4-11CF-9FC9-00805FC24DFE,19960820,681-92621105,wview' ("GET" method)

Response: The edit meeting page and form is shown, with current meeting attributes preset in the form.

At the same time, "Theta Tune" also logs in an OrgWeb session, and also finds the meeting invitation from "Alpha Tune". After reading the details of the meeting notice, he decides not to attend the meeting.

- **Declining the meeting invitation**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?noticeproc='901DCE32-7BEF-11CF-9FC9-00805FC24DFE,19960820,9206-17739540,decl, wview' ("GET" method)

Response: An invitation decline meeting notice is sent to the chairperson of the meeting. No meeting entry is inserted in the calendar database.

Some time later "Alpha Tune" logs in again, and finds out that "Beta" has sent him an invitation acceptance notice, while "Theta" has sent him an invitation decline notice.

- **Processing an invitation acceptance**

Request: http://torch.lotus.com/orgweb/test/orgweb.exe?noticeproc='178756EC-BA65-11CF-9FC9-00805FC24DFE,19960820,9206-31714C9E,acc, wview' ("GET" method)

Response: The invitation acceptance notice is processed and the status of the attendee who sent the notice is updated in the meeting entry in the chairperson's database.

Similarly the chairperson can process the invitation decline meeting notice.

Chapter 6 Session Management

I. Introduction

Session management is another important component in the HTTP-based C&S protocol. Without session management, the protocol cannot really work. In this chapter the problems of the protocol without session management will be identified, and then the design and the implementation of session management for the C&S system will be discussed.

6.1.1 Background for Organizer API

The need for session management is partly driven by the way in which the Organizer API is used to access the Organizer backend. Through the Organizer API calls, a user may read and write information in Organizer (.OR3) files or Lotus Notes mail files. However, before the records in the Organizer files can actually be accessed, the user needs to initialize an API session with the user's username in his or her mail account and the associated password [9]. After the initialization an Organizer API handle will be returned for further interactions within the API session. Moreover, the user also needs to open the Organizer file to work on, with its filespec and the password to get a file handle to access the database. After the user has finished accessing the information in the file, he or she needs to close that file and terminate the API session.

6.1.2 The Need for Session Management

HTTP is known as a “stateless” protocol, because the connection between the client and the server is terminated after each response from the server [3]. The server cannot keep state to support consistent behaviors between different connections. As the protocol is HTTP-based, the user will have problems in using the Organizer API layer to access the databases due to this “stateless” nature of HTTP. Since information such as the API handle and Organizer file handle are not kept from one connection to another, in each connection the user needs to initialize a new API session and open the database file. When the user finishes accessing the database file, he or she also needs to close the file and the API session before terminating the HTTP connection.

Two problems arise if the user uses a new API session, opening and closing the database file in every single request in the C&S protocol. First, since the user’s username and password in the mail account are required in initializing an API session, and since the Organizer files’ filespecs and access passwords are required in opening the database files, the user needs to supply these parameters in every request in the protocol. This is a very inefficient way to make requests, and passing the usernames and passwords in every request is poor from a security point of view.

One way to solve this problem is to form a session handle from the usernames and passwords. An encoding system can be designed to encode all the parameters for the API initialization and file open into a magic code. This magic code can be decoded to those parameters on the server side. When the client first makes a request in the protocol and supplies the usernames, passwords and other parameters, these parameters will be encoded to form the magic code, which is returned to the client. In subsequent requests, the client can supply the magic code in the requests, and the server can decode it to get back the usernames, passwords and parameters to make API initializations and file opens. Though the server is not really keeping any state, this scheme essentially provides session-like behaviour. However, a magic code in this scheme is insecure because crackers may be able to decode the magic code to get back the usernames and passwords.

The second problem in using a new API session and opening the database files in every request is that the performance of the C&S protocol will be severely affected. It is time-consuming and inefficient to open a new API session and open the database file again in every request, while a user is likely to access the same database file from one request to another.

Due to the above problems, a good and secure way to support session management is certainly needed in the C&S protocol.

6.1.3 Design Goals

Based on the problems pointed out in the previous section, there are several goals when the session management system is designed for the C&S protocol.

Minimizing API Session and File Operations

Minimizing the number of API session initializations and Organizer file opens can reduce the overhead involved in accessing the database information, and hence make the protocol perform better. This requires the protocol to keep state on the server side between different protocol requests, such as the usernames, passwords and filenames to open the API session and the Organizer files. Ideally for every protocol session, a user only needs to initialize one API session and open a particular database file once.

Security

A secure method should be designed to compose the session ID to identify a particular session in the protocol. Crackers should not be able to "decode" the session ID back to information such as the usernames and passwords. The authentication of the user should be dependent on the location of the session login, so that it is impossible to steal a session ID and use it to access a database file from a client machine other than the one from which the original session was established.

Multiple sessions and requests simultaneously

Multiple sessions should be allowed in the protocol, so that the same server can serve multiple clients at the same time. Moreover, for a busy Web server, it is possible that more than one C&S client will make requests to the server at exactly the same time. The session management system should be designed so that it can handle multiple requests simultaneously. By achieving this, the clients have a lower chance of having their requests refused by the server because the server is busy serving other clients.

6.2 Design Details

6.2.1 Separate Process from CGI Process

As pointed out in the previous section, to meet the design goal of minimizing API sessions and file operations, state needs to be kept on the server side in the protocol to store information for the API session and database file. However, in the current implementation of CGI program at Web servers, a new CGI process is started every time when the Web client makes an HTTP request, and the process terminates when the HTTP connection terminates [4]. As a result, if only a CGI process is used to serve the C&S requests in the protocol, state cannot be kept between different requests in the CGI process. A separate process on the Web server is necessary, which is always running on the Web server to keep the state across different C&S requests. Different CGI processes will communicate with this new separate process for session management and database access.

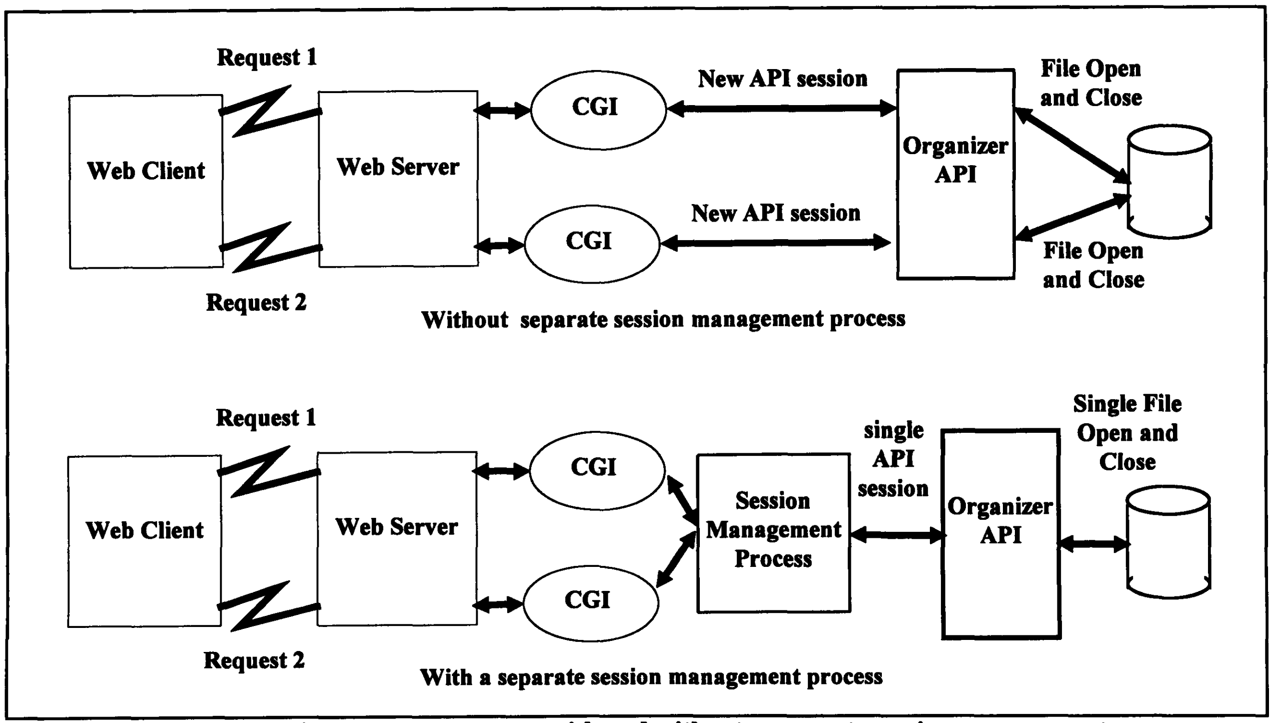


Figure 6.1: Comparison between systems with and without a separate session management process. With the separate process, only a single API session and single file open and close are required for a particular Web client. Information about the API session and database file is kept in the session management process. Different requests and CGI processes can use this information in the session management process to access the database.

The difference between a C&S system with and without the new separate session management process is illustrated in Figure 6.1. Without the separate process, every Web request starts a new instance of the CGI process, and each instance initializes a new

Organizer API session and opens the database file on their own, and closes the API session and database file after the request has been served. With the separate process, each instance of the CGI process talks to the new process, where the API session and database file information is kept. The separate process can use this information to actually access the database and return the desired database information to each instance of the CGI process. The API session is only initialized once and the file is only opened once to get the API session and database file information. In this way, the system with the separate session management process is more efficient.

Note that there is a client/ server relationship between the session management process and the CGI process. Each instance of the CGI process acts like a client, who asks for information about the Organizer database from the session management process, which acts like a server in this case.

6.2.2 Communication with the CGI Process

As two different processes are now involved in the C&S protocol on the server side, a communication method between the two processes has to be designed. There are several mechanisms for interprocess communication (IPC) in Windows NT system [10], which can be used to develop the communication channel between the CGI process and the session management process. The implementation details of the IPC on a Windows NT system will be discussed in the Section 6.3.2.

The design and the formats of the requests and responses between the two processes are discussed here. Suppose the communication channel has been established between them, how should the CGI process tell the session management process the Organizer file operations it wants? And how should the session management process pass the CGI process the database information requested? For the requests, there are different types of session management and Organizer file operations the CGI process may want to perform. They include logging on and logging off a C&S session; create, update and delete a particular appointment record; generate a list of calendar appointments within a date range; and generate an edit appointment form with fields preset with the attributes of an existing appointment. These different requests can have different parameters passed to the session management process, which actually performs the session or database operations. Instead of having different ways to make a request in all these cases, the request method can be simplified by using a standardized and universal request object in all request cases. The parameters in this standard request object can be categorized into the following five groups:

- Common parameters: The session ID, remote host of the C&S request (for authentication) and the request type. Required in all types of requests.
- Login parameters: Username and password for mail account, and filespec and access password for Organizer database file. Required in a request for session login.
- Appointment/ meeting/ meeting notice unique ID: Required in editing, deleting and updating a particular appointment/ meeting record in the database file. Also required in viewing and processing a particular meeting notice record.
- Date Range parameters: Required in generating a calendar view of appointment lists (e.g. daily view, weekly view and monthly view of appointment listings).

- Appointment parameters: Buffers to store the parameters of an appointment/ meeting record. Used in creating a new appointment or in updating an existing appointment/ meeting.
- Attendees Information: Buffers to store the attendee names, status, states and optional information in a meeting. Used in creating or updating an exisitng meeting.
- Notice Processing parameters: Include the meeting notice type as well as the response to the notice. Used in processing a particular meeting notice.

Not all the parameters in the request object are set in a request. Only some of the parameters are filled in a particular request, and other parameters can be left blank. This request object is passed to the session management process in a request through the communication channel, and the session management process can expect the same standard object to receive every time. This simplifies the communication between the two parties.

After the session management process has performed the session or database operations, a response will be passed to the CGI process. Again, in order to simplify the communication, the response is standardized. At the beginning of every response, there is a success/ error code. It is simply a number indicating whether the operation is successful or not. If it is unsuccessful, the number also indicates what the error is. For many request cases, the response just consists of this success/ error code. For requests of a calendar view with an appointment listing or requests of an edit appointment form with fields preset with attributes of an existing appointment, a stream of strings for an HTML document section will follow that success/ error code in the response, if the

Type of Request	Parameters Required	Response Expected
session logon	session ID, remote host, login parameters	success/ error code
session logout	session ID, remote host	success/ error code
create appointment	session ID, remote host, appointment parameters	success/ error code
create group meeting	session ID, remote host, appointment parameters, attendee information	success/ error code
delete appointment	session ID, remote host, appointment ID	success/ error code
update appointment	session ID, remote host, appointment ID	success/ error code
list calendar appointments	session ID, remote host, date range information	success/ error code + stream of strings for an HTML section to show appointment lists
generate edit appointment form	session ID, remote host, appointment ID	success/ error code + stream of strings for an HTML section to show an edit appointment form
generate edit group meeting form	session ID, remote host, meeting ID	success/ error code + stream of strings for an HTML section to show an edit meeting form (w/ VBScripts for the controls in the form)
list meeting notices in mailbox	session ID, remote host	success/ error code + stream of strings for an HTML section to show a list of unprocessed meeting notices in the mailbox
list the details of a meeting notice	session ID, remote host, meeting notice ID	success/ error code + stream of strings for an HTML section to show the details of the meeting notice and the links to process it
process a meeting notice	session ID, remote host, meeting notice ID, notice processing parameters	success/ error code

Table 6.1 List of requests, and the corresponding request parameters required and responses expected between the communication between the CGI process and the session management process

code indicates the operation is successful. So in all responses, the CGI process expects that there must be a success/ error code at the beginning of a response. Depending on the request type, there may be a stream of strings following this code. The CGI process can recognize the end of the stream of strings has been reached if it gets an end-of-response string or an error string.

Table 6.1 shows a list of different requests, the request parameters required in them and the type of responses expected.

6.2.3 Session Management and Request Authentication

The session management process keeps a session table. It keeps the session information for different users on the client side. A number of fields can be found in this session table:

- Session ID: For session authentication.
- Remote host of the client at login time: For session authentication. Avoid a cracker from stealing a session ID and use it from a different client machine to access the same Organizer database.
- User name of the session: For display in responses.
- Organizer API handle: For accessing the Organizer databases through the Organizer API layer.
- Organizer database handle: For accessing a particular Organizer database file.
- Last active time of the session: For removing inactive session after a certain time-out period.

In a request for session login, a new session entry will be created in the session table, except when the maximum number of session entries of the table has been reached. In subsequent requests, all the fields in the session entry will stay constant, except that the last active time of the session is updated in every request. Moreover, a last active time for all the sessions in the table is also kept to facilitate automatic shutdown of the session manager. This is updated whenever there is an operation performed in a session or whenever a session entry is removed from the table.

With the Organizer API handle and Organizer database handle kept in the session table, the user only needs to obtain the handles once in a C&S session at login time, and he or she can use these handles to access the database throughout the C&S session. This helps to achieve the design goal of the session management system to minimize the slow API session and file operations.

Moreover, the session ID and the remote host name of the client machine at login time can promote security in the protocol. As described in Chapter 3, the session ID used in the protocol is a computer-generated Universal Unique ID (UUID) in the Windows NT system, which is independent of the usernames and passwords the user supplies. In every C&S request (except at login times), the client must supply the session ID, which is matched against the session IDs in the session entries in this session table. If the session ID does not exist in the table, session authentication will fail. After this session ID checking, the remote host name of the requests will also be checked against the remote host name of the client machine when he or she first logged in to the current session. This makes sure that the client is making requests from the same client machine throughout the C&S session, avoiding a cracker from stealing a session ID and using it to access a database from another client machine.

It is common that a C&S client forget to logout a session after he or she has finished using the C&S service. This leads to the session entry being kept in the session table forever. Thus the last active time for a session entry is kept in the session table, and the session management process checks these times at regular intervals. If the process finds out that a session has been inactive for a period longer than a certain "time-out" period, it can delete that session entry from the session table, and close the associated API and file handles. This avoids the session table being filled with inactive sessions. Moreover, this protects the privacy of the user's calendar information even when the user forgets to login out a C&S session.

6.2.4 Startup and Shutdown of Session Management Process

The session management process can be manually started up and shut down on the Web server, but this task can be performed more intelligently and efficiently. The session management process can be designed to establish communication channels with the CGI process right after the process is started (This can be achieved by the named pipes mechanism in Windows NT system. See Section 6.3.2 for more details). The session management process will listen to these channels and wait for CGI processes to connect.

The startup of the session management process may be done by the CGI process. When a client tries to login a C&S session, the CGI process first tries to look for a communication channel to connect. If there is no channel available, it means that the session management process has not been started. Therefore, the CGI process will start the session manager, wait for the communication channels to be established and connect to it to create a session. If a communication channel is available when the CGI process tries to connect one, that means that the session management process has been started, so the CGI process can simply connect to one of the communication channels and continue the session or database operations. For non-login requests, the session manager is supposed to be running (since it should have been started in the login request) and the above checking procedure is not necessary. If communication channels are not available in non-login cases, either all the channels have been occupied or there is an error in the session management process. The flow chart in Figure 6.2 shows how the CGI process determines when to start the session management process.

The session manager also checks whether there is no active session in the session table at regular intervals. If it finds out that there is no active session in the table, and the amount of time from the last active time of the session table has exceeded a certain "time-out" period, the session management process will automatically shut down itself. Note that the last active time of the session table is actually the time when the last session entry is removed from the session table. That means if the session manager finds out that there are no clients logging in to a C&S session for a certain amount of time, it could shut itself down. When a client logs in to a session later after the shutdown, the session manager will be started up once again as described above.

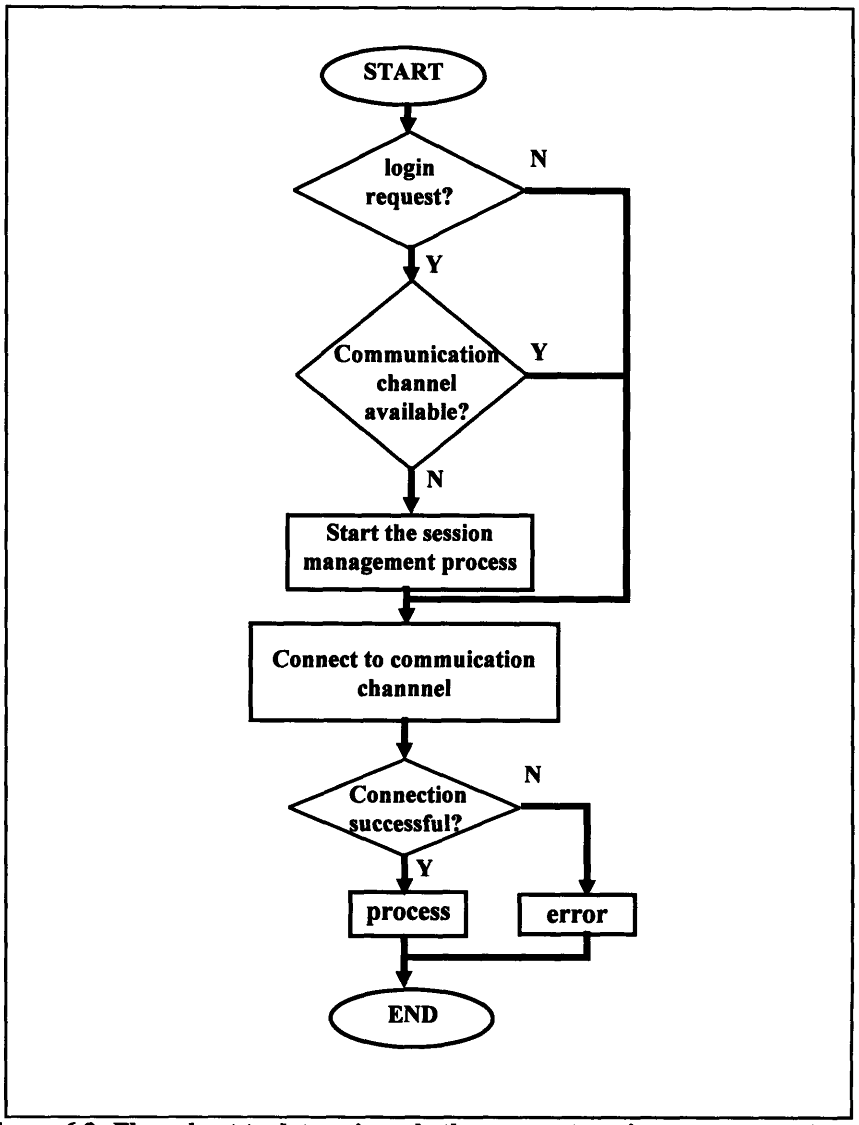

START
login request?
N
Y
Communication channel available?
Y
N
Start the session management process
Connect to commuication channnel
Connection successful?
N
Y
process
error
END

Figure 6.2: Flow chart to determine whether to start session management process in CGI process

This intelligent startup and shutdown of the session manager saves the Web server administrator some efforts in managing the session manager process, as the administrator does not need to check when the session manager should be started and when it should be shut down. All the operations are done automatically.

6.3 Implementation Details

The session manager is multi-threaded [11][12], so more than one instance of the CGI process can be served at any time. As a result, even when there are several clients making requests simultaneously, and several instances of the CGI process have been invoked to talk to the session manager, the session manager can still serve the different instances of the CGI process independently through different threads. In the current implementation, there is an instance of the communication channel between the CGI process and the session manager within each thread. When the session manager is started, a number of threads will be started at once, and thus the same number of communication channels are established. These channels will wait for the CGI process instances to connect to them all the time.Once a CGI process instance has connected to a channel, communication between the two processes starts.

Next the communication channels between the CGI process and the session manager need to be constructed. There are three popular mechanisms for interprocess communication in Windows NT: mailslot, memory mapped files and named pipes [10][12]. Mailslots are used in communication among several processes, which are not suitable in our case. In a memory mapped file, a file of a specified name can map itself into a process's address space, and it is possible for multiple processes to access the same mapped file by name, and in that way share data with one another [10]. It is also possible to share memory, without using an actual file on the hard drive, by using the system's paging file. The communication channel can be implemented with a memory mapped file, with the requests and responses made on the shared memory of the mapped file. As the session management process has been multithreaded, the address space of the mapped file has to be divided into a number of sections. Each section of the memory is allocated to one of the process threads for communication between the process thread and the CGI process.

Another scheme is to use a named pipe [12]. A process can set up a named pipe and listen for connections to it. Multiple instances of the same named pipe can be created to allow different processes to connect to the named pipe independently. Therefore, the communication channel can be implemented by simply setting up different named pipe instances in each of the session management process threads. Different instances of the CGI process can then connect to different instances of the named pipe for communication with the session manager.

The advantage of using the memory mapped file is that it is faster than named pipes [10]. No actual connections are made between the processes, and operations are simply made on shared memory space. The disadvantage is that the size of the memory space assigned for a session manger process thread is limited. It is possible that there is not enough memory in the assigned area to accommodate a huge response, unless some manipulations on the shared memory have been made to allow passing a response in several pieces. On the other hand, a named pipe has no limitations on the size of the requests and responses. Once a connection has been set up between the processes, requests and responses can be made within that connection in a very straightforward manner. As a result, the named pipes scheme is used to implement the communication channel between the session management process and the CGI process.

Chapter 7 User Interface Enhancement

7.1 Introduction

7.1.1 User Interface Problems in HTML

Traditionally, C&S applications have very rich and interactive user interface elements, allowing users to access the C&S functionality easily and efficiently. Nevertheless, moving these rich UI elements to Web pages can be a difficult task. Standard HTML only supports simple input controls, and Web pages created in HTML are usually pretty dull and static. In HTML 2.0, only the IMG tag is defined for inserting image media into HTML documents, but there is no support for other richer media [13]. The mappings of typical Windows UI elements such as tabs, list boxes and sliders are often impossible in HTML. This limitation in HTML severely affects the uses of a C&S application on the internet.

Specific to the C&S model used in the OrgWeb system, several features are particularly difficult to implement in simple HTML with reasonably good performance. One of them is the free time display for attendees in scheduling a group meeting. As seen in Section 4.2, accessing the free time information for an attendee when the attendee is inserted into the attendee list requires an extra request to the server. The server responds to the client with the free time information, and the client side should display the information in the browser. With static HTML, the request and response means a new

Web page created by the server and loaded into the browser, with the free time information display updated but all other HTML content and input values on the previous page staying the same. When the scheduling user adds another user into the attendee list, or moves the tentative meeting time to a time beyond the range of the current free time display, this process repeats.

Two inefficiency problems arise in this kind of free time display process with static HTML. First, in every update of the free time display, a new HTML response needs to be generated by the OrgWeb server. Yet this new response is only different from the previous response in terms of the free time display, and there are no changes in other HTML contents or form input values. This is fairly inefficient, and the frequent generations of the HTML responses substantially adds to the workload of the server. Secondly, since there are no changes in the form input values from the previous response to the new one, the client needs to pass all the form input values from the previous page to the server when the free time display is updated, so that the server can preserve and display these values in the new response. This complicates the protocol definition as information unrelated to the free time search needs to be supplied in the request commands.

Apart from the problems in free time display, the lack of interactivity among the input controls in HTML also makes the UI design difficult for internet C&S applications. Form input controls in an HTML page are isolated from each other. Complicated user actions such as editing the attendee list may be hard to implement on a static HTML form. Imagine how difficult it is to provide functionality such as adding and removing attendees in the attendee list, or setting and viewing the attendee attributes (e.g. Status and Optional/ Required information), with just the standard HTML input controls such as text edit boxes, SELECT controls or RADIO controls. With standard HTML controls, such user actions will require more protocol requests to continously update the browser view, making the protocol more complicated and reducing the performance of the C&S system[2].

Unless the user interface of the C&S responses are enhanced and made more interactive, the HTTP-based C&S protocol definition cannot be kept simple, and the protocol cannot be used efficiently.

7.1.2 Solution: Objects and Scripting in HTML

In view of the problems caused by the static properties of HTML, developers have been experimenting with ideas to activate Web pages. The most common solutions are inserting objects and inline scripting in HTML pages.

Objects are usually pre-compiled executables that can be run within an HTML page inside the browsers. These objects are distinct from HTML, and are downloaded from the server when an HTML page is downloaded. Special browsers or software may be required on the client side to run these executables in the browsers [15][16][17].

By inserting objects in HTML small applications can be run inside the browsers, and they can perform functionality that is beyond the limits of HTML. For example, we can develop a free time viewer object that can be placed in an HTML page, in which the object is capable of accepting names of attendees, making OrgWeb requests to fetch

[2] The demo of the Web Edition of Cambell Services' ONTIME uses simple HTML pages and forms to do internet scheduling [14]. When tested in a meeting creation with two attendees, more than 10 separate HTTP requests are made in the demo.

the free time information for the attendees and displaying them to the Web clients. Common object architectures in the marketplace now includes Netscape's Navigator plug-ins, Sun Microsystem's Java applets and Microsoft's ActiveX controls. The World Wide Web Consortium is also proposing a new HTML tag "OBJECT" to extend HTML to support object insertions [13].

Inline scripting allows internet application developers to embed scripts directly in an HTML page, using the scripts to automate and manipulate the various objects and controls on the page [18][19]. These embedded scripts can also recognize and respond to user events such as mouse clicks and keyboard inputs, and can read and set the values or properties of the objects or controls on the page. Note that these scripts are interpreted by runtime engines of the browsers on the client side after they are downloaded with the HTML page, and they are usually platform independent and portable. With these inline scripts, the objects and the form input controls in HTML pages can now respond to user actions by running scripts. Communications and interactions among different objects and input controls are also now possible.

Currently the most popular scripting languages for internet application development are Sun Microsystem's JavaScript and Microsoft's Visual Basic Script (VBScript).

7.2 Design Considerations

7.2.1 Object Type

Although object insertion in HTML pages can solve the UI problems in HTML, on the internet there is currently no standard architecture for building and using objects in HTML. A type of object may be supported in a certain Web browser, but may not be supported in another one. That means that a Web page carrying an object may not be universally usable for all Web clients. Also different object types have different building processes, properties and procedures in using them. As a result, when objects are developed for the OrgWeb C&S system, a number of issues should be considered.

Browser Support

As pointed out in Section 3.1, one of the design goals in the C&S protocol is to make the protocol ready and easy to use in the existing environment. According to various statistics on Web browser usage, currently more than 90% of internet users are using Netscape's Navigator or Microsoft's Internet Explorer [20][21]. Thus the internet C&S objects should be at least supported by these two browsers to meet the design goal.

Those most popular object types all seem to satisfy this requirement. Internet Explorer 3.0 supports Java applets, Navigator plug-ins and ActiveX controls. Navigator 2.0 supports Java applets and Navigator plug-ins directly, and supports ActiveX controls indirectly with a plug-in[3] which can view ActiveX controls.

[3] ActiveX plugin developed by NCompass Laboratory

Platform Support

In a client/ server application such as the internet C&S system, ideally there should be no constraints on which platform to use on both the client side and the server side in the system.

None of the most popular object types satisfy this requirement. Currently Java applets are the best in this area, and it is supported by Netscape's Navigator on a number of major platforms[4]. Navigator plug-ins are specific to the platform it has been developed for. Different versions of plug-in should be installed on different platforms, and the user on the client side must specify the perform he/ she is using before the plug-in is installed. Since an ActiveX control is based on the former OLE control technologies, it is only supported on Windows-based platforms.

Development Difficulty

Another consideration is the difficulty involved in developing an object.

For Java applets, they are small applications coded in a new language, Java, and their development requires new tools [16]. As a result, considerable amount of time and effort may be needed in building applets for C&S applications, since software reuse is hardly possible as most of the UI elements in existing C&S system were not developed in Java.

Navigator plug-ins can be general application modules (in the format of .DLL files) written in C or C++ [17], and ActiveX controls are similar to OLE controls which can be built in C++ [15]. Both types of objects can be developed using existing

[4] Java applets are supported on Sun Solaris, Sun OS, SGI IRIX, OSF/1, HP-UX, Macintosh, Unix (IBM AIX, BSDI), Windows 3.1, Windows NT and Windows 95 in Navigator 3.0 (Beta 6) [22].

development tools, making it easier to leverage existing code. Code that has already been written for the user interface of existing non-internet C&S systems may be reused to build plug-ins and ActiveX controls in the OrgWeb system, saving lots of time and development effort.

Plug-ins are installed on a client's machine to display objects with application specific MIME types [17]. Different objects may require different plug-ins. For example, in the C&S system if the free time viewer and the attendee list box are desired, two different plug-ins and two different MIME-type objects need to be developed. When more different types of UI elements are wanted, more plug-ins need to be built and installed. Otherwise we may need to build a single plug-in that is powerful enough to handle all C&S-related MIME-types, but that may be a difficult task. On the other hand, all ActiveX controls have a common standard - they are all based on the former OLE control technologies, and they are all .OCX files. Different UI elements can be made into different types of OCX's, with the common ActiveX architecture and interfaces to be used in a browser.

Ease of Use

The ease of using the C&S objects in browsers is also an important factor in deciding which type of objects should be built for the OrgWeb system.

Using a Java applet involves a special HTML tag to reference the applet. When a Java-enabled browser loads a Web page with that applet, the browser downloads the applet from the Web server and executes it on the client's local system [16]. No special installation procedure is required for the applets, but the applets are downloaded every time when the Web page is loaded and the applets are not saved on the client machine[5].

Navigator plug-ins work by installing a dynamic link library (DLL) into a directory on the client machine, which can display and handle application specific MIME objects (e.g. in the internet C&S system, an object with a new MIME type, which can be displayed by a new plug-in, can be developed for viewing free time information) [17]. When an object whose MIME type is not familiar is encountered by the browser, special Web pages will be returned, on which there is information and links to install the plug-in to handle that MIME object (in "assisted installation" in Netscape Navigator 2.0). Once the plug-in gets installed, all Web pages containing objects with that new MIME types can be displayed without further downloads of the plug-in. Note that the end-users are required to do the plug-in installation explicitly, and in the current implementation of Netscape Navigator, after the installation the users need to quit and restart the browser to use the plug-in.

Installation is also needed for ActiveX controls, but from a user's perspective, the process is fairly transparent. When a page with an ActiveX control is loaded, the browser (or the plug-in to run the ActiveX control) first checks the client's local system to see if the control is already available. If so, the control will be fetched from the local system and displayed within the Web page. If not, the control will be downloaded from the Web server, and will be automatically installed on the client's machine [15]. Since the

[5] Major Web browsers usually cache HTML page elements. In that case, Java applets are not necessarily downloaded from the server whenever it is used, if the applets are in the cache and have not been flushed from the cache. However this is more "volatile" when compared to plug-ins and ActiveX controls which are installed permanently on the local machine unless the user uninstalls them explicitly.

installation is automatic, end-users need do nothing more than just download the Web page. Moreover, as the ActiveX controls are saved on the client machine after the first download, subsequent uses of the controls need no further component downloads and is faster.

Conclusion

Considering all the above issues, ActiveX controls seem to be the most suitable type of objects to develop for the OrgWeb C&S system. Java applets require development of the objects from scratch in a new language, and downloading the applets every time when Web pages are loaded affects the overall performance. Navigator's plug-ins involve an explicit installation process on the client side, which may interrupt the C&S service for first-time users. Moreover, though software reuse is possible in developing plug-ins, new MIME-type objects need to be designed for existing UI elements. For ActiveX controls, the restriction that they can only be developed and used on Windows-based platforms is a major problem from a client/ server design's point of view. However, the possibility of leveraging existing code for UI elements and its one-time automatic installation process are attractive to object development and usage. Later it will be shown that converting a UI element in an existing C&S system into an ActiveX control may be easier than creating a new MIME object for the element and the plug-in to display it (Section 7.2.2).

7.2.2 Building the Components

In building objects for UI elements in the OrgWeb C&S system, ActiveX controls which can perform functionality similar to the UI elements in Lotus Organizer are developed. Like many other Windows-based applications, the UI elements in Organizer are implemented as Windows controls, and now they need to be converted into ActiveX controls.

There are two approaches to do the conversion. The first approach is to build a completely new ActiveX control, and then move all the functionality in the existing Windows control to the new control. Yet this is very time-consuming, especially when the UI elements in Organizer are fairly complex.

The other approach is to reuse the existing Windows control in Organizer, and an ActiveX control "wrapper" is added to the Windows control. Important parameters in the Windows control are exposed as OLE "properties" in the control. Moreover, Windows user messages (WM_COMMANDs) sent to the Windows control for different operations on the control are mapped to different "methods" of the ActiveX control. Furthermore, the notification messages sent from the Windows control to its parent window are mapped as "events" of the ActiveX control. By adding this ActiveX control "wrapper", existing code of the Windows controls in Organizer can be easily leveraged, and the Windows controls can be converted into ActiveX controls. This "wrapper" approach is illustrated in Figure 7.1, and the details of its implementation will be presented in Section 7.3.1.

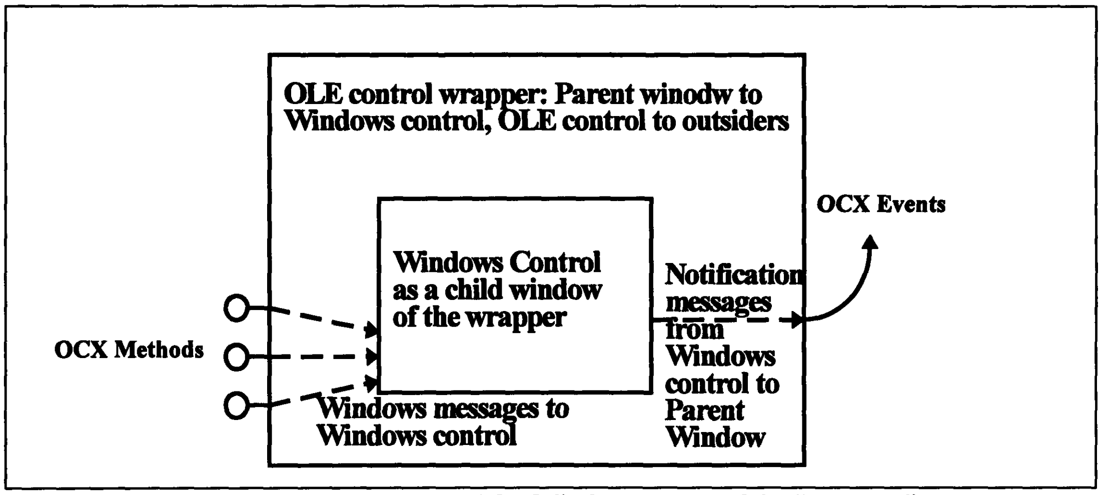

OLE control wrapper: Parent winodw to Windows control, OLE control to outsiders
OCX Events
Windows Control as a child window of the wrapper
Notification messages from Windows control to Parent Window
OCX Methods
Windows messages to Windows control

Figure 7.1: ActiveX Control "Wrapper" for Windows control. The "wrapper" appears as a Windows parent to the Windows control, but to outsiders it is an OLE control.

Since the ActiveX controls created in the "wrapper" approach are using the existing Windows controls in Organizer, those DLLs which define the Windows control classes and their functionality need to be present when the ActiveX controls are used in a browser on the client machine. Thus those DLLs should be downloaded at the same time when the ActiveX components are downloaded [23].

7.2.3 Use of Scripts

As mentioned in Section 7.1.2, scripts in HTML pages can make the Web pages in the C&S system more interactive. Also scripts are required to manipulate the ActiveX controls inserted in the HTML pages, so that the OLE methods of the OCX can be used, and the browser can receive and respond to the OLE events from the controls.

The scripts are usually platform independent, and are embedded within HTML pages. The scripts are interpreted by the browsers after they are downloaded with the HTML documents [18][19]. As a result, the scripts can be generated on the fly by the Web server, which can perform functions specific to a particular request.

In the OrgWeb system, the scripts are used in three main areas:

1. Initializing ActiveX Controls: Before a user can use the free time display and the attendee list edit ActiveX controls to create or edit a group meeting, the controls need to be initialized with the chairperson and the attendee list information. This information varies from one request to another (in each different OrgWeb session the chairperson is different, and in each different meeting the attendee list is different) , so this data must be generated within the HTML page on the fly when the server is responding to the client. In the current proposed implementation of the OBJECT tag which can be used to insert ActiveX controls in HTML, the tag parameters (the PARAM attribute) can only initialize the exposed properties of the controls [13]. This may force the control developers to expose some internal parameters of the controls in order to initialize them with the OBJECT tag. However, with the use of scripts the Web server can generate initialization scripts on the fly to initialize the ActiveX controls by accessing the controls' OLE methods. This can avoid undesirable exposures of some of the internal parameters in the controls.
2. Providing interactivity among controls: The scripts can respond to user actions, and events generated from the ActiveX controls or the form input controls. These events can trigger routines in the scripts which can perform operations on other controls [24]. As a result, the controls on the HTML page can communicate with each other, giving a more interactive user interface to the HTML page.
3. Collecting information from ActiveX controls for making requests: When an HTML form is submitted to make an OrgWeb request, the scripts can call the methods of the ActiveX controls to access the current states of the control, and format this state

information into form input values. In this manner, the data which is set within the ActiveX controls by the user can be passed to the server to make C&S requests.

As for which scripting language to use, both JavaScripts and VBScripts can be used to manipulate ActiveX controls, and both can perform the above functions required in the OrgWeb system. In the current implementation of the OrgWeb system, VBScript has been used.

7.3 Implementation

7.3.1 Building the ActiveX Components

As seen in Section 7.2.2, we can quickly convert an existing Windows control of an Organizer UI element into an ActiveX control that can be used in HTML pages, by wrapping the Windows control with an ActiveX "wrapper" layer. In this section the procedures on how to wrap the Windows controls are discussed.

Creating the Wrapper Layer

The wrapper layer is actually an ActiveX control. In Microsoft Foundation Class (MFC), an ActiveX control object belongs to the class *COleControl*[6], which is a subclass of *CWnd*, the base class for window classes [25]. That means that an ActiveX control is also a window by itself.

To include an existing Windows control in the "wrapper" ActiveX control, a new windows class is derived from the windows class of the Windows control. A window

[6] ActiveX control is formerly named OLE control.

object is created with this new windows class, and the object is a child window of the wrapper ActiveX control. Also the child window is made to be a protected member of the wrapper control so that the wrapper can access its functions. Suppose the wrapper ActiveX control class is *CWrapper*, and the new windows class is *CNewAppWindow*, this can be done as follows:

```
class CWrapper : public COleControl
{
      . . . .
protected:
      CNewAppWindow*    ChildWindow;
}
```

In the *CNewAppWindow* class, there should be a function to create a window object of the window class, and make the new object be a child window of the window calling the function. Here is an example:

```
class CNewAppWindow : public CWnd
{
      . . . .
public:
      void CNewWndCreate( CWnd* Parent, UINT uiID, CRect r )
      {
            Create( "NewWindow", "New Window Control",
                        WS_CHILD | WS_VISIBLE, r, Parent, uiID );
      }
}
```

NewWindow is the windows class of the old Windows control. The wrapper control can then call this function to create a child window for itself:

```
void CWrapper::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
      CRect r;
      . . . .
      ChildWindow->CNewWndCreate( this, CW_ID, r );
      . . . .
}
```

where CW_ID is the ID for the child window.

In this way, a wrapper layer can by created around the old Windows control, using the control as a child window control. To outsiders, the wrapper layer is an ActiveX control which can be inserted in HTML pages. To the old Windows control the wrapper layer is its parent window, which can accept the notification messages it sends.

Mapping the Methods

An ActiveX control container uses the methods provided by the control to operate on the control and change its states. In wrapping a Windows control into an OCX, each method in the ActiveX wrapper layer is mapped to a message sent to the wrapped child Windows control. Continuing with the above example, suppose there is a method call *reset*, which resets the state within the wrapped Windows control:

```
void CWrapper::Reset()
{
      return( ChildWindow->SendMessage( CWM_RESET,
                 (WPARAM) 0, (LPARAM) 0 );
}
```

CWM_RESET is the message that is sent to the wrapped Windows control to tell the control to reset its states. Other methods can be mapped in a similar manner to provide access to operate on the Windows control.

Mapping the Events

ActiveX controls use events to notify a container that something has happened to a control. Commonly an event is caused by some user interaction, such as mouse input within the control's client area. When such an action occurs, the control alerts the container by firing an event [25]. Using the ActiveX control in a browser, the event is received by the browser, which is the container of the OCX in this case.

Windows controls have analogous behaviors. They can notify their parent Windows that something has happened to the control by sending a "notification" Windows message to their parent Windows. In the wrapper ActiveX layer, these "notification" Windows messages are mapped to OCX events. Suppose in the wrapped Windows control there is a notification message called *StateChanged*, which notifies the parent that a state change has occurred within the control. In the message map of the wrapper ActiveX layer, there should be a mapping for the notification windows message to an OCX event:

```
BEGIN_MESSAGE_MAP( CWrapper, COleControl )
        . . . .
        ON_CONTROL( CWN_STATECHANGED, CW_ID, FireStateChanged )
        . . . .
END_MESSAGE_MAP()
```

where CWN_STATECHANGED is the ID for the notification message sent to the wrapper layer, and *FireStateChanged* is the function to fire off the corresponding OCX event.

By following the above procedures to create an ActiveX wrapper for an existing Windows control, the Windows control can be turned into an ActiveX control that can be used in internet scheduling.

7.3.2 Applications in Internet Scheduling

In the OrgWeb C&S system, three different ActiveX controls have been created to facilitate more efficient internet scheduling user interface, and to experiment with the idea of using ActiveX controls for internet scheduling. They are the free time viewer control, the attendee list box control and the date tracker control.

Free Time Viewer Control

The free time viewer control allows the scheduling user to see the free time of other attendees, and to choose a free time slot that is convenient for everyone. As a meeting chairperson adds a new invitee into the attendee list of a meeting, the name of the attendee is displayed in a vertical time bar in the viewer. The information about the attendee's status and free/ busy time is also shown in the time bar. Status is represented by various bitmap images, and busy time periods are displayed graphically in rectangular blocks. A time scale is shown beside the vertical time columns, and on the time scale there is a time tracker to allow users to set the tentative meeting start time and duration.

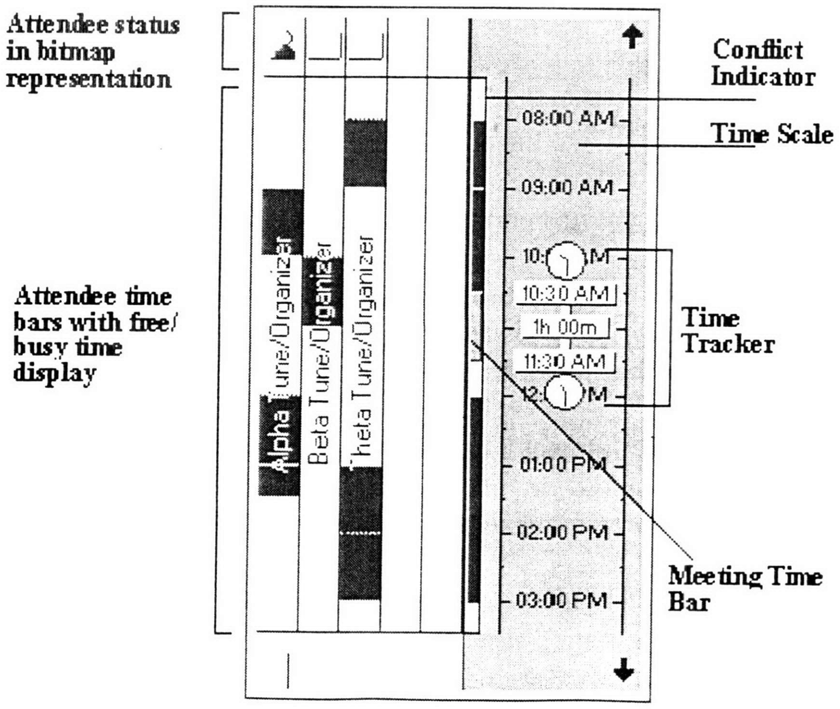


Figure 7.2: The Free Time Viewer control

There is a conflict indicator between the time bars and the time scale. Along the indicator there is a meeting time bar, which represents the meeting duration in the indicator. This meeting time bar changes in position and in length as the scheduling user changes the meeting time and duration. The meeting time bar indicates whether or not there is a conflict between the tentative meeting time and the schedules of the attendees, by means of showing different colors in the time bar. Thus the scheduling user can easily avoid schedule conflicts and choose the best time for all attendees. Figure 7.2 shows the free time viewer control.

In accessing the free time information for the attendees as they are inserted in the free time viewer, requests have to be made to the OrgWeb server which gets the information from the attendees' databases. To avoid loading a new Web page with all things being the same as the previous page but the updated free time display, the free time viewer OCX is designed to be able to make OrgWeb free time search requests *within* the OCX. When attendees are inserted into the control and the free time viewer needs to get updated, the OCX will make an internet connection to the OrgWeb server, making HTTP requests for free time search for the attendees, getting the HTTP responses and parsing the responses into busy time blocks in the free time viewer. The internet connections and HTTP calls are made through the Internet Extension of Windows API (WinInet API), which allows Windows applications to make transparent internet access without a browser middleman [26]. With this function, no additional HTML pages need to be downloaded when the free time viewer gets updated, making the process of choosing a meeting time more efficient.

Through WinInet API calls, the free time OCX makes an OrgWeb request[7] to the OrgWeb server, asking for the busy time periods for one or more users within a period of time. On the server side, the request is no difference from a normal OrgWeb request made from a Web browser, and the server returns the free/ busy time information in an HTML document. The OCX then parses the HTML response into busy time period data, which are inserted in the free time viewer. Note that before the internet connection between the OCX and the OrgWeb server can be established, the OCX needs to know the URL of the server, and possibly also proxy server it should talk to first. Thus the OCX should be initialized with this information before it is used. This can be done by the inline scripts embedded in the HTML response. If the client side is using a proxy server to access the internet, the client should get the name of the proxy server from the registry, and specify it in the OCX through an OCX property before the OCX is actually used.

To further improve the performance of the free time search service, the way in which the OrgWeb free time search requests in the OCX are made is carefully designed. There are two time ranges involved in making the free time search: the "search" range and the "safe" range. The "search" range starts at 96 hours before the tentative meeting start time, and ends at 96 hours after the tentative meeting end time. The "safe" range starts at 72 hours before the tentative start time, and ends 72 hours after the end time. When a scheduling user has set a tentative meeting time and starts adding people into the attendee list, free time information will be searched within the "search" range. Then the user can start looking for a meeting time that is good for everyone with the free time

[7] The "busytime" command

information. If, unfortunately, the user cannot find a good time within the "safe" range, and moves the tentative meeting time to a time outside the "safe" range, a new free time search will be performed within the "search" range of the new tentative time[8]. As the scheduling user is likely to find a convenient time for all attendees within a period not too far away (in this case within 72 hours) from the originally tentative time, usually a single free time search at the beginning gives sufficient free time information to the user in scheduling the meeting. By this scheme, the number of OrgWeb requests is minimized, promoting the performance and efficiency of the free time search service.

Attendee List Box Control

The attendee list box control is used to show an updated list of attendees in a meeting when the scheduling user is in the process of editing the attendee list. The underlying Windows control is just a simple list box with string entries.

When the scheduling user changes the attendee selection in the list box, an event is fired off from the OCX to the OCX container. The OLE container in turn can use a method in the OCX to get the updated selected attendee in the control. When the control is used in a browser, this enables the browser to update the display of the attendee attributes for the selected attendee by means of inline HTML scripts. Details on how to achieve this is discussed in Section 7.3.3.

An example of an attendee list box is shown in Figure 7.3.

[8] The "search" range is wider than the "safe" range because the scheduling user would like to view the free time in periods before and after the tentative meeting time.

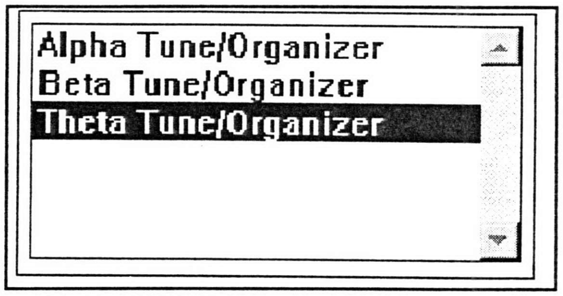


Figure 7.3: The Attendee List Box control

Date Tracker Control

The date tracker control is a "wrapper" type ActiveX control derived from the date tracker Windows control in Lotus Organizer. It allows a scheduling user to choose a date for a meeting easily by mouse-clicking a date from a monthly calendar control, which is a drop down from a combo box. Alternatively the user can type in the date in the text box of the combo box. The date tracker control will make sure the date typed in is in correct format and is a valid date. If the date typed in is not valid, the date of the control will be reset to the date before the new date is entered.

Figure 7.4 shows the date tracker control.

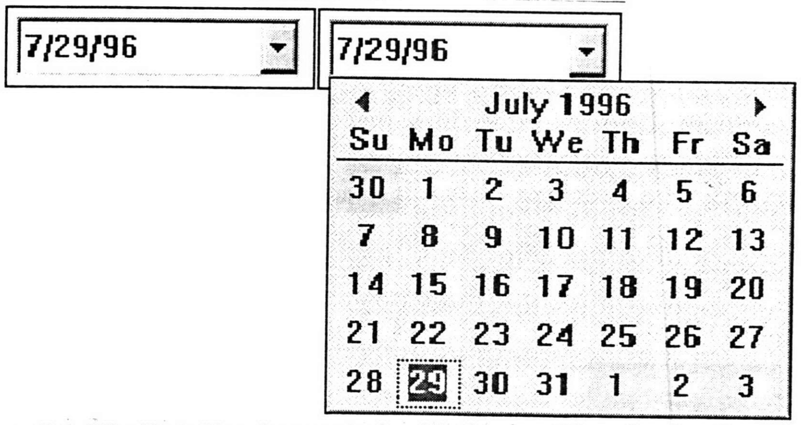
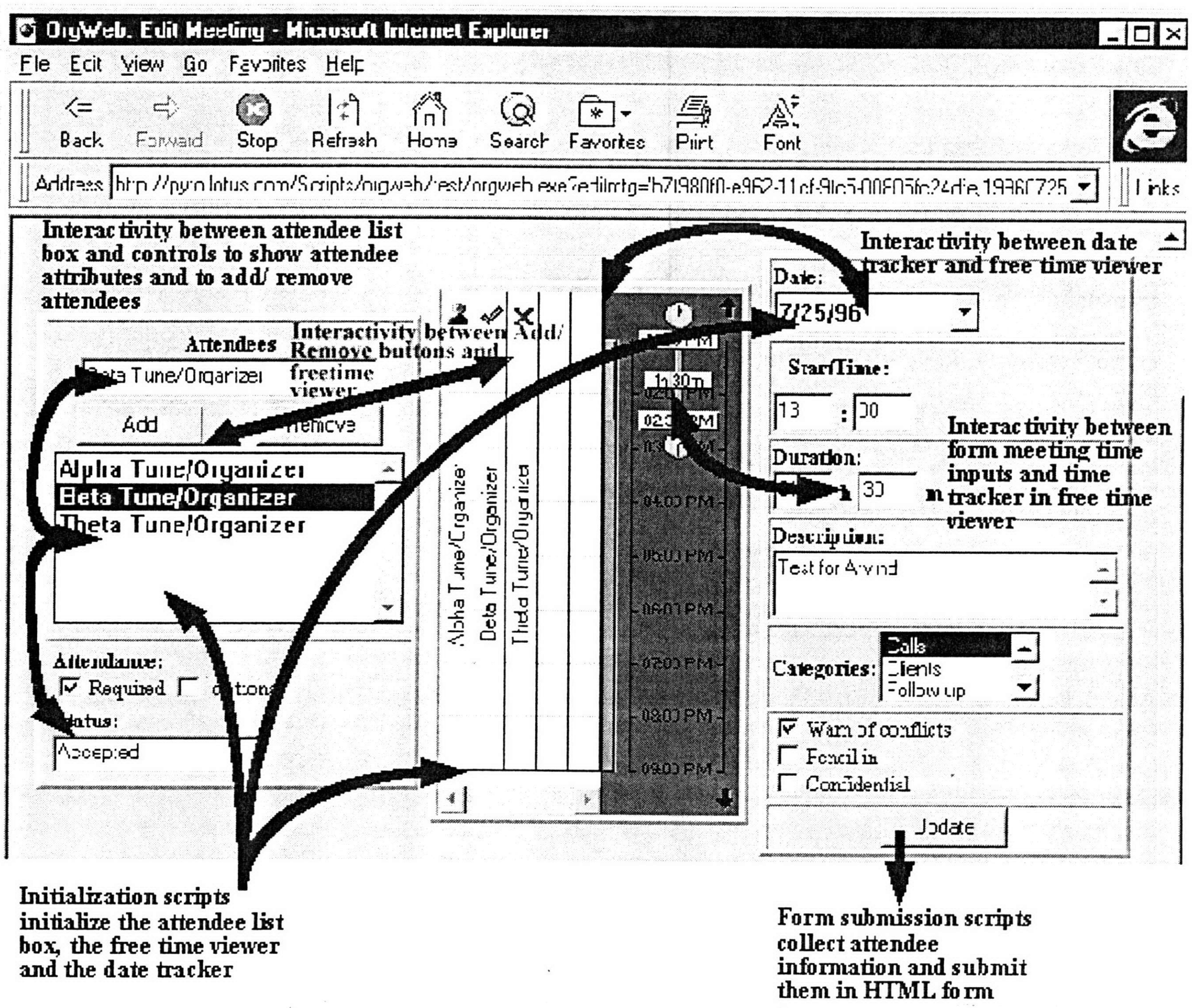


Figure 7.4: The Date Tracker control, with the monthly calendar drop down hidden (Left) and shown (Right)

7.3.3 HTML Scripts and Using the Components

The C&S ActiveX controls in the HTML pages do not work by themselves. Scripts embedded in the HTML documents are required to manipulate the controls to perform useful functionality. Figure 7.5 shows a snap shot of an "Edit meeting" page in the OrgWeb system, which summarizes how the scripts are used in manipulating the C&S ActiveX controls, and how the controls interact with each other.



Address: http://pyro.lotus.com/Scripts/orgweb/test/orgweb.exe?editmtg=h7l980f0-e962-11cf-9lc5-00F05fc24die,1936C725

Interactivity between attendee list box and controls to show attendee attributes and to add/ remove attendees

Interactivity between Add/ Remove buttons and freetime viewer

Interactivity between date tracker and free time viewer

Interactivity between form meeting time inputs and time tracker in free time viewer

Initialization scripts initialize the attendee list box, the free time viewer and the date tracker

Form submission scripts collect attendee information and submit them in HTML form

Attendees: Beta Tune/Organizer

Alpha Tune/Organizer
Beta Tune/Organizer
Theta Tune/Organizer

Attendance: Required ✓

Status: Accepted

Date: 7/25/96

StartTime: 13 : 30

Duration: 30

Description: Test for Arvind

Categories: Calls, Clients, Follow up

Warn of conflicts ✓

Figure 7.5: Use of scripts in the "Edit Meeting Page"

First of all, the scripts are responsible for initializing the attendee list, attendee list box and the free time viewer. They also set the meeting date in the date tracker.

The scripts also facilitate interactivity among the various controls in the HTML page. Changes made in the form date and time input controls are updated in the free time viewer and its time tracker, and changes made in the free time viewer are also reflected in the form date time input controls. The attendee list box interacts with the attendee attributes controls, the attendee text box as well as the attendee "Add" and "Remove" buttons. For example, in adding an attendee into the list box by clicking the "Add" button, the scripts look like this:

```
' AttAddBtn is the id for the "Add" button
' Attedit is the id for the attendee edit text box
' AttList is the id for the attendee list box
' freetime is the id for the free time viewer
Sub AttAddBtn_OnClick()
      ' Check attendee edit text box is not empty
      If Len(AttEdit.Value) > 0 then
            ' Adding the attendee into the attendee list box
            If AttList.AddItem(AttEdit.Value) <> -1 then
                  . . . .
                  ' Adding person in free time viewer and set the status
                  freetime.AddPerson Att(NumAtt, 0), AttEdit.Value
                  freetime.SetStatus Att(NumAtt, 0), 1
                  . . . .
            End If
      End If
End Sub
```

When the "Add" button is clicked, the subroutine *AttAddBtn_OnClick* in the scripts is invoked, adding the attendee into the attendee list box and the free time viewer. Similar routines can be embedded in the HTML documents to make the controls on the page interactive.

Finally, the scripts are responsible for collecting data in the ActiveX controls, formatting them and placing them in the form, so the data in the controls can be submitted to the OrgWeb server through the form submission. For example, there is an input control *mtgdate* in the "Edit Meeting" form, which takes the date set in the date tracker as the meeting date. The scripts to perform the fetching of data from date tracker and submitting them in a form can be found in the subroutine *SubmitBtn_OnClick*:

```
Sub SubmitBtn_OnClick()
      Dim DateStr
      Dim DelimPos1
      Dim DelimPos2
      Dim Yr
      Dim Mn
      Dim Dy
      . . . .
      ' Getting the date text from the date tracker
      DateStr = form.datetrack.GetText
      ' Extracting the Year, month and day of the date string
      DelimPos1 = InStr( DateStr, "/" )
      Mn = Left( DateStr, DelimPos1 - 1 )
      DelimPos2 = InStr( DelimPos1 + 1, DateStr, "/" )
      Dy = Mid( DateStr, DelimPos1 + 1, DelimPos2 - DelimPos1 - 1 )
      Yr = "19" & Right( DateStr, Len( DateStr ) - DelimPos2 )
      ' Setting the date value in the form
      form.mtgdate.Value = Yr & Mn & Dy
End Sub
```

SubmitBtn_OnClick is initiated by clicking the "Submit" button in the form. It takes the date string from the date tracker, formats it into a date parameter in the OrgWeb protocol, and puts it in the form input *mtgdate*. After the subroutine is executed, the "Edit Meeting" form will be submitted to the OrgWeb server. The attendee information can also be submitted in the form in a similar manner.

Chapter 8 Conclusion and Future Work

8.1 Summary and Conclusion

Before the conclusion of this thesis report, here is a summary of the ideas that have been explored and discussed.

In order to make the internet C&S protocol work readily and easily in the existing computing environment, the protocol is built on top of HTTP, which is the native protocol used widely in the internet community nowadays. Protocol requests are made in URL and in HTML forms on Web pages. HTML Web pages are used as the protocol responses. All this can make the protocol and the C&S system immediately workable in standard Web browsers and major Web servers. Moreover, the protocol can inherit the lightweight and fast nature from HTTP.

The introduction of response templates allow the packing of several C&S commands into a single protocol request command. This enables a protocol user to perform multiple but related tasks by executing a single request command, with a single HTTP request and response. This makes the protocol efficient and request command set simple. The use of response templates also facilitates flexibility in the user interface design in the C&S system.

To solve the problems caused by the "stateless" nature in HTTP, the session management process is designed and established to keep state in the C&S service. With a session manager process on the server side, session information can be kept across different HTTP requests within a C&S session. This promotes a more secure and efficient C&S protocol.

To eliminate a number of inefficiencies in internet scheduling services due to the user interface limitations in HTML, ActiveX objects are designed, implemented and inserted in HTML pages of the C&S system. The ActiveX objects, along with the inline scripts embedded in HTML pages that manipulate these objects, keep the protocol simple and efficient. Moreover, rich UI elements in C&S applications which are formerly difficult to implement in simple HTML can now be used on Web pages in the C&S system.

In this thesis project, the C&S protocol and a prototype of the OrgWeb system have been implemented, and the ideas discussed in this report have been tested with this prototype. In the system, a user can view calendar information, create, edit and delete personal appointments, create group meetings and respond to scheduling notices. The protocol and the system are evaluated in a number of areas:

- Efficiency and simplicity of protocol usage: The protocol is kept fairly efficient in the OrgWeb system. Viewing a day, a week or a month's worth of appointments can be achieved in one request. Creating, editing or deleting a particular appointment can be done in a couple of requests (one to get the HTML input form, and the other to make the actual request). Viewing a meeting notice in the mailbox and processing a meeting notice also takes one request each. Also the protocol definition is simple. For all the C&S features mentioned above, there are less than 20 published request commands in the command set.

- Keeping state in HTTP: The prototype proves that the session manager process is able to keep state in an HTTP-based application. Session management, including request authentication, has been made possible in the system.
- Interactivity in user interface: By inserting the C&S ActiveX controls and inline VBScript in HTML pages of the OrgWeb responses, the HTML pages are made much more interactive. Some of the interactive user actions which cannot be seen in static HTML pages have been described in Chapter 7. Very complicated UI elements from an existing C&S system (e.g. the free time viewer and the date tracker in Lotus Organizer) can be displayed in the HTML pages, giving the users a more interesting and intuitive user interface. Furthermore, with the ActiveX controls the scheduling services, such as the free time search service, can be implemented and used in the protocol in a straightforward and efficient way.

In conclusion, with the introduction of the session management process, and with the help from C&S ActiveX controls and HTML inline scripts, an efficient internet C&S protocol can be built on top of HTTP.

8.2 Future Work

In the thesis prototype, the HTTP-based C&S protocol can only support simple viewing and editing of calendar appointments, and simple group scheduling features. There are many other important features found in existing C&S applications that are worth adding to extend the protocol. Sections other than the calendar, such as to-do list, address book, call tracker and event planner which can be found in typical C&S

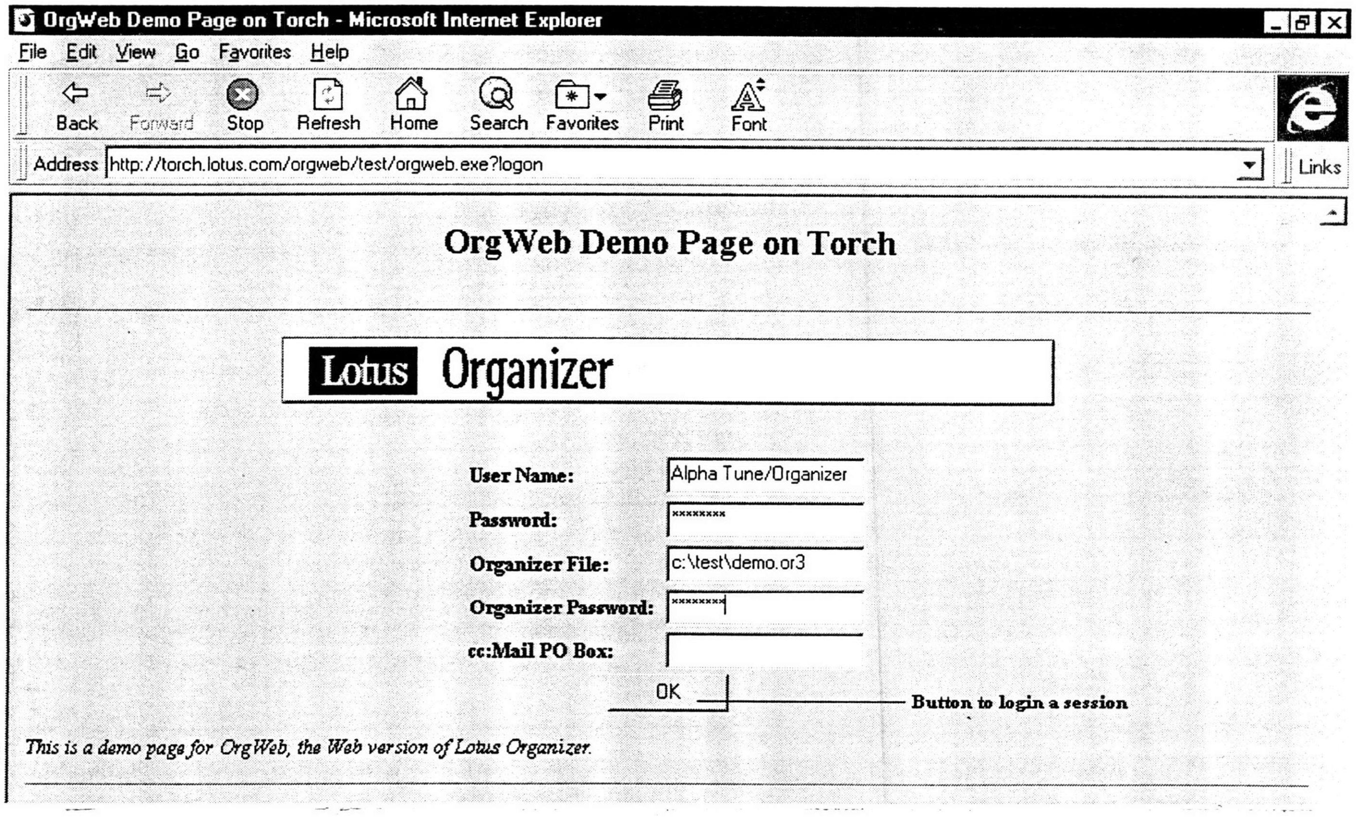
applications can be integrated into the OrgWeb system to provide a more complete set of C&S functionality. To accommodate more and more different sections and types of information in the protocol, the protocol can make use of the PATH_INFO CGI environment variable when requests are made, in order to provide more hierarchical and organized data retrievals and keep the protocol simple. Other features like audio reminders (e.g. alarms for appointment) and "show through" of information across different sections (e.g. showing items on to-do list on the calendar views) can be interesting extensions added to the protocol, and may require some thoughts and design before they can be supported by the protocol.

Currently in the OrgWeb system, the CGI process is responsible for parsing the response templates to create the HTML document for responses. If session or database information is required in making a response, the CGI process will talk to the session manager process through named pipes to access the data (See Chapter 6). However, when more than one piece of information is required to get from the session manager to make a response, the CGI process needs to communicate with the server process more than once. To make the system more efficient, the job of parsing response templates and forming the HTML responses can be moved to the session manager. The CGI process is now only responsible for passing the request parameters from the Web server to the session manager, and also passing the HTML responses from the session manager to the Web server. In this way, only one connection between the CGI process and the session manager is required in every protocol request, no matter how many pieces of session or database data are desired. As a result, we can make use of the session table in the session manager to further simplify the protocol commands. Information such as calendar view and current date can be kept in the session table. There is no need to specify them in the request commands, and the session manager can fetch them from the session table directly in forming the responses.

As pointed out in Chapter 7, the C&S ActiveX controls used to enhance the UI of the OrgWeb system can only be supported on Windows-based platforms. To make the system a truly platform-independent client/ server application, a platform-independent object insertable in HTML is needed. Java applets offers a solution to this problem, but the difficulties involved in leveraging existing code means a very long development time and extensive code rewrite. Further research on new object types to solve the problem is certainly desirable in the development of the internet C&S system, and also other internet applications.

Appendix A
User Interface in the OrgWeb System

Figures showing the Web pages seen in an OrgWeb session are displayed in this appendix.


OrgWeb Demo Page on Torch - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font
Address
Links
Lotus Organizer
OK
Button to login a session

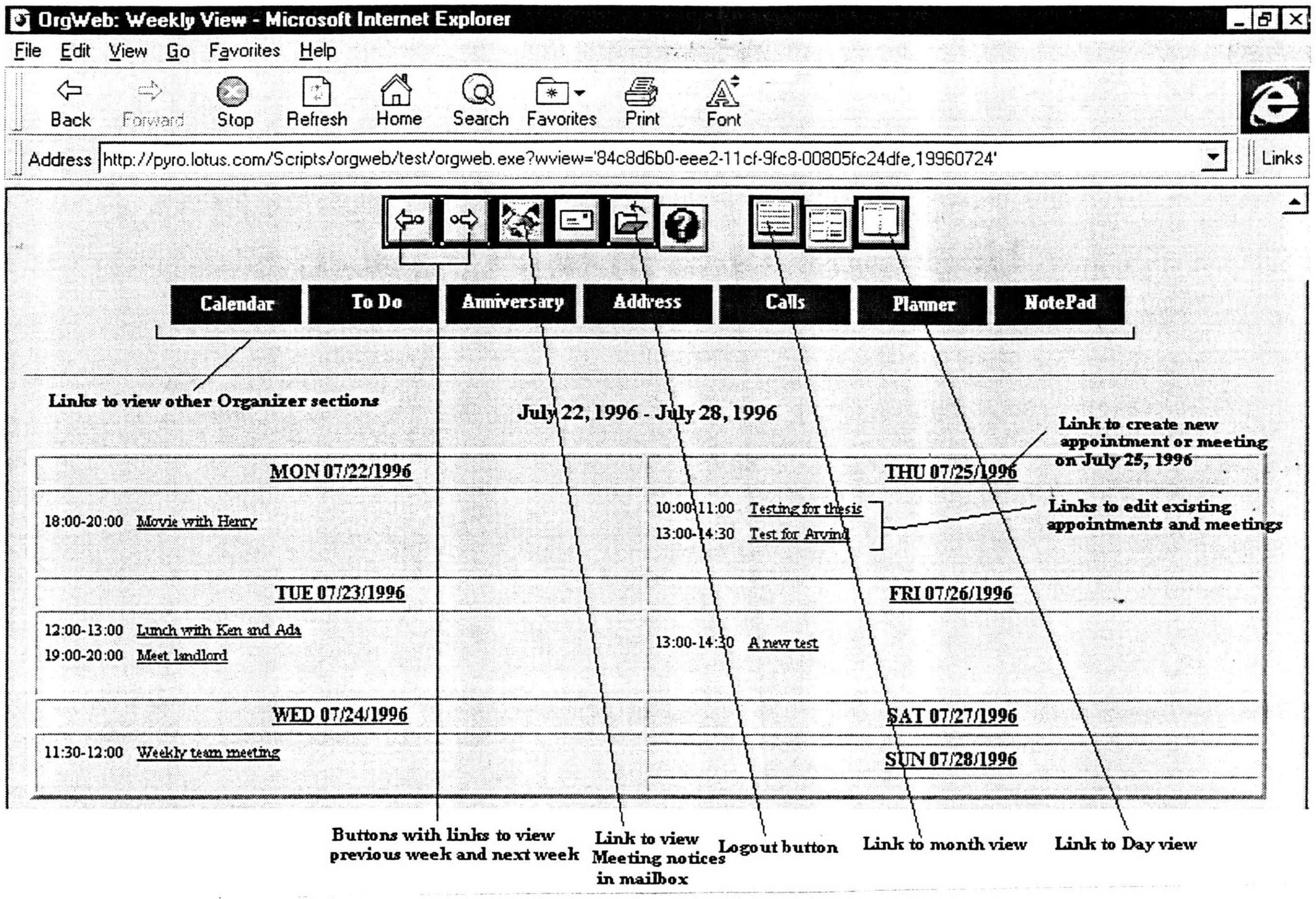
Address: http://torch.lotus.com/orgweb/test/orgweb.exe?logon

OrgWeb Demo Page on Torch

User Name:	Alpha Tune/Organizer
Password:	xxxxxxxx
Organizer File:	c:\test\demo.or3
Organizer Password:	xxxxxxxx
cc:Mail PO Box:	

This is a demo page for OrgWeb, the Web version of Lotus Organizer.

A.1: Logon page with the login form


OrgWeb: Weekly View - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font
Address http://pyro.lotus.com/Scripts/orgweb/test/orgweb.exe?wview='84c8d6b0-eee2-11cf-9fc8-00805fc24dfe,19960724'
Links
Calendar
To Do
Anniversary
Address
Calls
Planner
NotePad
Links to view other Organizer sections
Link to create new appointment or meeting on July 25, 1996
Links to edit existing appointments and meetings
Buttons with links to view previous week and next week
Link to view Meeting notices in mailbox
Logout button
Link to month view
Link to Day view

July 22, 1996 - July 28, 1996

MON 07/22/1996
18:00-20:00 Movie with Henry

TUE 07/23/1996
12:00-13:00 Lunch with Ken and Ada
19:00-20:00 Meet landlord

WED 07/24/1996
11:30-12:00 Weekly team meeting

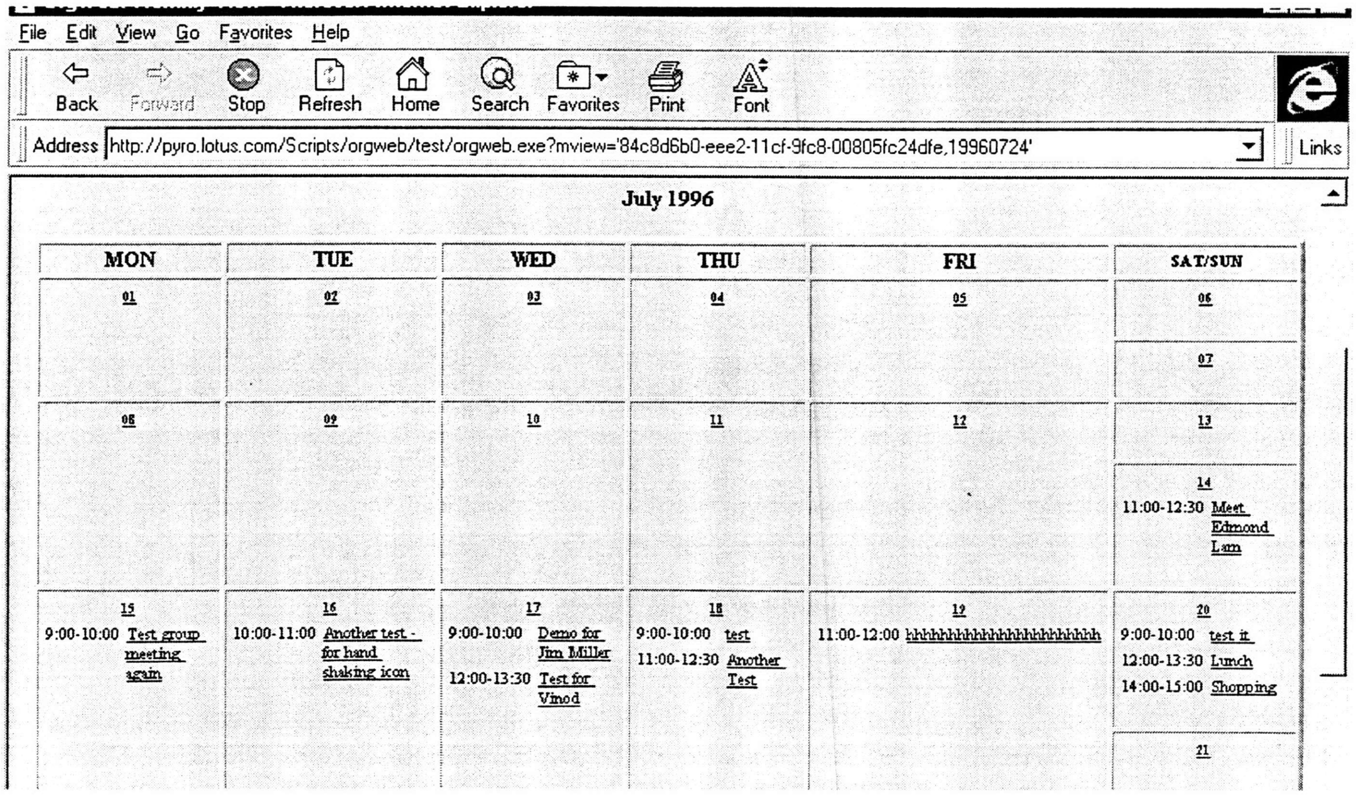
THU 07/25/1996
10:00-11:00 Testing for thesis
13:00-14:30 Test for Arvind

FRI 07/26/1996
13:00-14:30 A new test

SAT 07/27/1996

SUN 07/28/1996

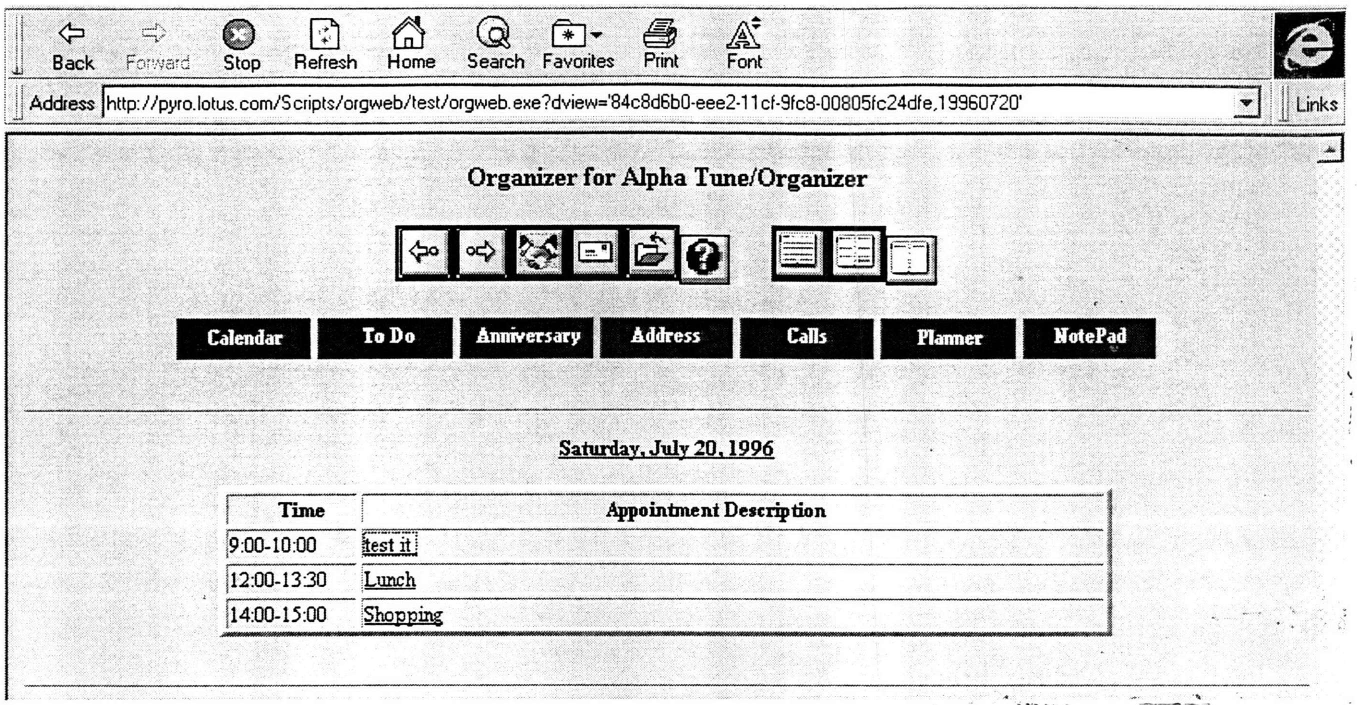
A.2: Calendar week view page


File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font
Address http://pyro.lotus.com/Scripts/orgweb/test/orgweb.exe?mview='84c8d6b0-eee2-11cf-9fc8-00805fc24dfe,19960724'
Links

July 1996

MON	TUE	WED	THU	FRI	SAT/SUN
01	02	03	04	05	06
					07
08	09	10	11	12	13
					14 11:00-12:30 Meet Edmond Lam
15 9:00-10:00 Test group meeting again	16 10:00-11:00 Another test - for hand shaking icon	17 9:00-10:00 Demo for Jim Miller 12:00-13:30 Test for Vinod	18 9:00-10:00 test 11:00-12:30 Another Test	19 11:00-12:00 hhhhhhhhhhhhhhhhhhhhhhh	20 9:00-10:00 test it 12:00-13:30 Lunch 14:00-15:00 Shopping
					21

A.3: Calendar month view page



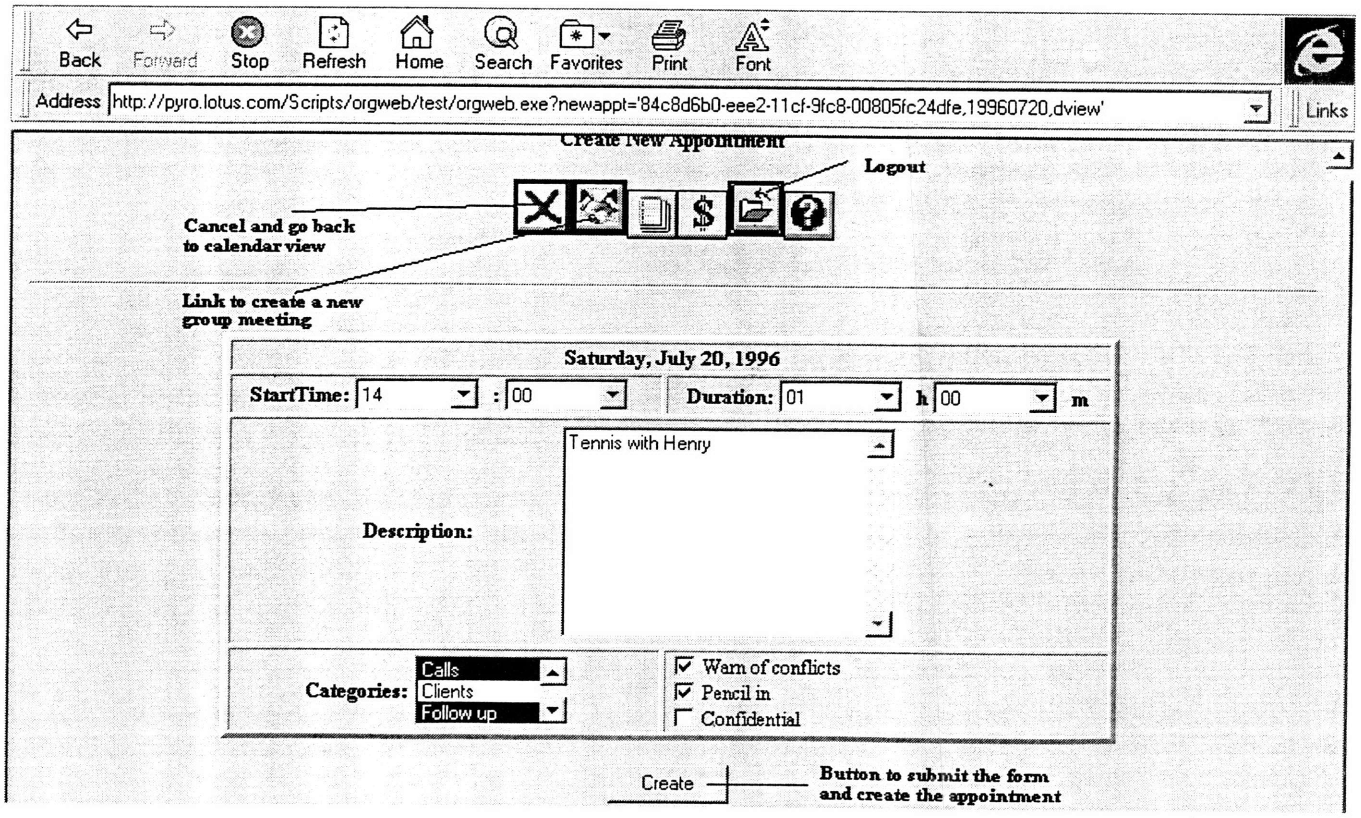
http://pyro.lotus.com/Scripts/orgweb/test/orgweb.exe?dview='84c8d6b0-eee2-11cf-9fc8-00805fc24dfe,19960720'

Organizer for Alpha Tune/Organizer

Saturday, July 20, 1996

Time	Appointment Description
9:00-10:00	test it
12:00-13:30	Lunch
14:00-15:00	Shopping

A.4: Calendar day view page


Back
Forward
Stop
Refresh
Home
Search
Favorites
Print
Font
Address
Links
Logout
Cancel and go back to calendar view
Link to create a new group meeting
Button to submit the form and create the appointment

http://pyro.lotus.com/Scripts/orgweb/test/orgweb.exe?newappt='84c8d6b0-eee2-11cf-9fc8-00805fc24dfe,19960720,dview'

Create New Appointment

Saturday, July 20, 1996

StartTime: 14 : 00 Duration: 01 h 00 m

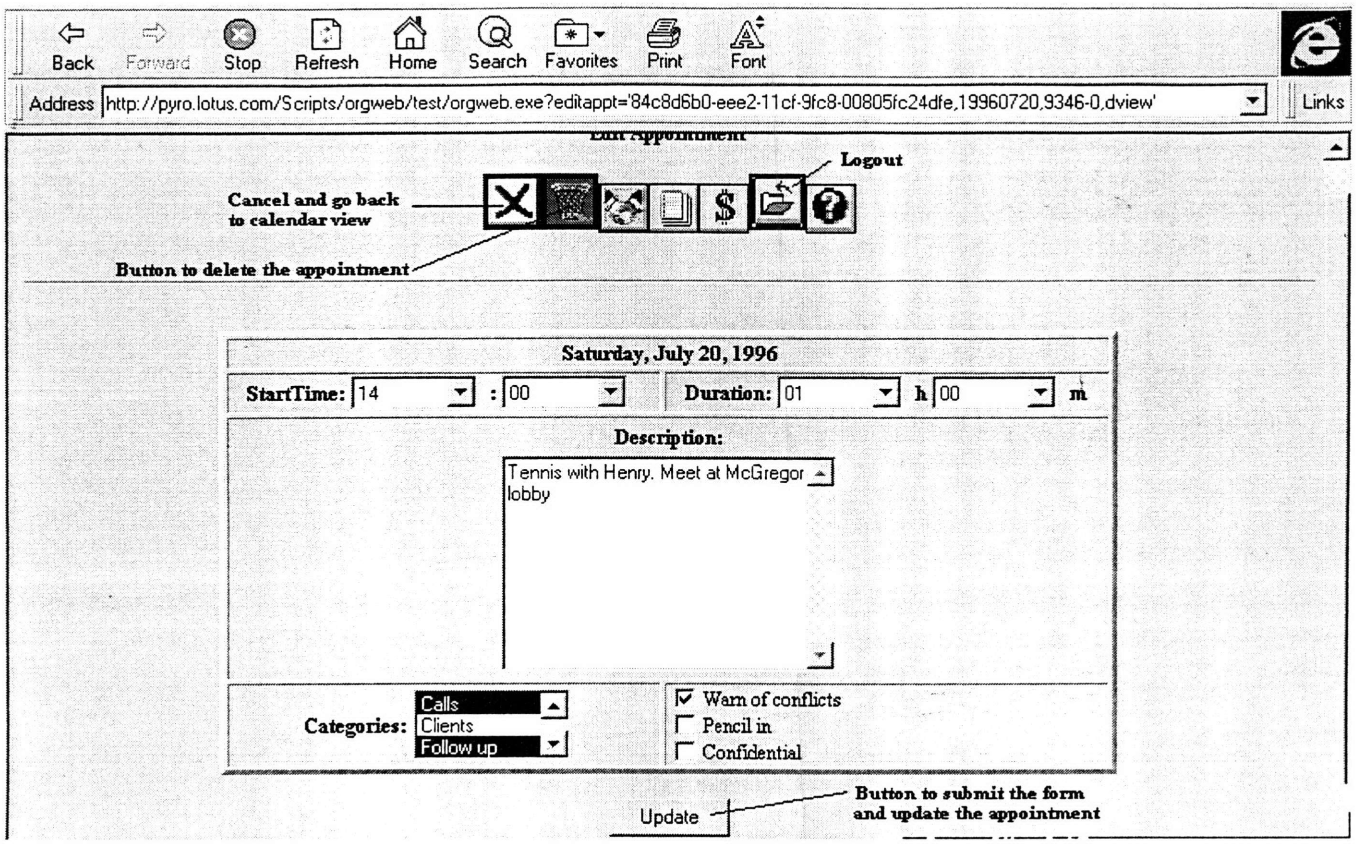
Description: Tennis with Henry

Categories: Calls, Clients, Follow up

☑ Warn of conflicts
☑ Pencil in
☐ Confidential

Create

A.5: Create new appointment page


Back
Forward
Stop
Refresh
Home
Search
Favorites
Print
Font
Links
Logout
Cancel and go back to calendar view
Button to delete the appointment
Button to submit the form and update the appointment

Address http://pyro.lotus.com/Scripts/orgweb/test/orgweb.exe?editappt='84c8d6b0-eee2-11cf-9fc8-00805fc24dfe,19960720,9346-0,dview'

Edit Appointment

Saturday, July 20, 1996

StartTime: 14 : 00 Duration: 01 h 00 m

Description:

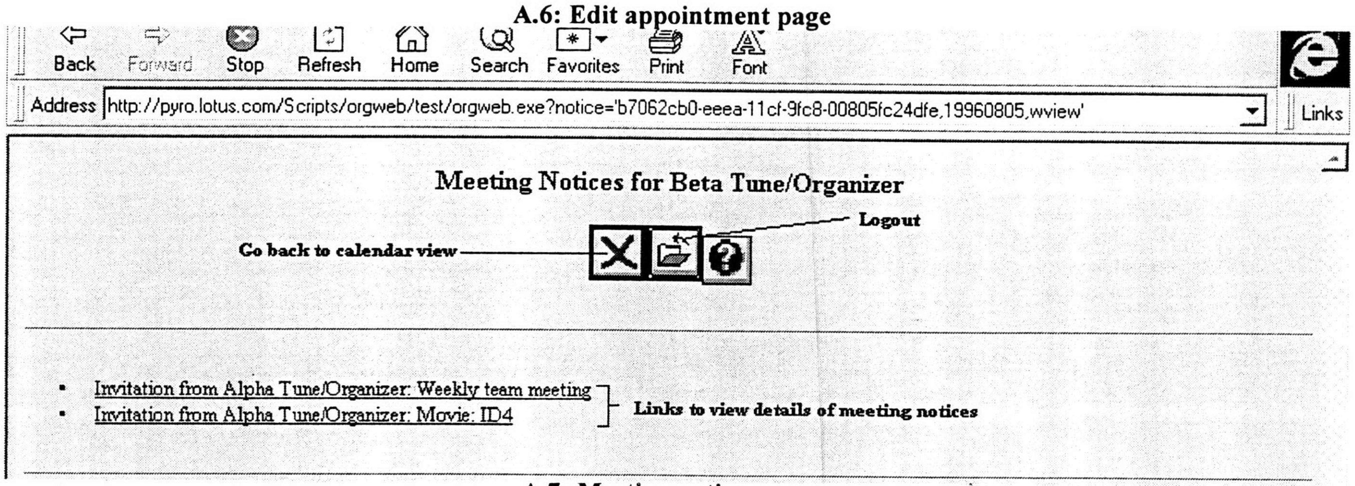
Tennis with Henry. Meet at McGregor lobby

Categories: Calls, Clients, Follow up

[✓] Warn of conflicts
[ ] Pencil in
[ ] Confidential

Update

A.6: Edit appointment page

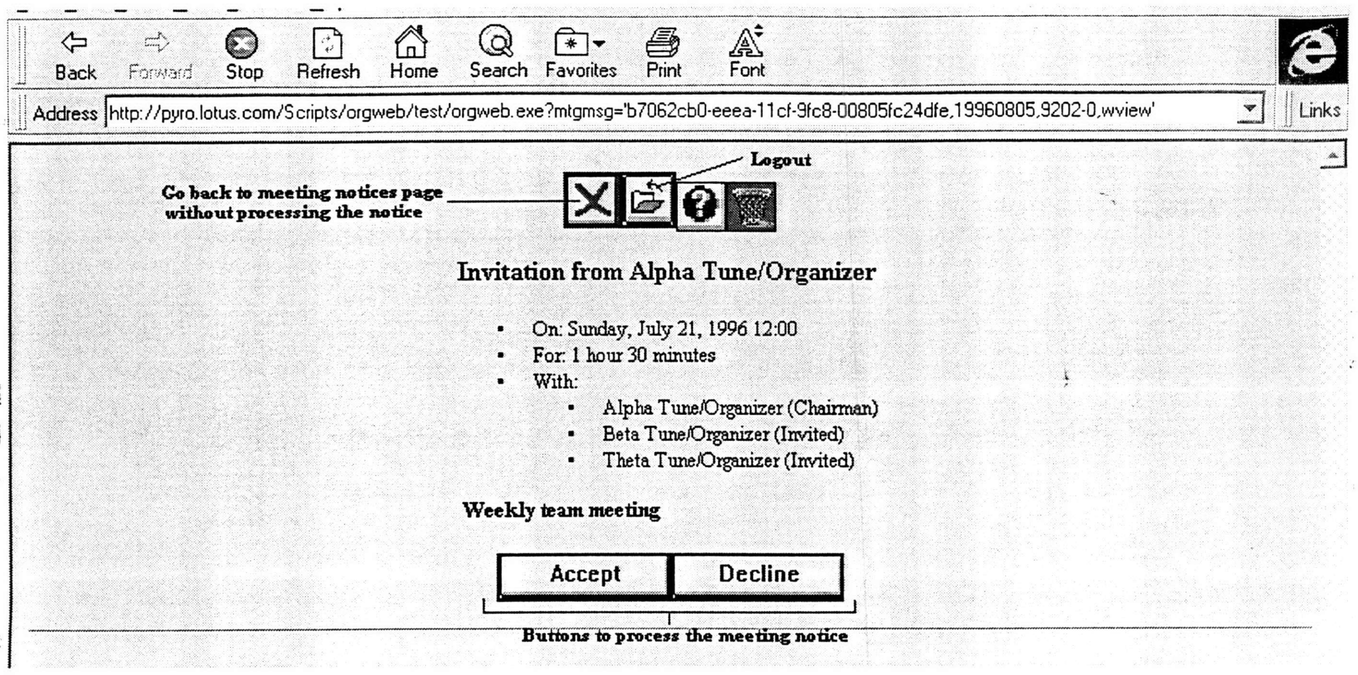


Address http://pyro.lotus.com/Scripts/orgweb/test/orgweb.exe?notice='b7062cb0-eeea-11cf-9fc8-00805fc24dfe,19960805,wview'

Meeting Notices for Beta Tune/Organizer

- Invitation from Alpha Tune/Organizer: Weekly team meeting
- Invitation from Alpha Tune/Organizer: Movie: ID4

A.7: Meeting notices page


Back
Forward
Stop
Refresh
Home
Search
Favorites
Print
Font
Address
Links
Logout
Go back to meeting notices page without processing the notice
Buttons to process the meeting notice

http://pyro.lotus.com/Scripts/orgweb/test/orgweb.exe?mtgmsg='b7062cb0-eeea-11cf-9fc8-00805fc24dfe,19960805,9202-0,wview'

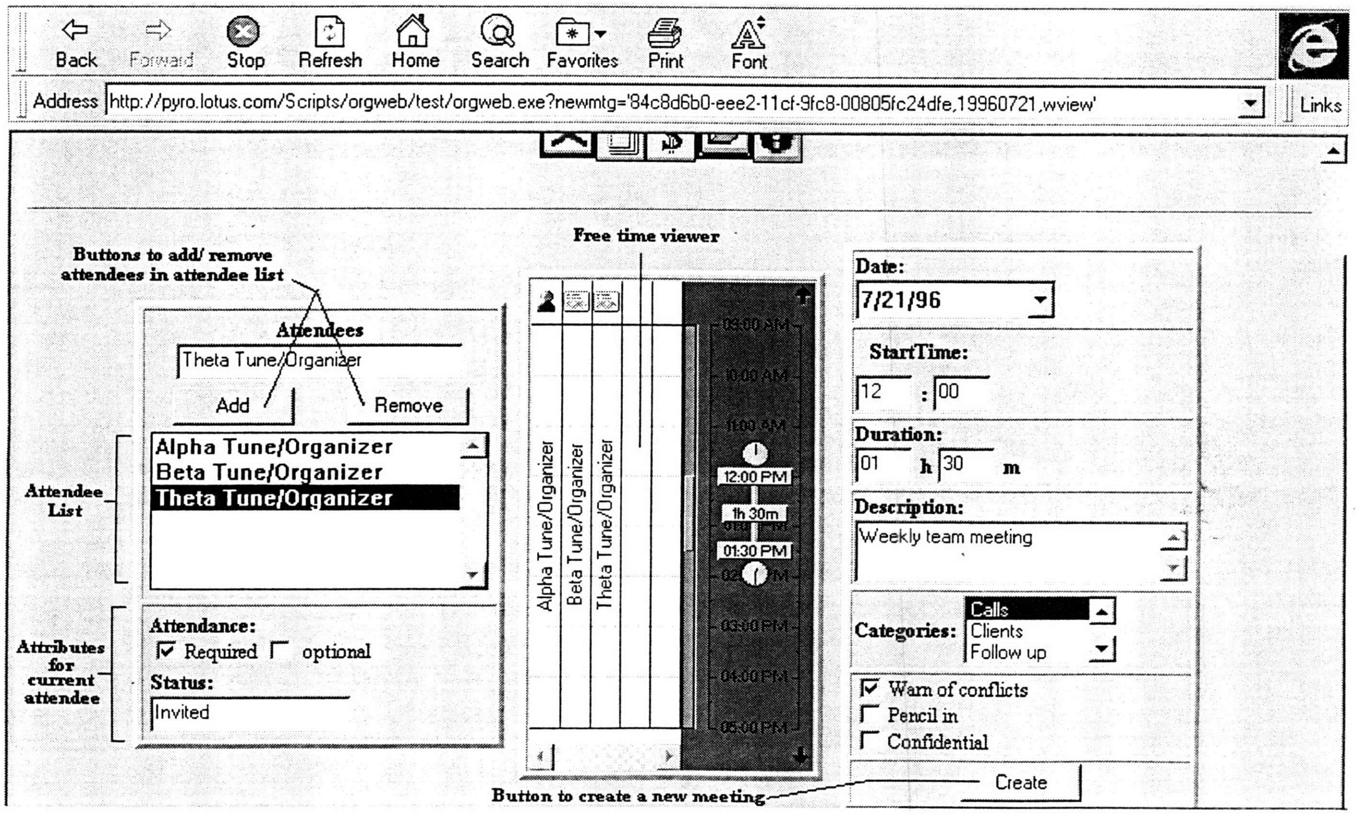
Invitation from Alpha Tune/Organizer

- On: Sunday, July 21, 1996 12:00
- For: 1 hour 30 minutes
- With:
  - Alpha Tune/Organizer (Chairman)
  - Beta Tune/Organizer (Invited)
  - Theta Tune/Organizer (Invited)

Weekly team meeting

Accept | Decline

A.8: Meeting notice details page



http://pyro.lotus.com/Scripts/orgweb/test/orgweb.exe?newmtg='84c8d6b0-eee2-11cf-9fc8-00805fc24dfe,19960721,wview'

Attendees: Theta Tune/Organizer

Alpha Tune/Organizer
Beta Tune/Organizer
Theta Tune/Organizer

Attendance: [✓ Required] optional
Status: Invited

Date: 7/21/96
StartTime: 12 : 00
Duration: 01 h 30 m
Description: Weekly team meeting
Categories: Calls, Clients, Follow up
[✓ Warn of conflicts]
Pencil in
Confidential

A.9: Creating new meeting page

Appendix B
Embedded Commands in Response Templates

A commands embedded within a response template appears between the OrgWeb-specific tags "<!--ORG-" and "-->". It is parsed by the server side CGI process to perform a particular task, and the command may be replaced by some HTML contents in the template afterwards. A list of these embedded commands is presented in this appendix.

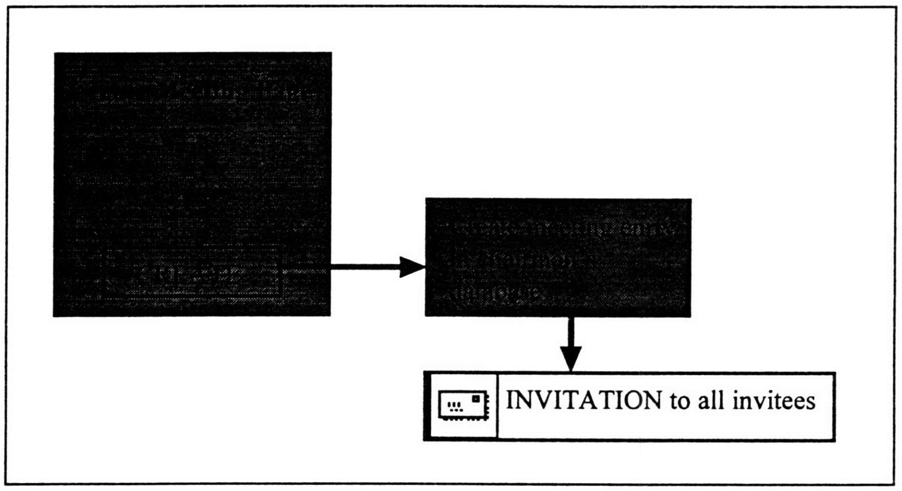
Command Name	**Meaning**
back	Output the calendar view the user should go back if the current operation is canceled.
categorynames	Output the category names the user can choose.
curruid	Output the unique ID of the current appointment/ meeting/ meeting notice
date	Output the current date in form of the date parameter in the protocol (See Section 5.1.3)
daystr	Output the current date in a user readable format, with week day information. e.g. "Saturday, July 20, 1996"
dfwd	Output the date one day after the current date
dprev	Output the date one day before the current date
dtrackdate	Output the current date in the format used in a date tracker: *MM/DD/YY* e.g. 7/20/1996
dvlistappt	List all the appointments on the current date in a table form
editapptform	Output an "Edit Appointment form" in the HTML document
editmtgform	Output an "Edit Meeting form" in the HTML document
icon	Output the path for the icons
listnotices	List the descriptions for all the meeting notices in the user's mailbox
msgdetails	List the details of the current meeting notice
mfwd	Output the date one month after the current date
mprev	Output the date one month before the current date
mvlistappt	List all the appointments in the current month in a table form
name	Output the name of the current user
pendingnotice	Check whether there are unprocessed meeting notices in the user's

	mailbox. If so, output the name of the icon which shows animated "hand-shaking" images. If not, output the name of the icon which shows static "hand-shaking" image.
script	Output the initialization scripts for a "Create Meeting Form" or an "Edit Meeting Form"
sessionid	Output the session ID for the current OrgWeb session
today	Output today's date from the operating system
wfwd	Output the date one week after the current date
wprev	Output the date one week before the current date
wvlistappt	List all the appointments in the current week in a table form

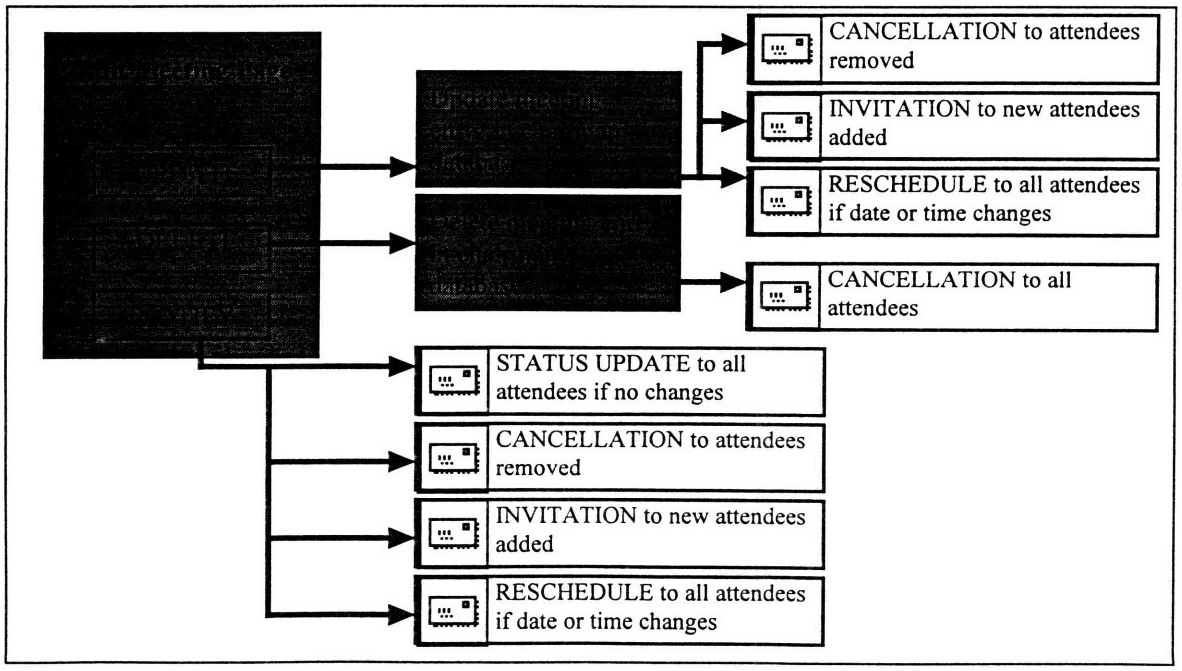
Appendix C
Specification of Group Scheduling in the OrgWeb System

C.1 Scheduling Actions on Meeting Entries

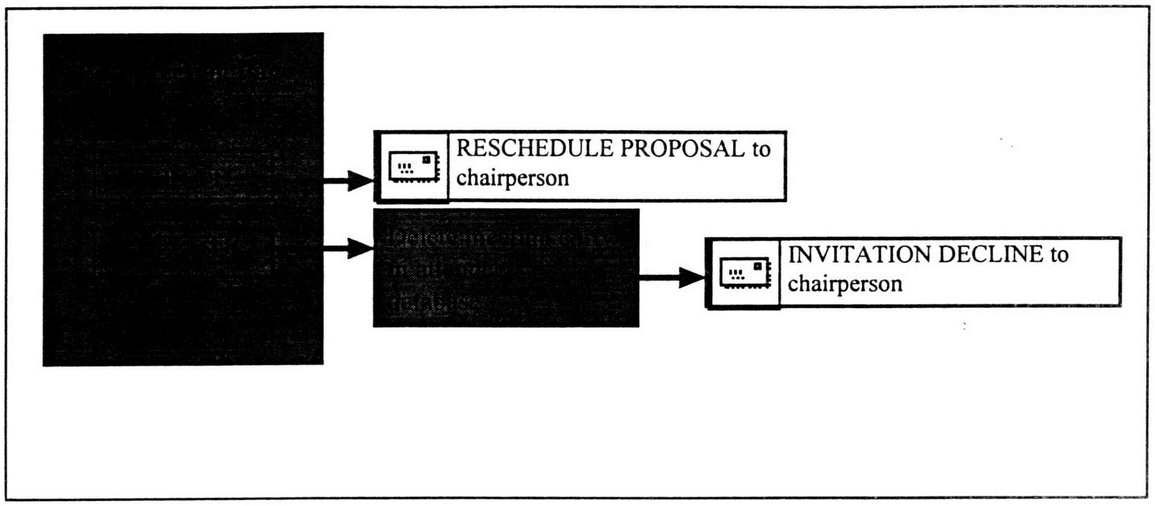
(a) Create New Meeting (by Chairperson)



(b) Edit Existing Meeting as Chairperson

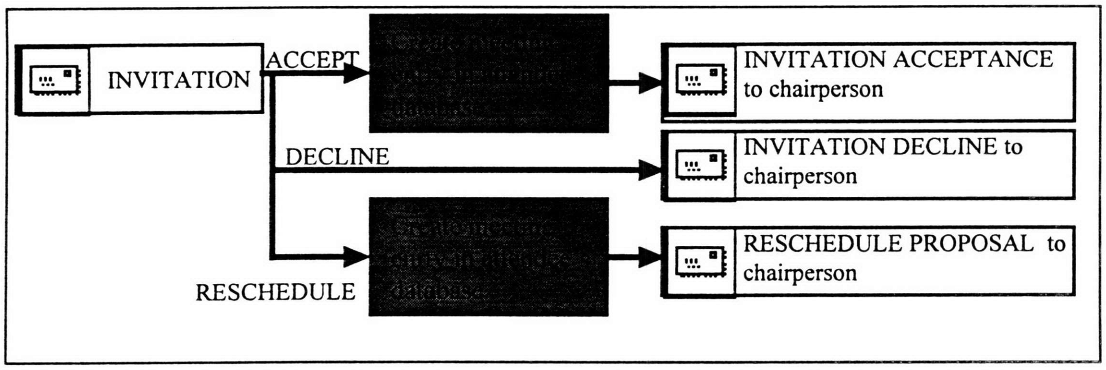

CANCELLATION to attendees removed
INVITATION to new attendees added
RESCHEDULE to all attendees if date or time changes
CANCELLATION to all attendees
STATUS UPDATE to all attendees if no changes
CANCELLATION to attendees removed
INVITATION to new attendees added
RESCHEDULE to all attendees if date or time changes

(c) Edit Existing Meeting as Attendee

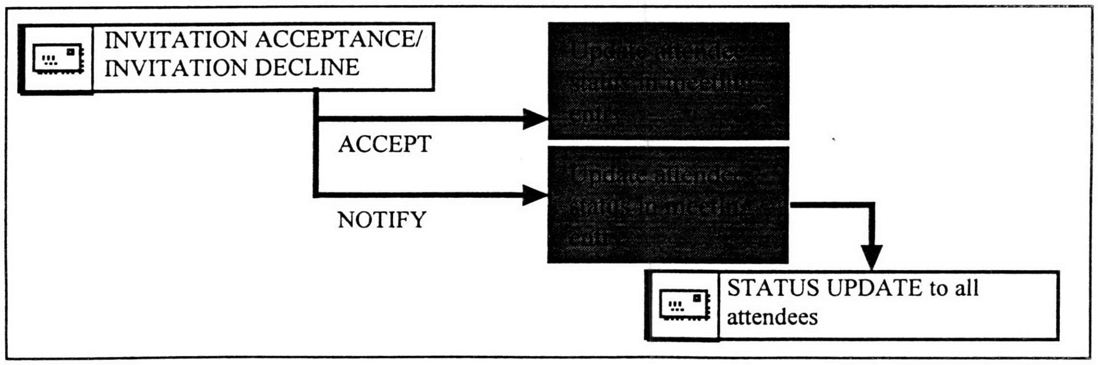

RESCHEDULE PROPOSAL to chairperson
INVITATION DECLINE to chairperson

C.2 Processing Meeting Notices

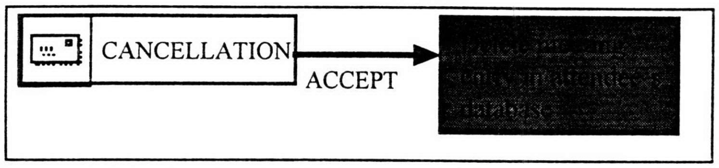
(a) Invitation (from chairperson)


INVITATION
ACCEPT
DECLINE
RESCHEDULE
INVITATION ACCEPTANCE to chairperson
INVITATION DECLINE to chairperson
RESCHEDULE PROPOSAL to chairperson

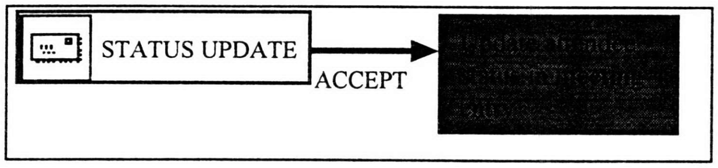
(b) Invitation Acceptance/ Invitation Decline (from attendee)


INVITATION ACCEPTANCE/ INVITATION DECLINE
ACCEPT
NOTIFY
STATUS UPDATE to all attendees

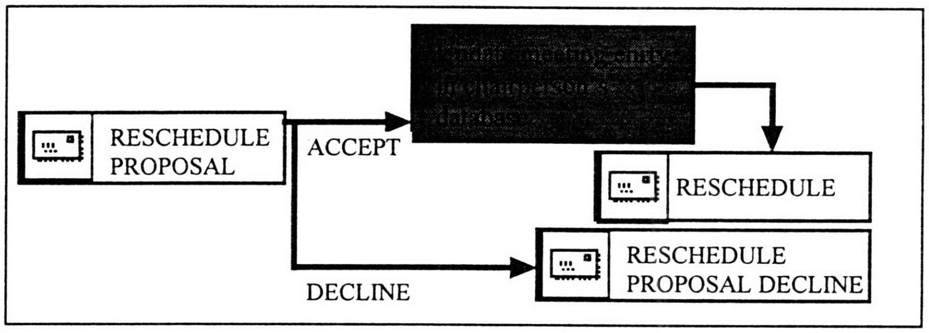
(c) Cancellation (from chairperson)



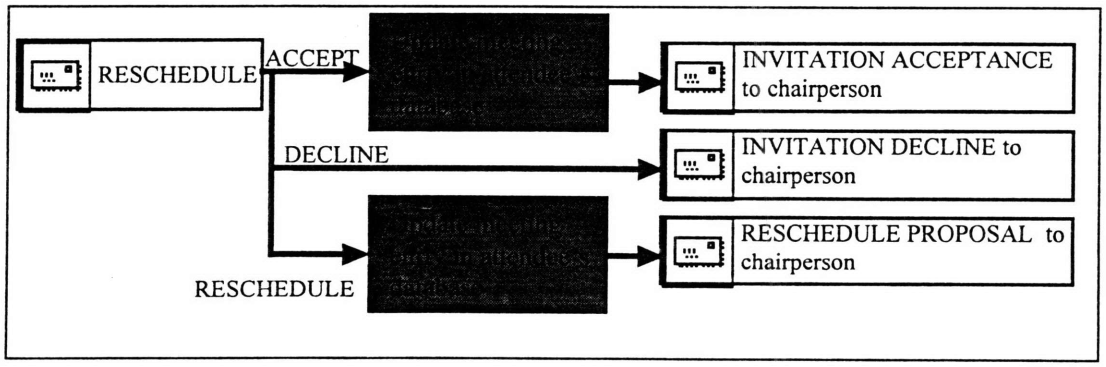
(d) Status Update (from chairperson)



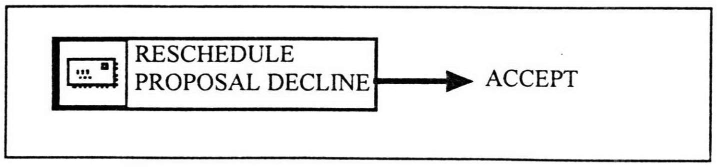
(e) Reschedule Proposal (from attendee)



(f) Reschedule (from chairperson)


RESCHEDULE
ACCEPT
INVITATION ACCEPTANCE to chairperson
DECLINE
INVITATION DECLINE to chairperson
RESCHEDULE
RESCHEDULE PROPOSAL to chairperson

(g) Reschedule Proposal Decline (from chairperson)



References

[1] "Internet Trends", General Magic, Inc. Available online at *http://www.genmagic.com/internet/trends/*

[2] David Belind, "Phillippe Kahn: The Comeback Kid", 1996, PCWeek, January 8, 1996. Available online at *http://www.pcweek.com/archive/960108/pcw00070.htm*

[3] Steve Jackson, "The Internet Control Pack", 1996, Microsoft Interactive Developer Magazine, Volume 1, No.1, Spring 1996.

[4] John December and Mark Ginsburg, "HTML & CGI Unleashed", 1995, Sams.net Publishing

[5] "The Common Gateway Interface" at "NCSA HTTPd Home Page". Available online at *http://hoohoo.ncsa.uiuc.edu/cgi/.*

[6] T. Berners-Lee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0", 1996, Internet Informational RFC 1945. Available online at *http://ds.internic.net/rfc/rfc1945.txt*

[7] John Ellsworth, "A Technical Look at Lotus Organizer 2.0", 1995, The View - Technical Journal for Lotus Notes Software, Volume 1, Issue 2, May/ June 1995.

[8] Kraig Brockschmidt, "Inside OLE", 1995, Microsoft Press.

[9] "Lotus Organizer API Reference", 1995, Lotus Development Corporation.

[10] Jeffrey Richter, "Advanced Windows, Developers' Guide to Windows NT 3.5 and Windows 95", 1995, Microsoft Press.

[11] Marshall Brain, "Win32 System Services, The Heart of Windows 95 and Windows NT", 1996, Prentice Hall, Inc.

[12] Brian Myers and Eric Hamer, "Mastering Windows NT Programming", 1993, Sybex, Inc.

[13] C. Kindel, L. Montulli, E. Sink, W. Gramlich, J. Hirschman, T. Berners-Lee, D. Connolly, "Inserting Objects into HTML", 1996, W3C working draft, WD-object-960422, Available online at *http://www.w3.org/WWW/TR/WD-object.html*

[14] "Preview the OnTime Web Edition v4.0", Cambell Services, Inc., 1996, Available online at *http://mars.ontime.com/*

[15] Paul DiLascia and Victor Stone, "Sweeper", 1996, Microsoft Interactive Developer Magazine, Volume 1, No.1, Spring 1996.

[16] Laura Lemay and Charles Perkins, "Teach Yourself Java in 21 Days", 1996, Sams.net Publishing.

[17] "Plug-in Guide", Netscape Communications Corporation, 1996, Avaliable online at *http://home.netscape.com/eng/mozilla/3.0/handbook/plugins/pguide.htm*

[18] "Microsoft Visual Basic Scripting Edition", Microsoft Corporation, 1996, Available online at *http://www.microsoft.com/vbscript/*

[19] "JavaScript Authoring Guide", Netscape Communications Corporation, 1996, Available online at *http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/index.htm*

[20] "Web Trends", Interse Corporation, 1996, Available online at *http://www.interse.com/webtrends/*

[21] "Browser Scorecard", Dataquest Interactive, 1996, Available online at *http://www.dataquest.com/insight/in-i001.html*

[22] "Java Applets", Netscape Communications Corporation, 1996, Available online at *http://home.netscape.com/comprod/products/navigator/version_2.0/java_applets/index.htm*

[23] "Internet Component Download", Microsoft Corporation, 1996, Available online at *http://www.microsoft.com/intdev/signcode/codedwld.htm*

[24] "Scripting Object Model", Microsoft Corporation, 1996, Available online at *http://198.105.232.30/intdev/sdk/docs/scriptom/index.htm*

[25] "Programming with MFC: Encyclopedia", 1995, Microsoft Visual C++ 4.1, Microsoft Corporation.

[26] Matthew Powell and Leon Braginski, "WinInet", 1996, Microsoft Interactive Developer Magazine, Volume 1, No.1, Spring 1996.

5466-62