Hierarchical Object Groups in Distributed Operating Systems

Kentaro Shimizu Mamoru Maekawa Jun Hamano
Department of Information Science,
Faculty of Science,
University of Tokyo

Abstract
This paper describes a new concept and mechanism, called an object group, which realizes hierarchical, integrated object processing and management. Object groups can be used as a software basis for constructing object-based distributed systems. They support hierarchical control of objects, group-oriented communication, group access control, group resource management, and some generic operations on groups, providing consistent and uniform interfaces. Applications include job control, parallel processing, servers, object pools, hierarchical resource management and so on. In this paper, we also present some design issues and discuss the implementation on a network-transparent global distributed system which is under development.

1. Introduction
This paper describes a new concept and mechanism, called an object group, which realizes hierarchical, integrated object processing and management. The object groups can be used in a unified way for constructing object-based distributed systems such as [1],[8],[9]. In an environment where many objects are distributed in the system, it is especially useful to be able to view some set of correlated objects as a logical entity, an object group, and perform operations on this entity.

In general, objects might be grouped according to the following relations among objects:
(a) objects that belong to the same owner
(b) objects that constitute the same object
(c) objects that can be referenced by the same access right
(d) objects that are managed under the same resource allocation rule
(e) objects that reside at the same location.
(f) objects of the same type
(g) objects whose contents are related to one another
(h) objects that implement the same function
(i) objects that are participants in the same activity
(j) objects that have the same parent
(k) objects that are created under the same rule

Such grouping of objects might be hierarchical. For example, objects of the same type can be further grouped by contents and locations. For the location of objects, there are several levels of addresses such as network domain, machine, hardware device, and so on.

Grouping of objects has been partially supported by conventional operating systems. For example, Unix provides the facility of process groups, which allows to send signals to all the member processes in a group. The Unix shell uses this facility to do job control, controlling the execution of a set of related programs such as a pipeline of filter programs. However, a Unix process group is limited to a single Unix host and its operations are limited to signaling. V Kernel [2] extends the facility of Unix process groups by making inter-node boundaries largely transparent, and in addition makes it possible to use a process group as an abstraction of processes; namely to use a group name as a representative name to designate an arbitrary member object. This facility is realized by extending the single-process, message sending mechanism for process groups, making group identification compatible with process identification and allowing a group identifier to be used (almost) anywhere that a process identifier can be used. Such group communication mechanisms have also been proposed in [3].

The access control over a group of objects is a natural extension of the concept of protection domain, which defines a set of objects that a subject can access. The access group of Sesame [7], which defines a subset of users to whom privileges may be awarded or from whom privileges may be revoked. As for the resource management, a fair share scheduler of Unix [4] uses a fair share group which defines a scope of resource allocation. However, it is limited to a single host and provides no mechanisms for resource management.

An object group is an extension or an expansion of the above grouping mechanisms. Furthermore, it has a wide range of objectives as follows:

(1) to send messages to all the members of the group at the same time
(2) to use a group name as a representative name to designate an arbitrary member object
(3) to realize a control mechanism to use a group as a pool of objects
to construct a hierarchical structure of objects

Among these items, some have already been realized in the conventional systems as described above. However, no system has incorporated such integrated mechanisms, providing consistent and uniform interfaces independent of the types of member objects in a distributed environment; thereby having wide varieties of applications.

Another kind of grouping is one of heterogeneous objects (objects of different types). The main objective of this kind of grouping is to define hierarchically organized objects. MIKE [9] realizes such objects by the concept of tasks. The task model provides encapsulation of structure and simplicity of interfacing. It is composed of one or more processes and possibly some objects such as files. The components of the task provide its address domain or share of control. The task also forms a protection domain to the outside world, which is realized by a process called guardian that performs the checking of access to objects. In MIKE, all components of a task must reside in the same node. HPC [5] allows the nested structure of objects. In HPC, an object can be constructed from component parts by combining other previously created objects, and an encapsulation shell. The encapsulation shell allows to encapsulate a set of communicating objects within the shell that provides appropriate interfaces, thus creating a recursive structure of black boxes to form complicated objects.

Our object model uses a similar approach for hierarchical definition of objects. In our object model, a user can define new types of active objects called compound objects. A compound object consist of one or more component objects and an activating process. A component object is either a primitive object and a compound object. A primitive object is directly managed by the system. Typical examples of this type of objects include files, devices, processes, etc. The activating process has control of component objects and performs the requested operations. The component objects may also be the objects defined by a user; thus, the nested structure of objects is realized. Such objects could also be viewed as object groups. In this paper, however, we distinguish object groups from these hierarchically-structured objects, defining object groups as those which consist of the same types of objects.

In this paper, we first describe our object model and then discuss the implementation of object groups on our network-transparent distributed operating system GALAXY [6] which is under development.

2. Object Model

We assume an object-based distributed system as a basis for implementing the object groups. In our model, hardware/software system components such as files, devices, processes, etc. are defined as objects. Conceptually, at the user level, objects can be viewed as active entities which are instances of abstract data types (classes) and act upon each other by means of invoking actions associated with individual entities; an action is invoked by sending an explicit message to another object.

There are two kinds of objects: primitive objects and compound objects as defined above. Every compound object consists of component objects and an activating process. The activating process has control of component objects and executes the requested operations. The appearance of activity of objects at the user level is an illusion created by the efforts of activating processes. In our model, objects can be defined recursively in that a component object may also be a compound object.

Every object is given a system-wide unique identifier (ID). It provides an unambiguous way of identifying any object in the system. Unique IDs are used by the kernel and user programs to refer to distinct objects.

Any class of object is managed by a special module dedicated to the class. We call such a module by a general term object manager. For example, process objects are managed by a process manager, file objects are managed by a file manager, and so on. Object managers are distributed among the nodes in the system. The locating of objects is performed by cooperative distributed object managers.

Objects are grouped into user-defined classes. For every class of object, there exists a class manager, which manages the creation and deletion of objects (instances) of that class. Each class manager has a registry of the objects belonging to its class, and a class entity which consists of definition of operations associated with the class and the inheritance information.

3. Basic Functions of Object Groups

An object group provides five basic functions: group-oriented communication, group access control, group resource management, generic group operations, and group hierarchies. These functions can be implemented efficiently by using the knowledge that the members of a group are of the same class.

1. Group-oriented communication

Many operations that can be applied to a single object can also be applied to an object group. For example, a multicast to members of a process group can be considered as a generalization of message sending to a single process. When an object group receives a message, the message is delivered to its members in various ways.

Many communication and synchronization schemes can be conceived about the group-oriented communication. Our object groups realize three different types of operations that seem to be necessary and appropriate for different practical applications: all-reliable, 1-reliable, and 0-reliable operations. We define a group operation to be k-reliable if either at least k members of the group are guaranteed to have had the operation performed on them or else a failure is notified.
to the sender [2]. All-reliable operations are useful to control a group of objects. One can suspend, resume or terminate all the processes that run in parallel. 1-reliable operations are especially useful for implementing servers (see Section 5.2). In this case, the sender may not know the addresses of all server processes and has only to know one process which performs the operation. 0-reliable operations are used to broadcast a message that is not so important to member objects; the message broadcast by this operation is lost if not received immediately by a receiving object. It has an analogy to a radio announcement in everyday life and may be useful for implementing a broadcast command such as 'wall' command of Unix, which sends a message to all logged in users.

From a practical point of view, these group communication schemes require different synchronization mechanisms. For example, it would be natural that in 1-reliable and 0-reliable operations, member objects need not block waiting for a request message. In this case it can be conceived that the recipient is signaled by such a mechanism as interrupts when messages are ready to be received. As for 1-reliable operation, an arbitrary object may also be selected automatically by an object group among those that are ready to receive messages. Figure 1 shows the message transactions of the above three types of operations.

(2) group access control

An object group may have an access control list associated with it, just like non-group objects. Access control is applicable to a group of both active objects (as subjects) and passive objects. This is a natural extension of protection domain.

(3) group resource management

The object group provides the mechanisms for allocating resources to and managing resources among the member objects of a group. Such mechanisms are implemented by exchanging information about the utilization of resources among the members of the group. It is achieved by the following steps:

1. At a certain interval of time, utilization of the specified resources for each member object is sent to a group managing process (group manager) of its object group.
2. The group manager stores it as a resource utilization information and delivers it to each member object when necessary.

An object manager executes resource allocation or scheduling by using the resource utilization information, for the object creation, expansion, and other requests requiring resources. The functions of resource allocation and scheduling themselves are provided by object managers; these functions are specific to a class of objects.

The kinds of resources may differ according to the classes of objects: processor loading and memory demand for process objects, use of disks for file objects, and so on. A workstation (WS) object is an object to collect and maintain the running state of the workstation. In an object group of WS objects, processor loading, memory demand, use of disks etc. are exchanged among the workstations. The resource management provided by object groups is very useful for resource allocation to a group of objects distributed in the system. For example, the fair share scheduler of Unix [4] could be implemented easily in a distributed

![Figure 1. Examples of group communication](image-url)
environment; a fair share group can be defined as an object group. Such scheduling can also be applied to the use of memory and disks.

(4) generic group operations

Generic group operations include creation and deletion of a group; union, intersection and set-difference of any two groups; join and remove of members. For example, an object group of any class can be created by the following sequence of operations:

\[
\text{GroupID} = \text{CreateGroup(ObjectClass)}
\]

JoinGroup(GroupID, ObjectID)

This simplifies both the code that uses the grouping facilities and the system interface.

(5) group hierarchies

Every object may belong to one or more object groups. Group hierarchies are defined based on the inclusive relations among groups. This facility guarantees that an object belonging to a group at the lower level is also to belong to a group at the upper level. Such hierarchy is only applicable to groups of the same class. The operation:

\[
\text{ImplyGroup(GroupID}_1, \text{GroupID}_2)
\]

indicates that any member object of GroupID1 is also a member of the group specified by GroupID2. Group hierarchies are useful for implementing hierarchical servers and hierarchical resource management as described in Section 5.

4. Implementation on the GALAXY Distributed Operating System

GALAXY [6] is a network-transparent, object-based distributed operating system. In this section, we discuss the implementation of object groups in GALAXY.

4.1. Object management in GALAXY

The GALAXY system is designed based on our object model. Objects can be defined as described in Section 2. Every GALAXY object is given a system-wide unique ID, which is used for identifying and locating the object. Unique IDs and all other information necessary to access objects (e.g. physical locations of replicas) are registered in a system-wide table, called ID table. Each node has a partial copy of the ID table which contains the IDs that are used by processes running on the node. Each entry of the ID table (IDTE, for short) corresponds to an object and may be duplicated in multiple nodes. All IDTEs of the same object are linked together so that any modification or revocation can be made through this link. This linking information is also described in the IDTEs. IDs are managed by an ID manager, which creates IDs and maintains the ID table. The ID manager resides on each node in the system.

Object managers are scattered among the nodes where the objects of this class exist; each manages a subset of the objects on the node cooperatively. When an operation invocation message is issued to an object, the corresponding object manager is invoked. It then invokes the ID manager to get the physical locations of replicas; selects an appropriate replica using some decision rule; and forwards the message to this replica.

4.2. Object groups in GALAXY

An object group is also an object of GALAXY and has a unique ID. The locating of an object group can be processed by the locating mechanism described in the previous section just like any other classes of objects. The IDTE of an object group has the group bit on, which represents an object group of the class specified in that IDTE. The group bit allows the system to efficiently distinguish between an object group and a non-group object.

The entity of an object group comprises:

(a) a list of its members
(b) information about group hierarchies (optional)
(c) resource utilization information (optional).

A group manager manages the object groups by using the above information; it executes group operations, exchanges the information about resource utilization, and maintains the membership and hierarchy of an object group. The membership information, (group ID, member object ID) pair is maintained by each object manager to make it possible to obtain the IDs of groups that the member objects belong to. Figure 2 shows this management scheme.

As is obvious from the above discussion, control is centralized in object groups. This is very significant because if a node that manages an object group goes down, the group facility becomes unavailable. For the sake of reliability, in a distributed environment, replication should be allowed for object groups. In GALAXY, the replication of an object

![Figure 2. Management of object groups](image-url)
group is realized by replicating a group manager and the entity of the group; each group manager has duplicate information for managing the object group. Figure 3 illustrates the replication of an object group. When the resource management facility is used, resource utilization of each member is informed to the group managers. Then, one of the group managers sends the collected information to all members.

5. Practical Applications

Many applications of object groups can be conceived. They include job control, parallel processing, servers, object pools, hierarchical resource management and so on. In this section, we describe three major applications and specific uses of object groups.

5.1. Parallel processing

A user may wish to suspend, resume and terminate a group of active objects (processes) running in parallel. Such operations can be facilitated by using an object group (process group), which provides a unified way for specifying an operation to a group of objects. Practical applications include parallel processing such as simulation and sensing in a distributed environment, pipeline processing of command execution, and duplicated processing.

One method of defining an object group (process group) is to specify it in the form:

```
GroupID = CreateGroup("process")
JoinGroup(GroupID, ProcessID).
```

Another method is to specify the child processes as a process group implicitly. For instance,

```
GroupID = CreateImplicitGroup("process")
```

5.2. Implementation of servers

A distributed system may have various servers providing file services, print services, etc. In a query to locate a servicing object, a client process may need to communicate with the group of servers [2]. In this case, the client may not specify explicitly the names or addresses of individual servers, and additional servers can be added and deleted dynamically without the need for modifying the client implementation. Using a group identifier as a destination of message sending,

```
ProcessID = SendMessage(GroupID, Message, Parameters)
```

blocks until at least one process has received the message and sent back a reply message (an example of 1-reliable operations). Servers may be implemented hierarchically as in [2].

If k-reliable operations would be allowed, we could send a message in the form:

```
ProcessIDs = SendMessage(GroupID, Message, Parameters)
```

which returns at most k process IDs. Such operations can be applied to implement a processor pool model. The k-reliable operation may be constructed by using 1-reliable operation.

5.3. Hierarchical resource management

All the workstations within the GALAXY system are divided into hierarchical groups (WS groups). Information exchanged among the workstations includes: (a) processor loading, (b) memory demand, (c) use of disks, (d) communication traffic between workstations, (e) running state of the workstation, (f) characteristics of the workstation such as processing power, memory capacity, disk capacity, the type of attached processor, etc. These kinds of information can be classified according to the time scale of variation as follows:

(1) level 1 (frequently changed) . . . (a) - (d)
(2) level 2 (sometimes changed) . . . (e)
(3) level 3 (rarely changed) . . . (f)

Corresponding to these levels, the scope of information exchange is determined; for example, the information of level 1 being exchanged within the innermost group and the information of level 2 within an outer group. Although the WS group is used as a software base for constructing distributed systems, it should be defined by taking into account of

Figure 3. Replication of an object group
the physical conditions such as communication speed, capacity, implementation (e.g. availability of hardware broadcast and multicast), monetary charge, etc. Also, simplified results obtained from the resource utilization information of a lower-level group may be exchanged within an upper-level group.

The resource utilization information is used for creation, migration, and replication of objects. Object managers ask WS managers to reference this information, in the depth-first order such as the current group, siblings’ groups, parents’ groups, and so on.

6. Design Issues

6.1. Global groups and local groups

Our object group is a logical entity; it is totally independent of physical locations and structures. For example, the member processes of process groups may communicate with each other by sharing memory or exchanging messages, and the same group operations can be used in both cases although some implementation mechanisms might be different. V Kernel process groups [2] allow a user to specify whether a group is local or global at the creation time. (A global group can have members on any node whereas the members of a local group must reside on the same node.) The primary intention to do this is to improve the efficiency of local groups, but this degrades the network transparency. In GALAXY, object groups are implemented on the network-transparent naming and locating mechanisms.

6.2. Replication and object groups

We have assumed that the replication of member objects is another level of operations. In such a system that supports replication by the basic locating mechanism, the grouping of objects can be made independent of the replication of member objects. However, the control of replication can also be realized by the mechanism of object groups itself; replicas are defined as members of an object group. In this case, the resource management mechanism would be useful for creating and migrating replicas. Object groups themselves can also be replicated in the way as described in Section 4.2.

6.3. Implementation of the group communication

Multicast or broadcast capabilities can be applied to group communications. Some local area networks (e.g. Ethernet) support these capabilities at the hardware level. In this case, group operations can be made more efficient than multiple single-object operations. Among the group facilities, the group resource management will be the most performance critical because it requires $O(mn)$ message exchanges during a certain period of time (n is the number of members and m is the number of replicas of the group). Since it can be implemented by n multicast operations, its performance will be much improved by using hardware-level multicast or broadcast capabilities.

At the software level, advanced interprocess communication (IPC) facilities can also be used for group operations. The object groups of GALAXY use IPC facility based on ports, which allows capability-based, many-to-many communication.

6.4. Programming interface

There are several approaches to specify a group operation as an extension of a single-object operation.

A simple approach is that the same system calls are made applicable to both object groups and single objects. For example,

```
SendMessage(ObjectID, Message, Parameters)
```
sends a message to an object, and the operation:

```
SendMessage(GroupID, Message, Parameters)
```
sends a message to a group of objects. This simplifies the system interface and the program code that uses the group facilities. However, additional techniques must be employed when some group-specific parameters are needed (e.g. communication and synchronization schemes for group-oriented communication). Moreover, it may complicate the system structure because the system cannot first check the group operations.

The second approach is the group-specific system calls. Some naming conventions can help the programming such as SendMessageGroup, CreateGroup, SuspendGroup, etc. The facility to declare group system calls dynamically will also be necessary.

The third approach is a common system call to execute group operations. For example,

```
GroupOP(Operation, GroupID, Parameters)
```
performs the Operation to members of the object group specified by GroupID. An ObjectClass parameter may be specified for the sake of efficiency. This operation is very flexible, but programming may be redundant. Such problem is solved by language translation techniques such as macro facilities.

6.5. Object naming

As for the grouping at the object naming level, the hierarchical naming is used widely and is proved to be sufficient for discriminating correlated objects from others and managing them as a group. In the hierarchical naming, each edge of the tree has a label, which denotes the relations between objects registered in each subtree; an object name consists of the labels along a path from the root of the tree to a leaf node. Although it is possible to name an object group just like a single object, the above hierarchical naming can be utilized to distinguish object groups from single objects. By using a context-relative naming, the local name space corresponding to an object group can be introduced.
7. Conclusions

The concept and mechanism of object groups have been discussed in this paper. We believe that they can be applied to many object-based distributed/nondistributed systems. The mechanism of object groups is currently being implemented on the GALAXY distributed operating system. GALAXY aims at the integration of information and coordination of resources scattered in a wide geographic area. We use the DDX packet exchange network and will use ISDN as a wide area network and are implementing the mechanism of object groups in such a widely distributed environment.

Acknowledgements

The authors wish to thank the members of the GALAXY project, G. S. Park, H. Ashihara, T. Sano, P. K. Sinha, X. Jia, N. Utsunomiya and N. Hirai, who have contributed to valuable and helpful discussions. They also give special thanks to P. K. Sinha, G. S. Park and X. Jia for reading and commenting on the work.