Agora–An Experiment in Multimedia Message Systems

Najah Naffah and Ahmed Karmouch, Bull Transac

The Agora multimedia message system was designed as part of the Kayak project, employing a multisite architecture for the distributed, integrated electronic office.

We consider a distributed message system that allows exchange of compound documents among workstations to be the foundation of integrated office systems in the future. Although not widely used in corporations today, experiments like the one we describe here have shown the feasibility of these concepts. In particular, they have highlighted some essentials for a technically successful electronic office system:

- multimedia workstations with good interface features,
- reliable message transfer,
- a flexible naming scheme, and
- a multisite, distributed architecture.

We here describe the Agora project, conducted within the context of a larger project called Kayak,[1] and its architecture. Agora's functional model was defined in parallel with the IFIP model.[2] Kayak's goal was to build a complete, integrated electronic office.

The Kayak project

When the decision was made in France in 1979 to launch a pilot project of office systems, word processors had just begun to penetrate the office automation market. On the other hand, there were few electronic-mail systems except in universities, research laboratories, and a few private companies. Arpanet mail,[3] built into the host systems connected to Arpanet, serves as a good example of a research mail system. Local networks were in the embryonic stage. Only Xerox PARC had a prototype of bitmapped workstations connected through the Ethernet local network to mail, file, and printing servers.[4] In the pilot project, called Kayak,[1] we wanted to explore in depth the concept of an integrated office system.

We investigated office integration at different levels: *workstation, network,* and *application.* The workstation goal was to build a personal machine supporting the functions necessary to acquire and present the kinds of media used by office workers in their daily tasks—voice, plus written documents containing manuscript, typed text, drawings, pictures, and graphics. A friendly interface criteria led to a voice command and signalling system, a large bitmap screen, a pointing device, and a standard keyboard.

The workstation, called Buroviseur,[5] was assembled from the components shown in Figure 1. It has a multiboard architecture based on the Intel Multibus. All boards have the standard Intel SBC for-

0018-9162/86/0500-0056$01.00 © 1986 IEEE

mat. Some boards, like the CPU (8086), memory (512K bytes), and disk controller, have been adopted off-the-shelf. Others, like the bitmap-mouse-keyboard driver, a hundred-word voice recognition system, text-to-speech synthesis, and voice codec boards, were designed internally.

The software running in the workstations of Kayak contains the necessary drivers for all these devices, plus a set of application programs for document management. Two important packages in the message application are the multiwindow package, called Vitrail,[6] and the compound-document editor/formatter, called Plume.[7]

Vitrail is a standard windowing package with a high-level interface to the application program developers, who interact with devices by manipulating primitives such as open window, display object, create menu, read input, and the like. The multiwindow package is built on top of a toolkit that deals with

(1) display of all objects, including multifont characters, graphical objects, and facsimile boxes;

(2) object manipulation, such as copy, move, reduce, and zoom; and

(3) the conversion of data, such as text-to-speech and word recognition.

The editor, Plume, is a WYSIWYG (what you see is what you get) editor that deals with structured documents containing entities of different levels, including chapters, sections, paragraphs, characters of various fonts, and alphabets.

Vitrail has enormously facilitated the implementation of the user agent (exploitation features) in the Buroviseur. On the other hand, Plume has served as the basis of interaction in Agora, the multimedia message system.

A local network based on the CSMA-CD access scheme interconnects workstations. A 1M-byte-per-second baseband local-area network, or LAN, called Danube, it can cover two kilometers and connect 255 nodes. Different Danube networks were connected with X.25 gateways.[8] This architecture is the infrastructure upon which the distributed message system was installed with distributed name servers and messaging servers. In a typical site, 1M-byte-per-second provides enough bandwidth to exchange multimedia messages among workstations. Real-time voice messages are exchanged through a telephone network driven by specific ports in the Buroviseur. No voice packets for real-time dialogue were adopted or mixed with non-real-time traffic. This decision was made because the potentially heavy load generated by real-time traffic would create instability in the CSMA-CD-based Danube network. However, pseudo-real-time information exchange among users takes place by means of workstations equipped with speech synthesizers. Text entered on one workstation can immediately be sent to another workstation and converted to voice.

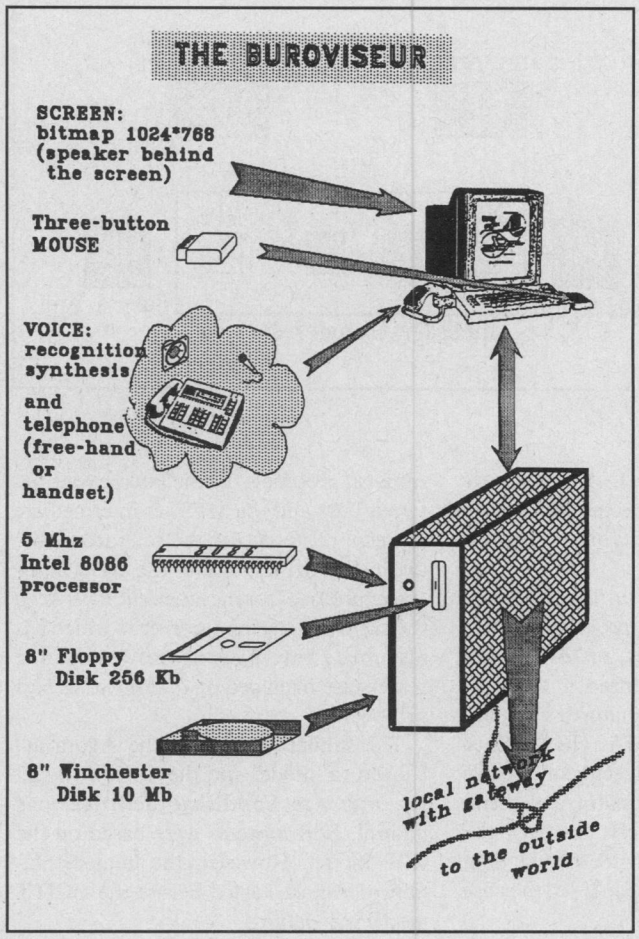

Figure 1. Buroviseur components.

Danube provides communication support corresponding to the first five layers of the ISO model for open systems interconnection. The NBS protocol[9] has been used above the session layer to exchange messages, while an SGML-like coding[10] has been adopted for the document structure.

Integration at the application level is as important for the user as the integration at the workstation level or the network level. It implies coexistence and information exchange among various applications. On a practical level, the user can switch between applications without stopping one to run another. He can also ask to take one document or piece of a document from one application and use it in another. In our implementation, this process was facilitated by the adoption of the same document representation standard and common session services in the workstations and servers.

The Agora architecture

The Agora system is designed according to a distributed and open architecture. From the conceptual point of view, Agora is based on a functional model similar to the IFIP model[2] and consists of different entities that work together to provide communication services.

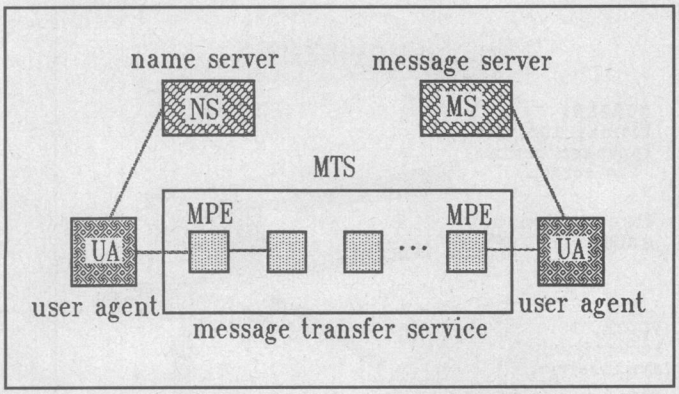

Figure 2. Agora architectural model.

Architectural model. A schematic description of the architecture appears in Figure 2. The components are as follows.

Message transfer system. The MTS consists of several message processors called message processing ends, or MPEs, that provide the functions needed to route messages from the originator user agent, or UA, to the recipient UA. The MPEs to which the originator and recipient UAs are attached are called originator and recipient MPEs, respectively. During the routing of a message, MPEs other than these two may be involved. If so, they are called intermediate MPEs.

User agent. The UA, a set of application processes, provides several functions: functions necessary to interact with the MTS and retrieve messages, and functions necessary to interact with the user to prepare and present outgoing and incoming messages.

Name server. The NS, a distributed database of names, manages the Agora users (humans, processes, and roles). The functions provided are mainly those necessary for manipulating names (creating, changing, and removing names) and names-to-address mapping. The NS can be reached from an entity called the *client,* a program that offers a set of primitives that allow its users to obtain all the functions provided by the name server. The client may be located in both UAs and the MTS.

Protocols. The interactions between the entities that appear in the architecture are managed and ruled by appropriate protocols. They are identified as *submission* and *retrieval* protocols for the interactions between UAs and the MTS, as intermediate or *relay* protocol for the interaction between two MPEs, and as the *client name server* protocol for the interaction between the client and the name server to which it is attached. Interactions between name servers are managed by *internname servers* protocol.

The similarity between the Agora architectural model and the CCITT model for message handling facilities[11] is natural. Both systems were based on the IFIP model. However, the implementation of Agora started before the CCITT model was defined.

Physical mapping. The Agora model is implemented with the following components:

- the access point, built in the Buroviseur, which offers the user agent facilities;
- the message server, which contains one MPE and a set of mailboxes; and
- the name server, which manages partitions of the subscribers' names.

The definition and role of these components can be explained by a description of operations executed from the time when a user has a message to communicate to the time when his message is perceived by another user. The two types of users are the originator, or the information creator, and the recipient, or the destination of the information.

The originator prepares his message with the Plume editor acting as the UA located in the Buroviseur. The UA produces a standard message for further transmission. The message is structured into various fields with equivalents to the envelope and content of standard mail. The former contains information necessary for routing the message. The latter contains information that the originator wants to communicate to the recipient. It contains a header and the body. The body may consist of different parts, each with various types of information (graphics, text, voice, and image).

The UA submits the message to the originator MPE. The MPE undertakes the analysis of all the envelope elements and may refuse the submitted message. During the envelope checking, the originator MPE may send queries to other components of the system (the name server) to obtain more information about the originator rights. Once an MPE has accepted the message for forwarding to the destination MPE, it starts a new phase of localization of the resources containing the mailbox.

To perform this function, the MPE asks the name server to provide the mapping between the recipient's name and the corresponding mailbox address. After the localization phase, the MPE establishes a route (according to its internal table) to the MPE that handles access to the mailboxes in the mail server.

In establishing the route, other MPEs may be involved. They act as intermediate MPEs. Private and public transmission media may be used to connect two MPEs on both ends. Once a route is established, the transfer of the message is performed and stored in the recipient mailbox located in the message server managed by the recipient MPE. The recipient can retrieve messages from his mailbox through his UA.

Name server

The name server represents the entity responsible for the management and administration of names. The management functions deal more specifically with names and addresses such as mapping, lookup, or list members of a distribution list, while the administration functions deal with attributes of mailboxes (such as right to write and delegate access) and the updating algorithm when a name or an address is to be created, changed, or removed.

Structure of names. Names are structured in a hierarchical manner. The

method used consists of breaking down the name domain into groups, each managing its own space of internal names.

A group is an attribute that can take different and unique values. Each value may represent an authority or an administration to which all local users are connected. An internal name is an attribute that may have different and unique values. Each of these values represents a user name belonging to a given group to which it is attached. Thus, the name of an Agora user takes the form

<user name> = <group name> <internal name>

The relation that exists between these two attributes is a hierarchical relation. The main advantages of this representation are reduction of ambiguity in names, and flexibility and adaptability for decentralized schemes and the naming authority. This naming scheme is satisfactory within an organization, but it should be extended to cover more attributes in an international environment.

Types of names. The name server in Agora defines and manages different types of names, including

- individual, an ordinary subscriber known to the system by his or her individual name (In addition to the mailbox address associated with the name, a password and a mailbox access key are also attached to the name for protection of the mailbox and control of information.);
- alias, considered another name of the individual to whom it is linked;
- distribution list, a list of predefined subscriber names (members of the list); and
- teleconference list, with a definition similar to that for the distribution list, but with a wider range of facilities. A teleconferencing list has an organizer, who is the creator of the list. A teleconference may be public or private. A public teleconference may be joined by any subscriber at any time. A private teleconference is restricted to subscribers authorized to join by the organizer.

Management of names. To keep information about group names continuously available, the database of group names is replicated in each name server. For the same reason, partitions of names may also be replicated. Hence, all the name servers have the same copy of names on different sites.

Because of the nature of message system applications, in which delays are counted in terms of minutes, hours, and even days, we can tolerate temporary inconsistencies that may occur during conflicting update operations of the name database without any damage. Accordingly, in Agora an *optimistic updating algorithm* is used based on a time-stamping mechanism summarized as follows:

The most critical update operations are create-name and delete-name queries. They can be initiated by any administrator from any site by interrogating the local name server. When a create-name query is received by a given name server from an administrator, a time stamp is attached (at the name server level) to the query, which is processed locally to see if the new name to create is already registered.

If this is the case, the create-name query is rejected. If not, the name is added with its time stamp to the name server and the update operation is broadcast to all other name servers.

Upon receipt of the update operation, each of the other name servers performs the query as follows: If the name to create does not exist in that name server, the name is added to it with its time stamp. Otherwise, the time stamps are compared. The name that has the most recent time stamp is ignored and the other name is kept in the name server with its time stamp.

For the delete-name query, the process differs. It is performed locally by the initiator name server, which compares the query time stamp and the time stamp of the already-registered name that must be deleted. The name is deleted if the query time stamp is more recent than the one associated with the registered name. Otherwise, the query is ignored. When the delete-name query is performed locally by the initiating name server, it is broadcast to all other name servers, which process the query in the same way as in the initiating name server.

The delete-name operation is performed in two steps:

(1) The name is marked "name in deletion progress" and must not be recreated until it is removed from the name server.

> We believe that a simple optimistic algorithm takes less time than other algorithms and provides a satisfactory solution here.

(2) The name is removed from the database of valid names.

This second step can be performed after time, even in the order of weeks, because we want to ensure that the name is deleted from all name servers before it is recreated.

Notice that each initiating name server performs an update operation before broadcasting it to all other name servers. Theoretically, the update operations should be received by the other name servers in the same time ordering as they were sent. However, this cannot be guaranteed because of the transmission delays and the computers' variable load. The result: inconsistency may occur for a period of time. Furthermore, the time-stamp mechanism used here does not guarantee that the administrator who initiated the update operation will win, because of the desynchronization of clocks between different name servers.

However, from our experience we believe that a simple optimistic algorithm, like the one we implemented, takes less time than other algorithms and provides a satisfactory solution in the context of a decentralized name server dedicated to a store-and-forward message communication system.

Message server

The message server is the tool that allows the user to receive messages from other subscribers and to retrieve messages from his mailbox. We might say that a message server is a collection of mailboxes.

We must emphasize that grouping of mailboxes in a message server does not

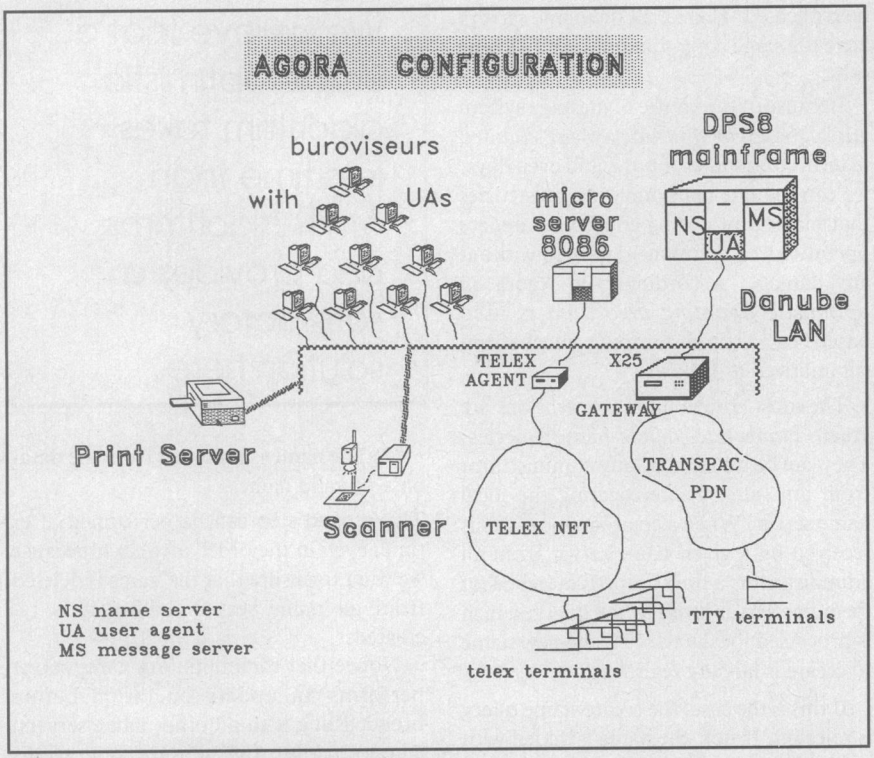

Figure 3. Agora implementation configuration.

reflect the naming strategy adopted in a name server. Within a message server, subscribers can belong to different groups. Likewise, all subscribers belonging to a given group can have their mailboxes in different message servers. The distribution depends on economic considerations, as well as professional criteria allowing a subscriber to have his mailbox (a primary mailbox if there is more than one) as close as possible to his current access point.

Server implementation. The first implementation of the message servers and name servers in Agora occurred in the mainframe Honeywell-Bull DPS8, located at INRIA- Rocquencourt (see Figure 3). The operating system was Multics. Three modules were installed: the name server, the message server, and a UA activated when access takes place with TTY terminals through Transpac (the national packet switched network in France). Danube was connected to Transpac through a special X.25 gateway. All Buroviseurs connected to Danube have their own UAs that can talk directly to the name and message servers in the mainframe without going through the TTY UA.

To manage storage and retrieval of messages, we used the Multics relational data store (MRDS), a relational database management system (DBMS). However, a DBMS does not suit large volumes of data, such as those contained in the bodies of messages. Hence, we restricted the use of MRDS to the fields of the header. The message body was managed by FMS, the Multics file management system.

MRDS itself remained hidden from users and served only as an implementation tool. User commands were mapped into MRDS queries by a special program.

Use of MRDS as an implementation tool benefited programming productivity. However, it is not mandatory to use a relational DBMS in a CBMS to enhance the quality of service. In fact, it increases the volume of code resident at run time and necessitates a large memory. For this reason, we decided to build another version of the name and message servers on top of a simple access method that uses FMS.

The implementation described was ported to an 8086-based microserver with 70M bytes of disk space. The volume of code in the mainframe measures approximately 22,000 lines of PL1.

The user agent

As in other mail systems, we faced the problem of terminals with different degrees of intelligence—teletype-like terminals, sophisticated workstations such as the Buroviseur, and telex terminals. The challenge: to provide a consistent interface for the various users independently of the type of terminal.

The user agent architecture consists of three modules: the dialog manager, the message editor, and the protocol handler.

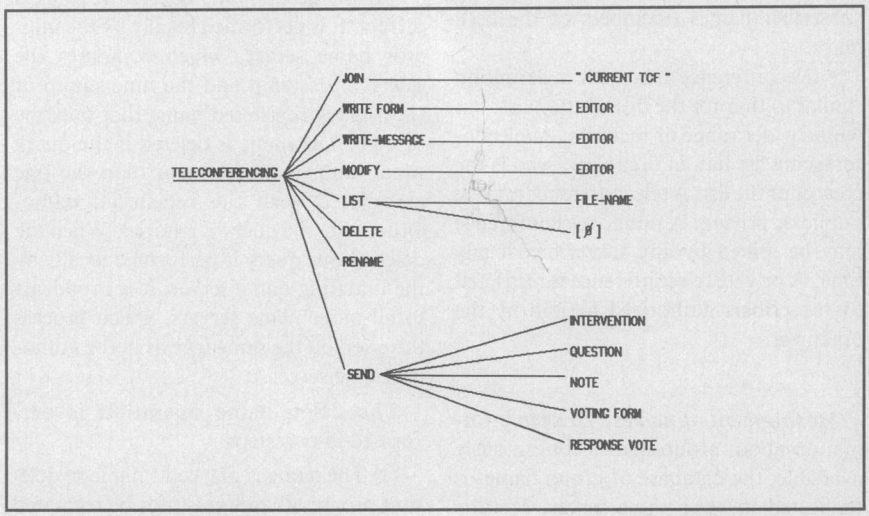

Figure 4. The teleconference tree.

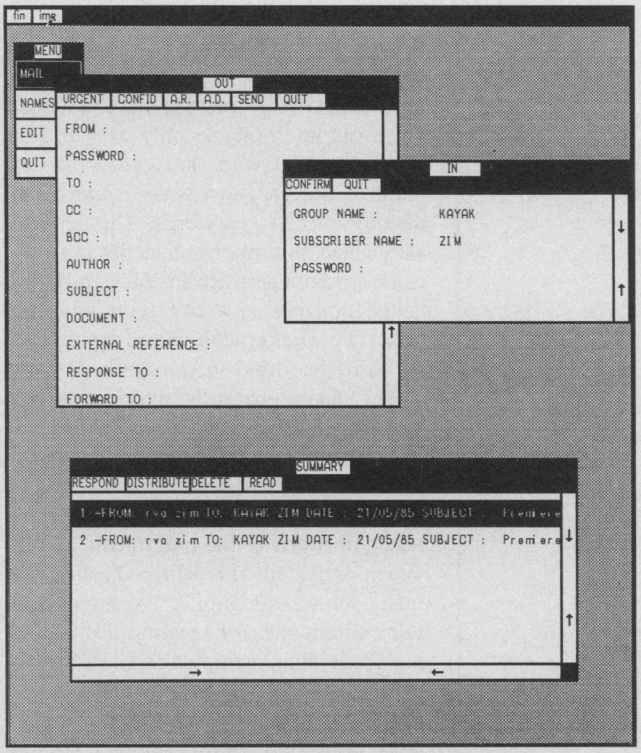

Figure 5. Command selected in multiwindow environment.

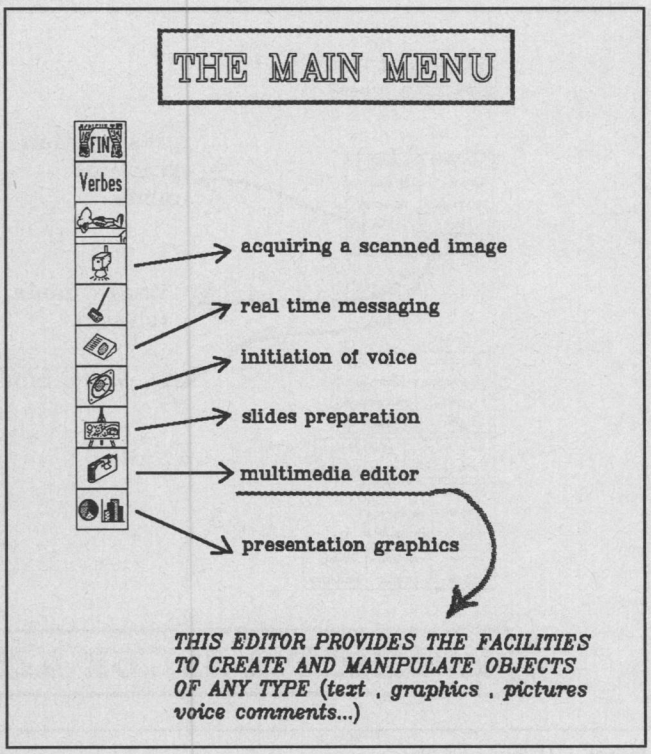

Figure 6. The main menu.

Dialog manager. Agora provides the user with the services of administration, teleconferencing, and mail. The functions included in these services are represented as a tree, which also gives the structure of the command language.

The administration tree has five branches representing five sets of commands: group, subscriber, alias, distribution list, and conference. In the group set, the user can list all groups managed by Agora, or search for a specific group. The subscriber set includes functions to present and modify the user's various parameters. The alias set allows for manipulation of additional names assigned to the user. The distribution list allows the user to create or delete distribution lists, or to manipulate a specific list. The conference set deals with the creation and manipulation of teleconferences and their parameters.

The mail tree contains the traditional commands of creating, filing, retrieving, and redistributing messages.

The teleconference tree allows the user to enter and participate in a teleconference by sending different types of messages (intervention, question, note, vote, and so forth) and retrieving or archiving information exchanged during a teleconference. Figure 4 illustrates the teleconference tree.

The syntax of every command follows exactly the tree structure. The materialization of the tree varies according to the available resources and intelligence of the terminal. For the Buroviseur, where a multiwindow package exists, dialog takes place through selection of commands displayed in a hierarchy of menus that exactly reflects the tree structure. Parameters are entered directly into predefined forms displayed on the screen (see Figure 5).

In the case of teletype, a query-response dialog replaces this interface. All queries and answers are displayed or entered in line mode.

Obviously, the first type of dialog is more practical because of the direct selection of commands with the mouse, the immediate feedback, and the simultaneous views on the screen of the steps of the interaction.

Although the difference between physical representation of both interfaces is important, we have succeeded in providing the same logical interface, and thus the same service, to both categories of users.

The multimedia editor. The multimedia editor used to prepare compound documents on the Buroviseur is fully interactive (what you see is what you get). Is is based on a system of menus and icons inside the menus representing the commands or the objects. At initiation time the system presents a main menu (see Figure 6) containing commands that can be invoked directly. Those that deal with messaging and editing are indicated with arrows in Figure 6.

When using the picture acquisition icon, the user can place a new picture under the scanning device and order digitizing of the image. The image can then be sorted on the disk attached to the scanning device and transmitted to the workstation, where it is automatically displayed. Before deciding to the store the picture, the user can clean it up with the editor and store all or parts of it.

When using the real-time messaging icon, a window opens at the workstation. The user can enter the destination name

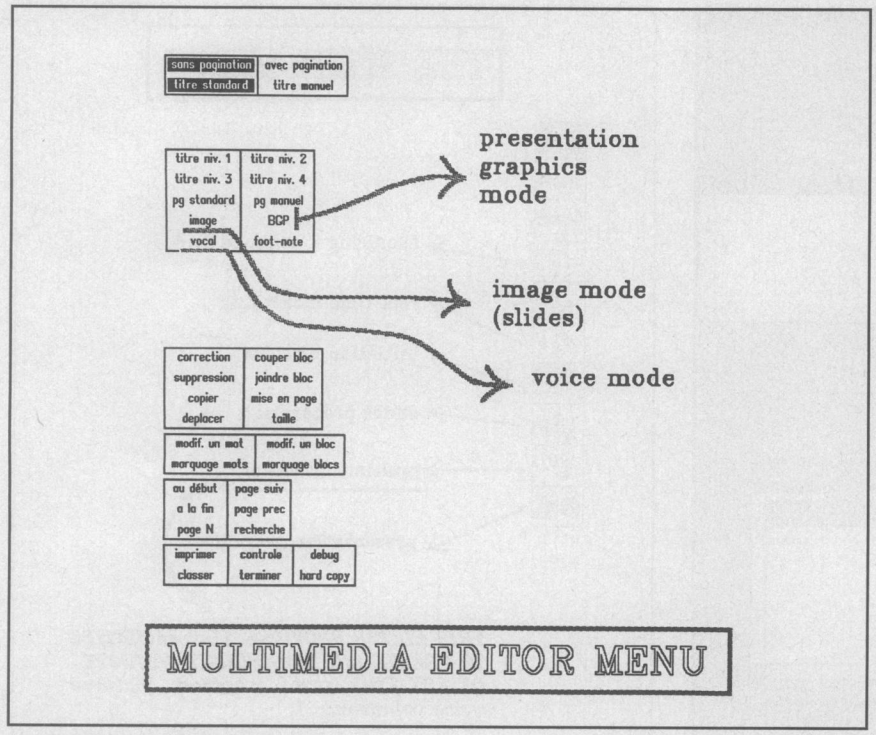

Figure 7. Multimedia editor menu.

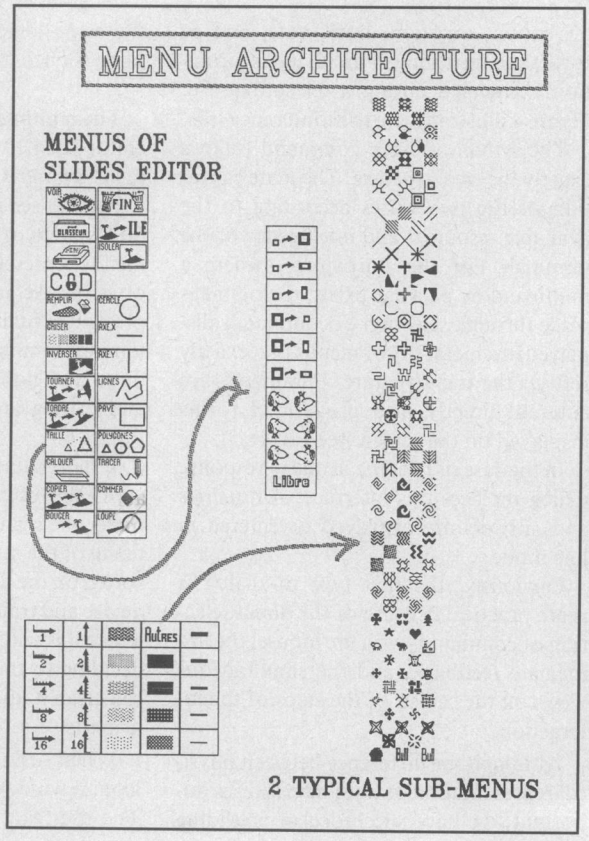

Figure 8. Menu architecture.

(individual or group name) and the text of the message, transmitted directly over the local network to the destination. If the recipients of the message have initiated voice output (possible only at workstations equipped with the voice synthesis feature), the arriving message is translated directly into spoken words. This service, also called Interphone, benefits a group work environment. Recipients can be solicited for a meeting or to execute a specific task. The speaker icon allows the user to authorize or forbid speech synthesis.

The slide preparation and presentation graphics editors provide tools for producing information materials one page at a time. Although they can be invoked and executed independently, the modules are also embedded in the multimedia editor. When activated, this editor displays the menu shown in Figure 7. The menu contains commands for creating text blocks and for joining, manipulating, and browsing these blocks. Figure 7 highlights three particular types of command: image, voice, and presentation graphics. Blocks containing images can be retrieved from the filing system or digitized. The image can be placed anywhere.

The image editor is probably one of the most powerful tools we have. The menu shown in Figure 8 illustrates the rich set of functions provided. All geometric transformations can be applied to various objects. Each transformation has its own submenu. In Figure 8 we see two typical submenus, one indicating the different modes of scaling and the other, the library of textures available. Any object can be slanted and combined with other objects, which simplifies adding captions to pictures or creating images such as that shown in the middle of Figure 9.

Since it is difficult to show how voice is created, we describe in written form the whole process. In voice mode, an open window accepts two types of data: written text to be synthesized, or digitized voice at 64K-bits per second. The voice annotation can be modified only when in the text mode. Once created, it is stored in an attached address. In this respect, it is processed in the same way as a footnote. When presented on a screen, the voice is pronounced only if the voice output is activated (as explained before). In the current block, a special marker indicates vocal annotation. The user can ask for display of the property sheet of any object

including a voice marker and, in this case, hear the attached comment.

Protocols and message structuring

The interaction among Agora components is through appropriate protocols located at the application layer in the ISO reference model. The interaction between two components (entities) is based on the client-server dialog mode. The server performs the service and reports the result to the client. The dialog between client and server takes place in an interactive mode. Each service is represented by an operation code and arguments that are the parameters necessary to perform the operation. Protocol elements are transferred by a message transfer protocol, or MTP. The MTP uses session services to create and maintain a logical session between entities.

Data transfer is performed on a one-way alternate mode on a logical session. The data encoding scheme derives from the NBS proposal for message format syntax encoding,[9] while the message content is structured according to SGML.[10]

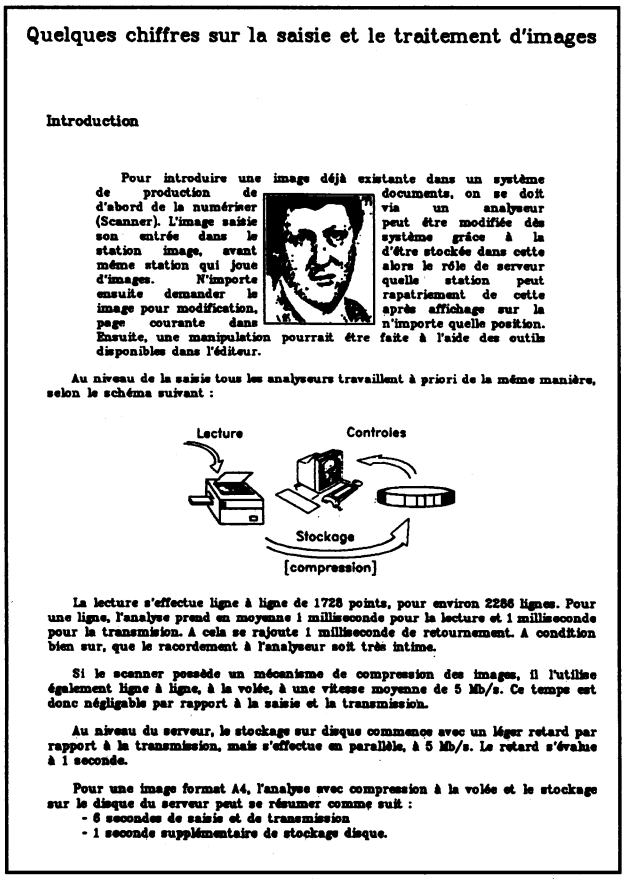

Figure 9. Example of a multimedia page.

The NBS protocol. The NBS standard is based on the fields model[9] that defines the message as a series of fields—more appropriate than the header and text model. Thirty-nine fields have been proposed and divided into three categories: Required, Basic, and Optional. The required fields must be present in all messages and must be processed by all CBMSs.

In Agora we adopted the three fields From, Posted-date, and To. The Basic message fields need not be present in all messages, but must be processed by receiving CBMSs. Four basic fields have been implemented in Agora: Cc, Reply-to, Subject, and Text. The last category, Optional, contains fields that need not be present in all messages and may be ignored by receiving CBMSs.

We implemented 22 fields: Attachments, Author, Bcc, Circulate-next, Circulate-to, Comments, Date, End-date, In-reply-to, Keywords, Message-class, Message-id, Obsoletes, Originator-serial number, Precedence, Received-date, Received-from, References, Reissued-type, Sender, Start-date, Warning-date. The syntax in the NBS format is machine-readable, which makes it flexible, especially when nontextual data is to be presented.

The message is structured in two sets of data elements: Basic (ASCII-string, Date, End-of-constructor, Field, and Message) and optional data elements (14 types). We mention in particular Bit-string, Compressed, Property-list, Extension, and Vendor-defined because of their use in Agora. The envelope was implemented as a Vendor-defined data element. Other types, such as Extension or Property-list, could be considered for encoding voice and fax. Naffah and Mazoyer argued these options.[12] However, we adopted the SGML language for the message content structure because it better suits the nature of complex documents transported by the message systems.

The NBS processor takes approximately 2000 lines of Pascal.

Message content architecture. The content of the message is usually called Document. Within this message field all of the semantics of the message should be embedded. When we examined the office environment and analyzed the documents exchanged, we found it mandatory to provide (in CBMS) a rich document architecture and thus a sophisticated document representation for the following reasons:

- Multimedia environment: Office documents may include text with a variety of languages and alphabets, geometric graphics, facsimile pictures, mathematics, chemistry, professional symbols, voice, or a combination of these data types.
- Processing environment: Some of the documents will be stored in databases, which necessitates their storage in revisable form. Other documents may be sent to formatting or printing systems. Therefore, they are sent with physical layout parameters, or in final form.
- Logical environment: Documents may vary from a simple note or memo to a book of many volumes. Those logical entities are built according to a tree structure of elements, which may have complex logical links in addition to the tree-like relationship. Documents can also be linked together.

In Agora we adopted the SGML representation to encode the content of messages exchanged between UAs and message servers.

For these reasons, document architecture should be chosen with great care. The first proposals for multimedia protocols came from the Arpanet community.[13,14] Recently, international standards groups such as CCITT and ISO started working on document standardization and transfer protocols. At the CCITT, T73 is proposing a structure for mixed-mode teletex service. ECMA and ISO proposed ODA-ODIF (office document architecture and office documentation interchange format). In addition, ISO is considering the SGML (Standard Generalized Markup Language), also elaborated upon by an expert group called CLPT (Command Language for Processing of Text). Xerox has proposed Interscript.[15] The current position of the standards groups is to combine the two families.[16]

In Agora we adopted the SGML representation to encode the content of messages exchanged between user agents and message servers. During the editing process, all documents are automatically externalized in SGML format and sent in the Text field of the NBS format to a remote message processing end. Documents retrieved from mailboxes are internalized by UAs for editing or display on the screen.

Interconnection with public message services

The Telex case. The Telex network constitutes the largest text-exchange service in the world, with millions of telex workstations in different countries. Actually, it is the simplest form of electronic message sending, distinguished from the present CBMS by the lack of a store-and-forward service. To enhance this service, the CCITT has adopted the Teletex service, in which simple telex terminals are replaced by sophisticated Teletex terminals having editing and storage capabilities.

Whether telex today or Teletex tomorrow, it is essential that any CBMS aimed at a large community of users provide special ports to this "telematic" terminal. Therefore, we decided to include the telex interface in the Agora project and provide the interconnection of CBMS and telex services in both directions. The major challenge in accomplishing this is solving the problems of coexistence between two services with different levels of functionality at the terminal level.

A telex agent built in special hardware is connected on one side to the telex network, where it is known by a unique international telex number. On the other side it is connected to the Danube network, where it has a mailbox in the message server. The agent periodically polls its own mailbox to check the content. Whenever it finds a message, it takes it out and sends it through the telex network. When the agent receives a message from a distant telex terminal, it acts as an MPE and submits the message to the appropriate mailbox in the message server.

A naming convention has been adopted: The telex subscribers are registered under one group, called Telex, with each member having a simple name as his international telex number. The sender of a message from the Buroviseur or a telex terminal indicates the Agora naming structure after the telex number.

Conversion problems arise when the content of the message sent by a Buroviseur contains data types that cannot be interpreted by the telex terminal. The telex UA identifies these data types (graphics, voice, facsimile) and replaces them with textual comment, indicating to the destination that parts are missing.

Interconnection to public MTS. Recently, CCITT adopted a model of message handling, or MH, services the administrations will provide to enable subscribers to exchange messages on a store-and-forward basis.

Two MH services are offered. The interpersonal message service supports interpersonal communication, while the message transfer service, or MTS, supports transfer of messages independently of application. Recommendations X.400[11] define the protocols that apply between system components.

An originator prepares and submits messages with the assistance of an application process called the user agent. The MTS delivers the messages to one or more recipient UAs. UAs interact with MTS using the submission and delivery procedures defined in the P3 protocol (recommendation X.411). MTS consists of a number of message transfer agents, or MTAs, that operate together to accept messages from originating UAs, relay these messages, and deliver them to recipient UAs. MTAs interact according to protocol P1 (recommendation X.411). The interpersonal message service provided by the UAs in cooperation uses protocol P2 (recommendation X.420). Figure 10 shows a layered representation of this model, the various cooperating entities, and the types of protocols that apply.

To promote this model in France, the French PTT laboratory CNET launched a project called Cosac to interconnect existing CBMSs according to the standard protocols. We were asked to make Agora compatible with these protocols. The basic choices made include

(1) Agora should apply P1 and P2 protocols to exchange messages with MTS and provide interpersonal message service to end users.

(2) The naming strategy for Cosac used country-name: Fr; administrative-domain-name: PTT; private-domain-name: rva. For Agora, country-name: Fr; administrative-domain-name: PTT; private-domain-name: agora.

(3) In terms of functions, we selected common functions provided by both services. For some functions not provided by Agora, we could only generate the corresponding protocol element (such as Probe), but could not carry out the processing upon reception. On the other hand, some functions provided by Agora were not authorized to pass through the Cosac gateway. The following lists the functions supported at both P1 and P2 levels:

(a) message transfer services, P1 level—Basic (message identification, nondelivery notifications), Submission and Deliv-

ery (delivery notification, delivery time stamp, disclosure of other recipients, grade of delivery selection, multidestination delivery, prevention of nondelivery notification, submission time stamp), and Query (Probe-generated only), and

(b) UA cooperation services, P2 level — Basic (basic message transfer service elements, user message identification, typed body), Submission and Delivery (same as message transfer service), cooperating UA information (originator indications, primary and secondary recipient indications, importance indication, sensitivity indication, subject indication), and Query (Probe-generated only).

The implementation has taken the form of a dedicated gateway composed of three modules:

- an MPE to interface Agora components,
- a conversion module to perform the adaptation of functions and protocols, and
- an MTA to interface Cosac with P1 and P2 protocols.

The gateway was written in Pascal under the UCSD system. A proprietary real-time executive was added to deal with concurrent tasks. The executable code occupies 80K bytes of memory.

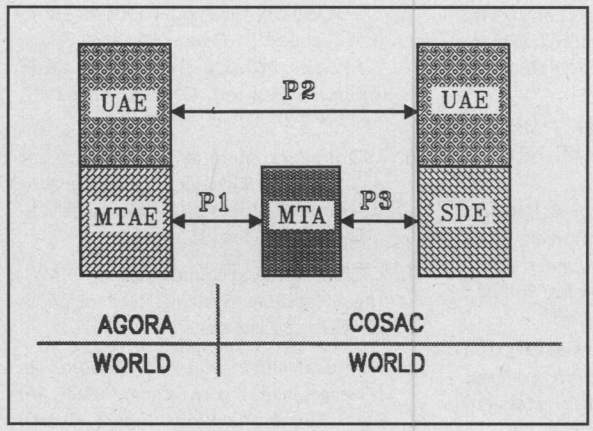

Figure 10. CCITT model and Agora as a private domain.

Agora took a long time and covered a number of interesting areas in office systems, including distributed systems architecture, with particular algorithms for updating distributed databases (name servers) and solving the consistency problems; multimedia messaging; conversion between services of different qualities; man-machine interface consistency to the same service across different types of terminals; and validation of X.400 recommendations (P1 and P2) in a heterogeneous environment.

The lessons we learned during this experience have influenced most of the prototypes and products currently under development in different research centers in France. However, many issues remain needing solutions and many solutions need refinement.

In distributed systems, we encountered major problems in name management in distributed directories and the introduction of a new set of high level protocols in the system components. The management of names should satisfy two conflicting criteria: service flexibility and directory consistency. The service should be flexible enough to allow the user to change his attributes whenever he wishes to do so and at the same time, keep data consistent in the distributed databases of directories. We proposed an algorithm to solve this problem. However, work should continue to validate the scheme in a real international environment, where public and private directories will work together.

The other issue concerns the X.400 protocols (P1, P2, and P3) proposed by CCITT. The introduction of these protocols led to the addition of 80K bytes of code to the protocols of the five other layers of the open system architecture. We believe that an economical solution would be to split the functions into two parts, the user interface and protocols management. We would put the former into the workstation and the latter into a dedicated server shared by a group of workstations located at the same site.

We have shown this concept to be feasible in the multimedia area. Few other systems have launched similar projects.[17]

Two issues need solution before putting these systems into real offices. The first deals with ergonomics, or human interaction with multimedia documents. Current solutions for display seem acceptable, relying on menus, icons, and windows. However, interaction with voice segments remains in its infancy. We have seen that customization of the workstation by authorizing or prohibiting voice is an important feature, but we don't yet know the impact of editing voice in communal situations.

The second issue relates to the ability of multimedia systems to be generalized, with adequate workstation support in the office. The present situation looks very promising for graphics, and we all expect a spectacular evolution there. However, the embedding of voice in commercial systems messages remains uncertain. We shall simply have to wait and see what happens. □

Acknowledgments

Many people contributed to the design and implementation of the Agora project. We would like to thank G. Frantz, J.M. Grugeaux, and M. Hue for their work in the mainframe version. We are grateful to E. Ouze and G. Grierè, who participated in the micro version, and to D. Petonnet and P. Gries for their efforts in the telex agent. J.P. Zimmerman was the major implementor of P1 and P2 protocols.

Many thanks to the Kayak team, who built the Buroviseurs, the microservers, and the network, and who provided the ideal environment for the Agora experiment.

Finally, we would like to thank the reviewers for their comments, and in particular, J.J. Garcia-Luna Aceves for the advice he gave us in restructuring and refining this article, and the help in editing it.

References

1. N. Naffah, "Distributed Office Systems in Practice," Online Conference, London, May 1982.

2. IFIP WG6.5, "Reports of the 2nd and 3rd Meetings of the System Environment Subgroup," IFIP-WG6.5 N16, N17, 1979.
3. T.H. Myer and J. Vital, "Message Technology in the Arpanet," NTC 77, IEEE, Dec. 1977.
4. C.P. Thacker et al., "Alto: A Personal Computer," in D.P. Siewioreck, C.G. Bell, and A. Newell, *Computer Structures: Principles and Examples,* 2nd ed., McGraw-Hill, New York, 1981.
5. B. Scheurer, "Office Workstation Design," *Office Information Systems,* ed. N. Naffah, St. Maximin, N. Holland, Oct. 1981, pp. 105-116.
6. A. Wegmann, "Vitrail: Window Manager for an Office Information System," *Proc. 2nd Conf. Office Information Systems,* ACM, Toronto, June 1984.
7. N. Naffah, "Editing Multitype Documents," *Office Information Systems,* ed. N. Naffah, St. Maximin, N. Holland, Oct. 1981, pp. 125-148.
8. J.P. Ansart et al., "Danube Interconnection via Transpac," *Local Computer Networks,* eds. P.C. Ravasio, G. Hopkins, and N. Naffah, N. Holland, Florence, Apr. 1982, pp. 279-287.
9. D.P. Deutsch, "Design of a Message Format Standard," *Proc. IFIP Int'l Symp. Computer Message Systems,* ed. R.P. Uhlig, N. Holland, Ottawa, Apr. 1981, pp. 199-220.
10. ISO draft proposal 8879/6, "Processing and Markup Language," part six, generic document representation specification—SGML, Jan. 15, 1985.
11. CCITT-X.400 recommendation, "Message Handling Systems: System Model—Service Elements."
12. N. Naffah and B. Mazoyer, "Multimedia Messages in the Agora System," *Proc. 8th Data Communications Symp.,* North Falmouth, MA, Oct. 1983, pp. 194-196.
13. J. Postel, "Internet Multimedia Mail Transfer Protocol," RFC 759, USC Information Sciences Institute, March 1982.
14. J.J. Garcia-Luna Aceves and A. Poggio, "Multimedia Message Content Protocols for Computer Mail," *Proc. IFIP 6.5 Working Conf.,* Nottingham, England, May 1984, pp. 85-96.
15. V. Joloboff and T. Schleich, "Introduction to Interscript," Jan. 1985. Also Xerox contribution to ISO, ref. ISO/TC97/SC18/WG3-N439.
16. ISO-TC97-SC18-WG8-N36, "Relationship of ODA, ODIF, and SGML," Ch. Goldfarb et al., Oct. 1984.
17. H. Forsdick and R. Thomas, "Initial Experience with Multimedia Documents in Diamond," *Proc. IFIP 6.5 Working Conf.,* Nottingham, England, May 1984, pp. 97- 111.

Najah Naffah is head of the advanced studies department in office systems at Bull Transac, where he manages a group of researchers in various areas, such as workstation design, document production, database management systems, and artificial intelligence-based applications. Prior to this, he was leader of the Kayak project at INRIA-Rocquencourt in France and researcher for Cyclades, a datagram-based computer network.

Naffah received two degrees in engineering from ESIB in Beirut and ENST in Paris. He also holds a PhD in computer science from the University of Paris.

Naffah serves as an associate editor for *Trans. on Office Information Systems.*

Ahmed Karmouch received the MS and PhD degrees in computer science from the Paul Sabatier University of Toulouse, France, in 1979.

He started in distributed databases in the Sirius project at INRIA (Institut National de la Recherche en Informatique et Automatique). In 1980 he joined the Kayak project at INRIA, responsible for the message system group. In 1984, Karmouch joined Bull Transac, where he is responsible for the department of advanced studies of the multimedia documents management project.

Readers may write to the authors at Bull Transac, 1 Rue Ampere, BP 92 91301 Massy, France.