An Object-Oriented Distributed System Integrating Multimedia Resources

Qianshan He and Hidehiko Tanaka
Department of Electrical Engineering
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

Abstract

In this paper, we introduce a distributed system which provides extensible as well as integrated environments for users and application programs to access and combine distributed resources conveniently. The system is based on the concept of object-oriented service bases. Distributed resources are abstracted as objects containing the meta-information of the resources. A service base in each node is constructed to manage these objects. Three layered views (external, conceptual and internal view) about objects are divided. A service base can be seen as a software package built on an existing operating system to provide mapping from the external view of objects to the internal view of objects to realize transparency and to absorb heterogeneity and different media types. We present the modeling and an implementation of this distributed system in this paper.

1 Introduction

Recently, developments in networks, especially the high bandwidth and high speed networks (for example, ISDN and LAN), make the sharing of multimedia data in a distributed environment available and practical. Moreover, Developments in workstations provide a powerful interface for users to create or display data of different media types (image media, text media or graphic media). On the other hand, In this kind of distributed environments, the numbers and types of data available to users are being expanded, users may have troubles to access and manipulate these data. Application programs, functions provided by computers, operations on multimedia data are becoming more complicated (we call these multimedia data, application programs and functions resources). Therefore, it is necessary to use an uniform method to manage these distributed resources and to provide convenient and integrated high level services to the users.

In this paper, to solve the problems above, we propose a distributed system based on object-oriented service bases. We use the object model to represent resources. In a service base, three layered views of objects are used to manage objects. Distributed resources can be used conveniently by users and application programs though an uniform interface without worrying about the different types of media or the heterogeneity of computers. Our design considerations emphasis on providing an integrated, extensible and easy to be constructed distributed system.

This paper is organized as follows. In section 2, we define what is a distributed object and present the method to manage distributed objects in our system. Section 3 describes the structure of an object-oriented service base. Section 4 presents an experimental system and discusses the implementation issues. Finally, in section 5, some conclusions regarding the characteristics and application aspects of the system are discussed.

2 Modeling

2.1 What Are Distributed Objects?

In a computer system, processors, memories, I/O, files and networks are usually managed by an operating system which provides higher level functions to users. On the top of the operating system, users can see various programs and data: a program can be a function provided by a computer (for example, a command of an operating system), or an application program created by an user with a certain language to perform a specific work; data may be expressed in different media types (for example, image media, text media). An application system are constructed by combining these programs and data. By resources in our system, we refer to these high level functions, data and application programs. Usually, using the resources can be described as: "An operation

is imposed on data and certain results are produced". The operation may be a function provided by computer (for example, a compiler or an editor) or an application program. This concept can be extended directly to the object-oriented concept: an object contains its data and operations on the data. In this way, using the object model to express resources is natural.

On the other hands, In a distributed environment, resources may have their more complicated structures. For example, where are the data? Where are the programs executed? Who possesses these data and programs? What media types are the data? Are the data or programs in disk or in memory? We call these information meta-information of resources. When abstracting a resource as object, we include these meta-information in the conceptual views of the object. In the external view of a distributed object shown to users, these meta-information is unnecessary. In other words, network transparency should be realized in the users level.

Object model has the characteristic of information encapsulation ability. This characteristic provides us a good method to overcome the different types of media and the heterogeneity. We include the meta-information of a resource inside an object as its attributes. Therefore, only the external interface (specifications) of a resource are shown to the users, the structures or implementation details of the resource are encapsulated inside the object. The attributes of the object will be used by the system to absorb the different types of media and the heterogeneity. Object model also have good modularity. This characteristic provides a way to extending resources. In a distributed environment where the numbers and types of resources available to users are being expanded, this extensibility is very significant. If the common structure of the resource management system has been defined, then adding a new type of resource is a simple matter by defining a new class for that specific type.

2.2 How to Manage Distributed Objects?

In each node of the networks, a service base is built up to manage the local objects and cooperate with other service bases in other nodes for accessing remote objects. Distributed *services* are provided on top of the service bases by combining local or remote objects. Inside a service base, as shown in Figure 1 , three layered views about objects (*external view, conceptual view, and internal view*) are divided:

- The external view is the view shown to users or application programs. Objects in the external view are transparent and easy to be used.
- In the conceptual view, meta-information of a resource is abstracted as an object. The difference of media types, heterogeneity and location distribution of resources are absorbed in this layer.
- An object in the internal view is the entity of a resource. The internal view about an object concerns its physical realization in a computer. Objects in the conceptual view are reflected to their respective objects in the internal view.

A service base processes the queries from users or application programs by mapping the objects in three layered views: when a service request (which may contains several objects) is invoked in the external view, the request is mapped to the objects in the conceptual view. According to the location attributes of the objects in the conceptual view, the objects in the conceptual view are mapped either to the objects of the internal view in the local node or to the objects of the external view in a remote node. We will take an example in section 3.2 to show how the mapping of objects in different views works.

In the following sections, we will discuss the details of object management methods in each layer: object managements in the conceptual view are based on a directory and a database; in the external view, a browser and a library are provided as interface for users and application programs to access objects; object managements in the conceptual view are based on the existing operating systems or database management systems.

2.3 System Configuration

We suppose that the system is built on a long-haul network or a local area network (LAN) environment. In a LAN environment, administrative of machines according to subgroups of an organization will be natural. We call these subgroups *clusters*. In a cluster, as shown in Figure 2 , there may be some servers such as file servers and print servers which are shared by the machines inside the cluster. Because of the high speeds of the LAN and the uniform administrations, a centralized management configuration will be efficient within a cluster: that is, a service base in each node are constructed to manage local objects and provide objects to other nodes, all objects provided by the cluster are registered centrally in the service base of the cluster server (the cluster server should

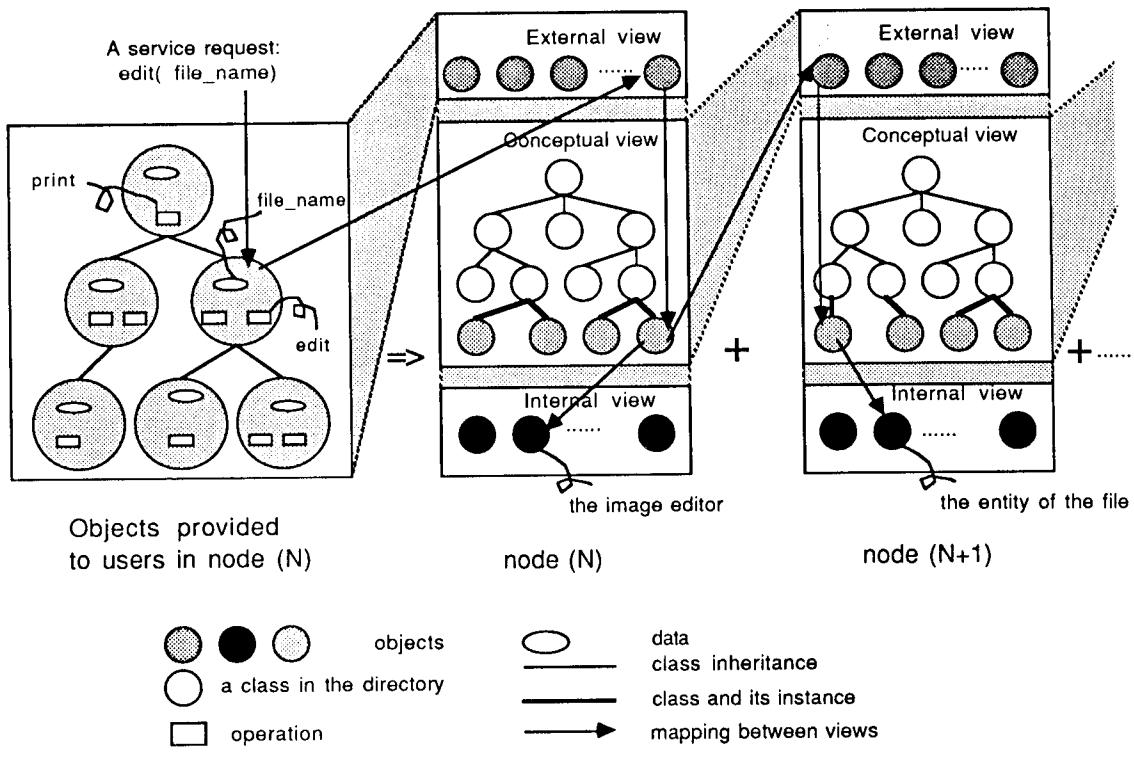

Figure 1: The Three Layered Views and the Mapping of Views

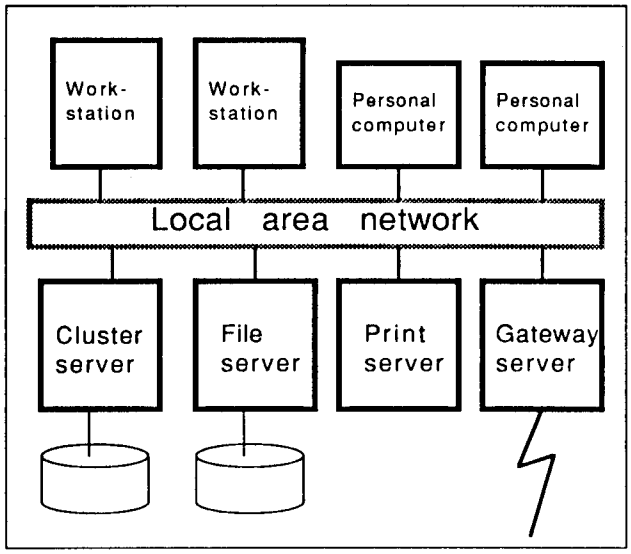

Figure 2: Configuration of a Cluster

be a node which has fast responding time and massive secondary storage inside the cluster). The cluster server acts as an overall guidance for the nodes in a cluster. When a request of some objects is invoked in a node of a cluster, the service base in this node first checks if objects are located locally, if not, a further request to the cluster server is invoked. Based on the information about these objects in the cluster server, the objects are fetched from a service base of a node inside the cluster. If the objects are outside the cluster, the cluster server broadcasts the request to the cluster servers in other clusters.

Nowadays, most of the long-haul networks are connected by a number of local area networks. In a long-haul network environment, we can consider that the system consists of many centralized clusters. Objects management within one cluster is just like that of LAN environments. More details of the discussions about the configuration of the system in a LAN or a long-haul network environment can be referenced to [4].

Workstations, personal computers and servers (file servers, cluster servers etc.) may have their different emphases. A workstation usually acts as an access point to manipulate or display multimedia data. This means that the user interface to the service base in a worksta-

tion should provide powerful display functions such as multiple windows and graphical displays. On the other hand, servers dedicate to provide objects. Large secondary storage systems and service bases with fast responding time are important in these servers.

2.4 The Position of the System

Our system is constructed in a relatively high level. A service base in each node is built on an existing operating system and the OSI communication interface. The existing operating system can be a network operating system or a distributed operating system [1], or just a single operating system in each heterogeneous machine with minimum communication facilities. We only define high level protocols. A concrete method to establish an actual network system and its low level protocols is left to system creators. Our system can be seen as a framework which integrates various services in the application layer. In the future, with the development of fundamental protocols and protocols used for various applications in OSI, it is important to have some systems to combine these protocols for applications. To establish the unified structure of the application layer will be important. Our system is expected to be useful in these area.

3 Construct a Service Base

The structure of a service base in a workstation node, a personal computer node or a cluster server is about the same. The only difference is that a service base in a workstation or a personal computer contains the information of the local objects, but a cluster server contains the information of objects inside a cluster. In a workstation or a personal computer, if a requested object is not local object, it will send a massage to its cluster server, the cluster server tells which node or cluster the requested object is in.

Figure 3 shows the structure of a service base. We structure the service base (including the interface) as a collection of objects. The *directory* classifies all objects which can be used though this service base. The *query analyzer* object is responsible to handle the queries sent from users or application programs, and to perform mapping of the three layered views. The *communication module* object provides communication interface between objects in different nodes.

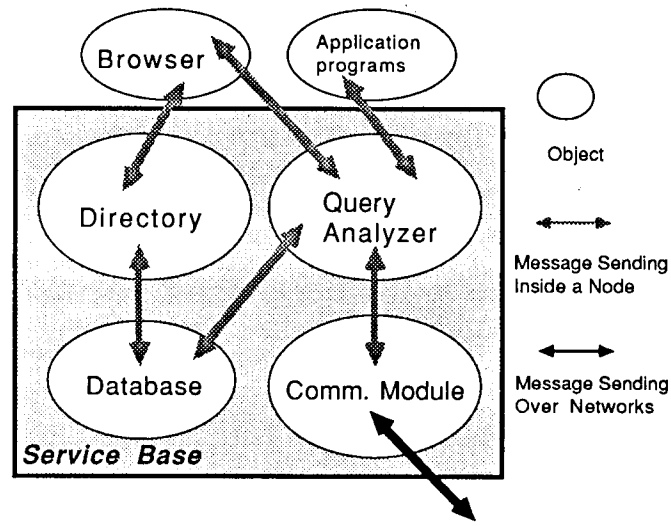

Figure 3: The Structure of a Service Base

3.1 Directories

In the conceptual view of a service base, the meta-information about a resource are abstracted as an object. To classify various kinds of objects, a directory is created in each service base. The directory indicates the characteristics of resources: for example, what attributes a certain kind of resources has? What operations can be performed on this kind of resources? What is the relationship between this kind of resources and other kind of resources? etc.. The directory contains a collection of classes. A class is defined to classify resources which have the same characteristics. For each class, the operations which can be applied to this class are defined. A class may have its superclass. The attributes and the operations of the superclasses are inherited by its subclasses. In this way, a subclass only need to define the attributes which are different from its superclasses. We structure the directory as a single-inherited class collections. The attributes of a class are expressed by instance variables of the class, and operations are expressed by methods. Figure 5 shows an example of classes in the directory (this example will be explained later). In this example, C++ [7] language is used to define the classes.

Directories follow a tree structure in which information are hierarchical (Figure 4). We suppose directories in different nodes may have their different tree structures according to their environment. This enriches the extensibility of each node, and make our system applicable to different environments. On the other hand, we define some minimum standards to ensure the whole system

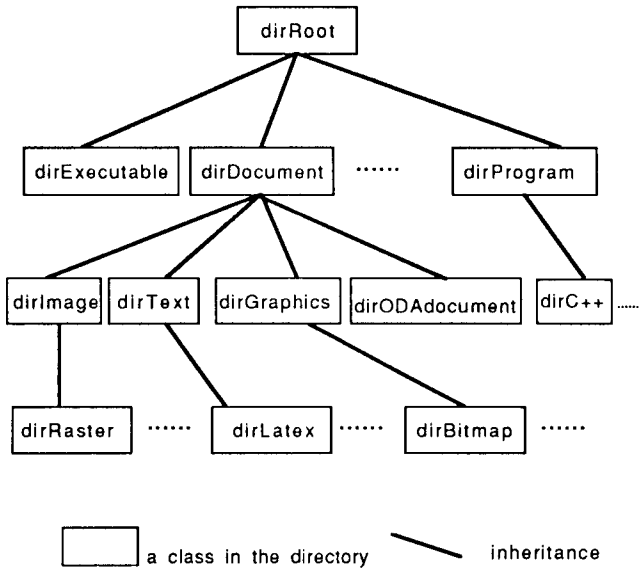

Figure 4: The Directory in a Service Base

```
class dirDocument : public dirRoot
{
    Attribute name;
    Attribute location;
    Attribute format;
    Attribute mediaType;
    Attribute creatTime;
    Attribute owner;
    ......
public:

    void multimediaMail(char* dest);
    void multimediaEdit();
    ......
}

class dirODAdocument :
      public dirDocument
{
    Attribute format = ODA_DOCUMENT;
    Attribute mediaType = MULTIMEDIA;
    Attribute status;
    ......
public:

    void editingProcess();
    void layoutProcess();
    void presentationProcess();
    ......
}
```

Figure 5: Class Examples in Directories

```
AttributeValue locationValue[]={
  /* node name:  */
  ENTERPRISE,
  DISCOVERY,
  ......
};

AttributeValue formatValue[]={
  ODA_DOCUMENT,
  LATEX,       /*text in latex  */
  C_PROGRAM,   /*source in C    */
  RASTER,      /*image in raster*/
  ......
};

AttributeValue statusValue[]={
  /* for ODA_DOCUMENT: */
  PREVIOUS_CONTENT,
  EDITED_CONTENT,
  LAYOUT_CONTENT
};

AttributeValue mediaTypeValue[]={
  MULTIMEDIA,
  TEXT,
  IMAGE,
  GRAPHICS
};
```

Figure 6: Some Attribute Values in Directories

can co-operate with each others. This means that we must define some classes and attributes which are common to variable objects. In an actual implementation of the system, classes and attributes which are not defined in the standard can be added to a directory to make the system suitable to certain applications. For example, if a new type of media or a new type of functions is to be added to the system, then a new class for that specific type should be defined.

In a heterogeneous environment or a multimedia environment, attributes of the objects are very different from each other. How to handle the attributes appropriately is the main point of integrating them as a whole system. In our experimental system, we only define parts of attributes and attribute values (some examples of attribute values are shown in Figure 6). How to include the attributes and attribute values which are necessary to an environment and exclude the attributes which are unnecessary is remained as a problem which should be solved in an actual implementation.

For the attribute values in the classes, we define their syntax and semantics. An attribute value is expressed by some key works which are predefined by the system. An attribute may have several attribute values, their

relationship follows a certain syntax rules (for example, "and", "or" relationship). These limitations make actual system constructions possible. The structure of classes, attributes and attribute values forms the directory schema.

A directory class is only a description of one kind of resources. To register a resource in the service bases and to make it available to network users, a class instance (object) which contains the specific attribute values of the resource should be produced and registered in a database.

We take an example here to show how the directories are structured and used. As shown in Figure 5 and Figure 4, a *dirDocument* class is defined. According to the attribute *format* of class *dirDocument*, the subclasses of *dirDocument* can be class *dirODAdocument*, class *dirText* and class *dirImage* etc.. All attributes and operations of class *dirDocument* can be inherited by its subclasses.

The class instances of *dirODAdocument* are the documents which follow ODA standard of OSI [2] [3]. Besides the attributes in class *dirDocument*, class *dirODAdocument* may contain the attributes like *status* which shows the status of the document. The attribute value of *status* (in a class instance) can be *PREVIOUS_CONTENT*, *EDITED_CONTENT* and *LAYOUT_CONTENT* etc.. [1] The operations in *dirODAdocument* may contain *editingProcess*, *layoutProcess* and presentationProcess etc.. To a specific document, an object which contains certain attribute values is created and registered in a database. To print this document (that is, to call the *presentationProcess*), the service base first locates and fetches the document (according to the *location* attribute), then checks if this document is printable (attribute *status* is *LAYOUT_CONTENT* or not). If not, say, attribute *status* is *EDITED_CONTENT*, the service base will automatically call the *layoutProcess* service to produce a *LAYOUT_CONTENT* of the document, then print it. In this example, we can see logically that service bases provide a new service that can print a document which is not in printable status by combining some existing registered services (*layoutProcess* in the example). This is only a simple example. When large numbers of services are registered in the service bases, logically more functions can be supplied to users by combining these services.

[1] ODA distinguishes between a document's logical structure and its layout structure. Before logical structure editing, the content of a document is called previous content. After logical structure editing, the content of a document is called edited content. After layout structure generation, the content of a document is called layout content and can be presented on a medium (paper, screen) through a presentation process.

3.2 Mapping of the Three Layered Views

We use a simple example to show what relationship the objects inside one node or outside the node have and how they are mapped each other. This example is show in Figure 1. We suppose there is a file registered in the service bases, its name in the external view of node(N) is called *file_name*. An user in node(N) want to display this file. We suppose he does not know where the file is actually located and what media type the file is expressed by. Therefore, he is not certain to use what kind of editor to display this file. The logically inherited objects tree in the external view of node(N) (this tree is shown to users in a command shell window) indicates that this file can be imposed various operations: it can be displaied by using the *edit* operation or printed by using the *print* operation (an operation which is inherited from the super-object) etc.. After the request *edit(file_name)* is inputed to the service base in node(N), the file (which actually is an image file) is displaied in an image editor in node(N).

Inside the service bases, the object in the external view of node(N) is first mapped to an object in the conceptual view in node(N). The object in the conceptual view (meta-information of the resource) indicates that the file is located in node(N+1) and the media type of the file is image media, and that there is an image editor in node(N) which can display this file. Then, the operation part of the object in the conceptual view is mapped to an object in the internal view in node(N) (the entity of the resource), and a request is sent to node(N+1) to map a object in the external view in node(N+1). In node(N+1), same processes of view mapping are done and finally the file is copied to node(N) to be displaied by the editor.

If node(N) is called front node, node(N+1) is the back node of node(N), and node(N+2) is the back node of node(N+1). A front node may has its several back nodes (logically, when the front node can send a request to a certain node, we call this node back node.). By means of mapping between the conceptual view of front node and the external view of back nodes, objects provided to the users in the external views of node(N) are the sum of the objects in the internal view of node(N) and the objects provided to the users through the external view of node(N+1). The objects provided to the users through the external view of node(N+1) also follows this way. In other words, through the cooperations of service base in each node, users in a node can access a large number

of transparent objects through the external view, and each node can extend its resources independently in the internal view.

From this example we can also see that logically the objects in the external view of node(N) have a global inherited relationship. Actually node(N) does not contain a global object pointer table. The global object image is realized by the so-called forwarding pointers: the objects in the conceptual view of node(N) have pointers to the objects in the external view of its back nodes, and the objects in the conceptual view of the back nodes have forwarding pointers to the objects in the external view of the back-back nodes.

3.3 Query Analyzer

Query analyzer is responsible to handle the queries sent from users or application programs, and to perform mapping of the three layered views. When an user or an application program makes a request (consisting of objects in the external view) to the system, query analyzer first maps their respective objects in the conceptual view, then searches the database to get the attribute information of the objects. Basing on the attributes of the objects, query analyer in a service base works as follows:

- *Location checking:* if the requested objects are local objects, the objects in the conceptual view are mapped to the objects in the internal view. If some of the requested objects are remote objects, messages are sent to the cluster server to locate the objects. Data objects will be fetched to the node where the requested operation is located. Fetching means replication of objects. If there is a network file system supported by an operating system (for example, NFS of SUN[6]), it will be efficient to only mount the objects dynamically rather than copy the objects.

- *Semantics checking:* Before the requested objects are combined, the semantics of each object are checked based on the attributes of the object. This is important in the different media types or heterogeneous cases. For example, to display an image data, a window which can display image rather than text should be used.

- *Combining:* Combinations of objects can be explicitly defined by the user in an application program, or automatically made inside the service base. The combinations of objects inside a service base mean that after the semantics checking, query analyzer finds some suitable registered objects to meet the needs of the user's request.

- *Executing:* If the operations of an object are local, the query analyzer forks processes to execute the operations in the internal view.

3.4 Communication between Objects

Inside the service base of a node, communications between objects are supported by object communication mechanism of the language which implements the service base. As discussed above, query analyzer, browser, objects in the directory and communication module are all objects, their communications follow this pattern.

Communication between objects which exit in different nodes is supported by the communication module object shown in Figure 3. After the source object sends a message, control is passed to the communication module. In other words, objects inside a service base are not directly communicate with objects in other nodes. Therefore, it is not necessary that the language which implements our system is a distributed language. In the communication module, two kinds of communication facility are provided: remote procedure call (RPC) and message passing. For example, in our system, to execute a service is to combine objects which are related to this service. When the operations of an object are local and the data of the object are remote, the data are fetched to local machine through message passing. When the operations are local and the data are remote, the data and control are migrated to the remote machine through RPC. In our experimental system, the communication module is implemented by the *socket* primitives in UNIX. The socket function covers protocols under session layer.

4 An Implementation of the System

4.1 Environment

We have implemented an experimental system based the model proposed above. This experimental system is used to detail the problems which occur in an actual implementation of the model and and to show the effectiveness of this model. Multimedia data can be included in the system. The network used is an Ethernet-based LAN which is located in the Faculty of Engineering, University of Tokyo [4]. Through a gateway server, this LAN can link to a long-haul network called *junet* in Japan. The

junet mainly provides mail facilities. Our experimental system contains a number of SUN3, SUN4, VAX workstations. The experimental system is written in object-oriented language C++.

4.2 Database

As discussed above, in the conceptual view, a disrectory is used to classify resources. For each resource, a class instance (object) is created and registered as a persistent object in a database.

In an actual implementation, we must make choice to use what kind of databases (for example, relational database or object-oriented database). The external view and the conceptual view of our system are implemented by object-oriented language C++. It will be efficient for object-oriented language to access an object-oriented database. Actually, in an object-oriented database, the language and the database are usually designed as a whole system [5]. But in our experimental environment, object-oriented database in each node is not available. Therefore, we decide to use a relational database called INGRES [8]. To use a relational database, the main problem is focused on how to map the directory schema to the schema of the database. To represent the directory schema which is a tree structure in a normalized relational form will need a lot of relations (tables). To solve this problem, we use unnormalized relational form to represent the directory. In the database, each directory class is represented by one relation, and several relations are used to keep the relationship of the directory tree.

4.3 Interface

In the external view, a standard query language C++ is provided as a standard interface for application programs to access objects managed by the service bases. Moreover, we provide a browser for users to manipulate or display objects. The structure of the interface is shown in Figure 7.

The browser consists of command shell windows, text windows, image windows and graphic windows etc.. Until now, we have not implemented a window which can display all objects in different media types. Each kind of window is used to display or edit objects with the relative kind of media types. The selection of using which window to display or edit a multimedia objects will be made by the system automatically according to the attributes of the objects, say, image data will be display in an image window. To display a document which contains elements

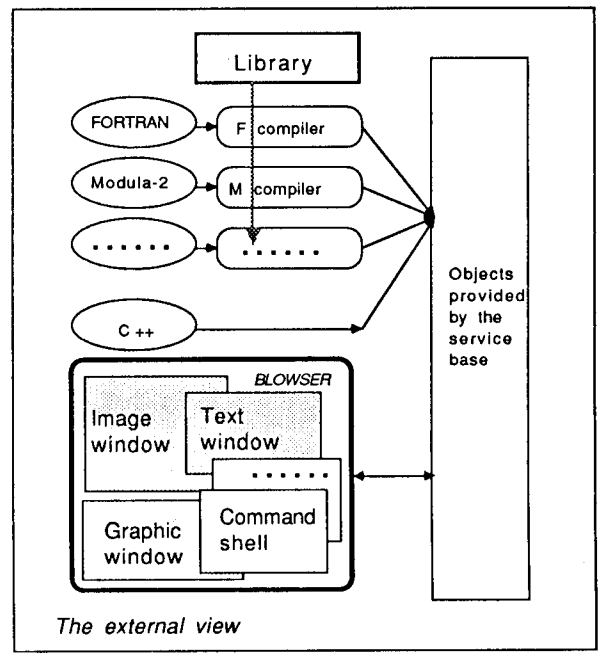

Figure 7: The Structure of the Interface

in different media types, the text window will be first opened and the text parts of the content are displaied. Each element which is not text is represented by an icon and a text caption. Selecting the icon will cause the element to be displaied in a window which can display the media type of the element. The command shell windows are used to define the directory, register and delete objects in the service base, list registered objects and make service requests. The browser can be seen as a collection of classes. When media types increase, more windows can be added to the browser by simplely adding subclasses in class browser and creating its class instances. The browser in our system is implemented by using the X-window system in SUN workstations. An example of the browser is shown in Figure 8. As to the workstation without bitmap display, an interpreter which has the same function of the command shell windows in the browser is created to provided scroll display functions.

For applications written in common languages (Fortran, Mudula-2, etc.), a library is provided. The library provides a way for common languages to access the service bases through standard query language C++.

4.4 Application Examples

Some application examples are available in our experimental system. These examples include distributed computation facility, distributed multimedia document management facility and remote compilation facility etc.. In

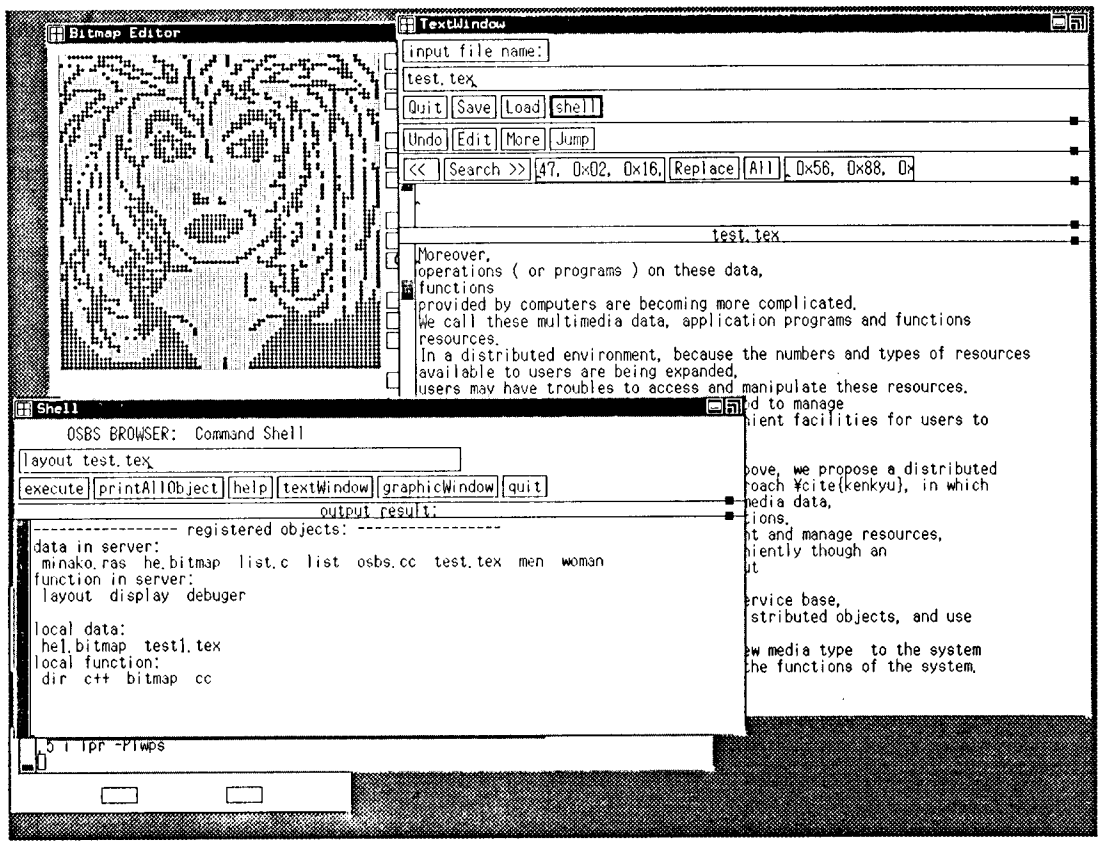

Figure 8: Windows in the Browser

the distributed computation facility, individual program components (objects) in an application can be distributed to the computing servers or workstations, providing both higher performance for workstation users and better uses of computing resources throughout the networks. In the follows, we give some introduces to two application examples.

The distributed document management facility: recently, in the area of office automation, there are few problems to create or display multimedia documents. Many workstations provide various facilities to support document creation. In an environment where a large number of documents (for example, over hundreds) are distributed and shared by users, the integrated management of these document is difficult. In our distributed document management facility, we suppose the documents exit in the local file system of each node. The documents can be documents in ODA format or in various other formats (for example, a text-only document in Latex or PostScript format, or an image-only document in Raster format). Information (news, messages and mails etc.) can also be referred to documents. Our experimental system provides facilities to creat, access and store multimedia documents. The shared documents are mainly stored in file servers in some clusters. The cluster servers in each cluster contain meta-information of these documents and accept requests from workstations to provide guidances to access the documents.

The remote compilation facility: when an user is in a workstation, while a much faster compiler runs at a powerful computing server, remote compilation is extremely effective. In a network environment, to compile local files in a remote machine, one may do the jobs as follows:

Copy the local files to the remote machine;
Log into the remote machine;
Activate the compiler;
Copy the executable results back to the local machine;
Log out the remote machine.

In all these jobs, the user must take care of the locations of the files and the compiler. In other words, the environment of remote process execution in the network is not transparent. In our system, these steps are done by the service bases. The user just needs to invoke a compiling service request and input the local file names, he does not need to know which compiler will compile for him and where the fast compiler is in. The local files and the control will be automatically migrated to

the remote machine where the compiler locates in. Service bases combine the remote copy function, remote log in function and remote command execution function to form the compilation service.

5 Conclusion

In this paper, we have presented the concept of object-oriented service bases and discussed how to construct a distributed system based on service bases to manage multimedia resources. Our system focuses on how to abstract the meta-information from resources and how to manage these meta-information. The object-oriented approach provides us a modeling method.

Recently, many distributed systems have been designed to manage and share distributed resources (for example, distributed operating systems and network operating systems [1], distributed database systems). There are also some systems which are designed specially for multimedia information or multimedia message management[9] [10]. Although space does not permit a detail comparison between our system and those systems, here we can still mention some characteristics of our system by comparing to those systems. First, the extensibility: new resources of different types can be added to each node independently. Our system provide an open-end resource management framework in a developing network environment. Second, the combining ability: existing resources can be combined to provide more services. Finally, easy construction: the existing operating systems and network facilities are utilized. Our system is not constructed from beginning. We consider that these characteristics are important in practical distributed systems.

Our system is expected to be effective in the environment where new resources are extended frequently and the types of resources are various, for example, in office automation environments or research laboratory environments in universities or organizations. Recently, many researches about multimedia database system have been done, but there are few researches in this area concerning distributed environments. We are considering to construct a distributed multimedia database system on the framework of service bases.

References

[1] A. S. Tanenbaum and R. V. Renesse, "Distributed Operating Systems", Computing Surveys, Vol.17, No.4, Dec. 1985.

[2] ISO/IEC, "Text and Office System – Office Document Architecture (ODA) and Interchange Formate", DIS 8613, 1988.

[3] W. Horak, "Office Document Architecture and Office Document Interchange Formats: Current Status of International Standardization", IEEE COMPUTER, October 1985.

[4] Q. He and H. Tanaka, "Modeling and Implementation of Service Base System for Local Area Networks", Conf. on Multimedia Communication and Distributed Processing, IPSJ, No.37-7, Okinawa, Japan, May 1988.

[5] B. Toby, "Issues in the Design of Object-Oriented Database Programming Languages", Proc. of ACM Conf. on OOPSLA, pp. 441-451, San Diego, California, Sept. 1987.

[6] D. Walsh, R. Lyon and G. Sager, "Overview of The SUN Network File System", In Proc. of The Usenix Winter Conf., pp.117-124, Dallas Texas, 1985.

[7] B. Stroustrup, "The C++ Programming Language", Addition-Wesley, 1986.

[8] M. Stonebraker, E. Wong, P. Kreps and G. Held, "The Design and Implementation of INGRES", ACM TODS, 1, 3, pp. 189 - 222, 1976.

[9] Robert H. Thomas, et al., "Diamond: A Multimedia Message System Built on a Distributed Architecture", IEEE COMPUTER, December 1985.

[10] Michael Caplinger, "An Information System Based on Distributed Objects", Proc. of ACM OOPSLA'87, October 1987.