DESIGN AND IMPLEMENTATION OF A CORBA FAULT-TOLERANT OBJECT GROUP SERVICE

G. Morgan, S.K. Shrivastava, P.D. Ezhilchelvan and M.C. Little

Department of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU, England.

ABSTRACT

Many fault-tolerant distributed applications can be structured as one or more groups of objects that cooperate by multicasting invocations on member objects. The building of group based applications is considerably simplified if the members of a group can multicast reliably and have a mutually consistent view of the order in which events (such as invocations, host machine failures) have taken place. With this observation in mind, this paper describes the design and implementation of a CORBA middleware service for managing object groups. The object group service is portable and intended for a wide variety of applications; objects can simultaneously belong to many groups, group size could be large, and objects could be geographically widely separated. The service can provide causality preserving total order delivery to members of a group, ensuring that total order delivery is preserved even for multi-group objects. Both symmetric and asymmetric total order protocols are supported, permitting a member to use say symmetric version in one group and asymmetric version in another group simultaneously. The service is both dynamic and fault-tolerant: ordering and liveness is preserved even if membership changes occur due to (real or suspected) member failures, voluntary member departures and new group formations.

Key words: CORBA, fault tolerance, middleware service, object groups, atomic multicast, total order.

1 INTRODUCTION

Object Management Group's (OMG's) Common Object Request Broker Architecture (CORBA) specification provides an industry standard for building applications from distributed objects [1]; two of its main features are:

- *Object Request Broker (ORB)*, which enables objects to invoke operations on objects in a distributed, heterogeneous environment. This component is the core of the OMG reference model. Internet Inter-ORB-Protocol (IIOP) has been specified to enable ORBs from different vendors to communicate with each other over the Internet.

- *Common Object Services*, a collection of 'middleware' services that support functions for using and implementing objects. Such services are considered to be necessary for the construction of any distributed application. These include transactions, concurrency control, persistence, and many more.

Currently there is no OMG standard for an *object group* service. However, such a service would be highly desirable, as many fault-tolerant distributed applications can be structured as one or more groups of objects that cooperate by multicasting invocations on member objects. OMG is currently considering proposals for fault tolerance in CORBA that would require facilities for managing groups of objects [2]. The building of group based applications is considerably simplified if the members of a group can multicast reliably and have a mutually consistent view of the order in which events (such as invocations, host machine failures) have taken place. Design and development of fault-tolerant group communication protocols for distributed systems satisfying such properties has therefore been a very active area of research (e.g., [3-9]). More recently, the emphasis of research on fault tolerant group communication has shifted towards the provision of group communication as a CORBA middleware service [10-13]. This paper represents a research effort in the same spirit and describes the design and implementation of a CORBA middleware service for managing object groups.

Our CORBA object group service is intended for a wide variety of applications. This has been made possible because it has been implemented using a general purpose group communication protocol suite, Newtop [8], that contains several features not supported by other object group services. These and related aspects are discussed in section two; sections three and four then describe the Newtop object group service in detail; concluding remarks are presented in section five. The actual algorithms used in Newtop for ordered delivery and membership management are presented in [8], and will not be discussed here as they are not necessary for understanding this paper.

2 BACKGROUND, MOTIVATION AND RELATED WORK

2.1 INTRODUCTION TO GROUPS

A *group* is defined as a collection of distributed entities (objects, processes) in which a member entity can communicate with other members by multicasting to the full membership of the group. A desirable property is that a given multicast be *failure atomic*: if a process crashes while multicasting a message, either all or none of the functioning members deliver the message. An additional property of interest is guaranteeing *total order*: all the

functioning members are delivered messages in identical order. As an example, these properties are ideal for replicated data management: each member manages a copy of data, and given atomic delivery and total order, it can be ensured that copies of data do not diverge. However, as we discuss below, achieving these properties in the presence of process and network failures is not simple.

We assume that group members could be geographically widely separated, say communicating over the Internet. We therefore model the communication environment as asynchronous, where message transmission times cannot be accurately estimated, and the underlying network may well get partitioned, preventing functioning members from communicating with each other.

Suppose that a multicast is interrupted due to the crash of the member making the multicast; this can result in some members not receiving the message. Member crashes should ideally be handled by a fault tolerant protocol in the following manner: when a member does crash, all functioning members must promptly observe that crash event and agree on the order of that event relative to other events in the system. In an asynchronous environment this is impossible to achieve: when members are prone to failures, it is impossible to guarantee that all functioning members will reach agreement in finite time [14]. This impossibility stems from the inability of a process to distinguish slow members from crashed or disconnected ones. One way to circumvent this impossibility result is to permit processes to *suspect* [15] process crashes (sometimes incorrectly) and to reach agreement only among those processes which do not suspect each other [5] [6]. This leads to a *partitionable membership* service which ensures that the functioning members that do not suspect each other install an identical sequence of membership views, with each view installation being identically synchronised with respect to message delivery events.

2.2 MOTIVATION

Newtop object group service has been designed to support a wide variety of group interactions, some of which will be illustrated in this section. Objects can simultaneously belong to many groups, group size could be large, and objects could be geographically widely separated. Newtop can provide causality preserving atomic, total order message delivery to members of a group, ensuring that total order delivery is preserved even for multi-group objects. Both symmetric and asymmetric order protocols (see below) are supported, permitting a member to use say symmetric version in one group and asymmetric version in another group. Newtop is both dynamic and fault-tolerant: ordering and liveness is preserved even if membership changes occur due to (real or suspected) member failures, voluntary member departures and new group formations. We do not know of any CORBA object group service that simultaneously supports all of these features.

We present a few examples of group based applications to justify the features provided by our service.

- *Replication*: A service may be replicated over a number of nodes to increase its availability. The replicated service may be consigned to a single group with all the required replica group services - such as replacing failed members and state transfer- confined within the server group. Clients may participate with the replica group by forming a group consisting of the replica group and itself (a *client/server* group). When a client no longer requires a service, the client/server group may be disbanded. Fig.. 1 (a) shows a client/server group. Client X forms the client/server group g1 to enable participation with server group g2 with three replicas P1, P2, P3. An alternative method suited to situations where it is impractical for clients to participate in group communications is shown in fig. 1 (b). In this method client invocations are directed at a single member of a group. A group member receiving invocations from a client in this manner is then responsible for redistributing the client invocation to the other group members.

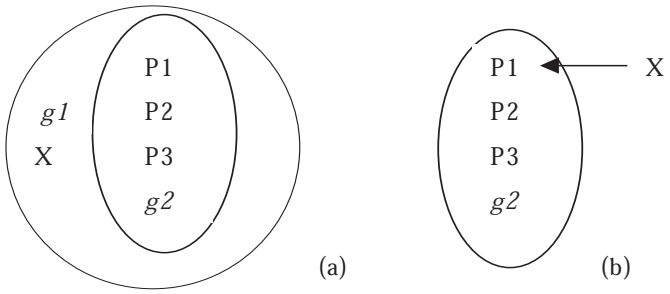

Fig. 1: Client-server group interactions

- *Online server migration*: Assume that it is necessary to migrate a member of a replicated server group to some other machine. The task is complicated (in this example) because each server replica maintains a substantial amount of state (say several megabytes of data), but it is required that the migration process must not cause any noticeable disruption in service or compromise availability. A possible solution will work as follows. Assume group g_1 (fig. 2(a)) to be the server group, and P_2 is to be migrated. A server process P_3 is created at the intended location. This process initiates the formation of a new group, g_2, containing P_1, P_2 and itself (fig. 2 (b)). Within g_2, P_1 and P_2 use some specific protocol for updating the state of P_3; at the same time, P_1 and P_2 remain responsive to clients by servicing requests directed to g_1; eventually P_1 departs from g_1, and P_2 departs from both g_1 and g_2, leaving g_2 to be the surviving group with P_1 and P_3. This specific solution also suggests the possibility of using multiple groups for developing a general approach for performing online software upgrades in a system (e.g., replace component P_2 by P_3).

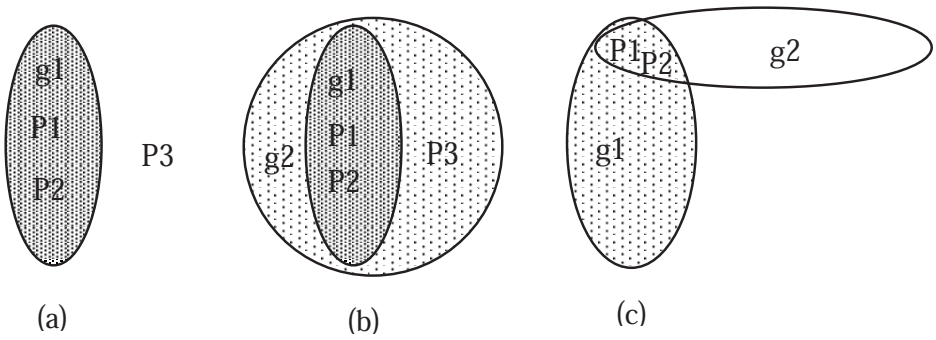

Fig. 2: Multiple groups

- *Computer supported co-operative work (CSCW)*: We consider two online conference groups g_1 and g_2, where some members (P_1 and P_2, fig. 2(c)) are common to both. We require the functionality

that the members of each group are delivered messages in causality preserving total order; this total order delivery must also be preserved when members that belong to multiple groups are delivered messages of different destination groups in an identical order (this latter requirement makes it harder to deal with overlapping groups). Imagine that g_1 represents a moderated discussion group, so members direct the messages to the moderator, who then distributes the message to all the members. It is natural to use a sequencer based (*asymmetric*) total order protocol for g_1, with the moderator acting as the sequencer. Members of g_2 on the other hand are collaborating via a shared whiteboard; here all the members have equal status. Within g_2 then, it seems natural to employ a *symmetric* total order protocol that does not require a sequencer. There could be other system specific reasons (e.g., network topology, frequency of multicasts etc.) for preferring a symmetric version over an asymmetric one and vice versa [16]. Thus, in addition to being capable of belonging to multiple groups, a member should be capable of using an asymmetric protocol in one group and a symmetric one in another group.

Newtop object group service supports all of the above forms of group interactions.

2.3 RELATED WORK

Three ways of incorporating object groups in CORBA have been identified [12,13]. The *integration* approach takes an existing group communication system and replaces the transport service of the ORB by the group service [10]. Although this is a very efficient way of incorporating group functionality in an ORB, the main disadvantage is that this approach is not CORBA compliant, lacking in interoperability. The second approach called the *interceptor* approach also makes use of an existing group communication system; here (IIOP) messages issued by an ORB are intercepted and mapped on to calls of the group communication system. The best known example of this approach is the Eternal system [11] for object replication that makes use of Totem group communication system [4] in this manner. The major advantage of this approach is that no modifications to the ORB are required. The shortcomings are that the approach is only possible if the host operating system permits interception (Unix in the case of Eternal); secondly, as the group communication system is not available to CORBA application builders, it can only be used in a fixed manner (replication in the case of Eternal). The third approach is the *service* approach: it does not make use of any existing group communication system; rather the group communication system is implemented as a CORBA service from scratch. In addition to being CORBA compliant, the advantage here is that the service is available to application builders so can be used for a variety of purposes. This approach was first developed in the Object Group Service (OGS) [12,13], and has been taken in the Newtop service. We briefly compare and contrast our service with OGS.

The Newtop service offers a more comprehensive set of group management facilities than OGS. In terms of functionality, OGS essentially supports the type of group interaction depicted in fig. 1(b) and does not support objects belonging to multiple groups like Newtop. OGS supports only the asymmetric way of total ordering, whereas our system supports the symmetric way as well. Newtop and OGS take differing approaches to the handling of failures. Newtop permits a group to be partitioned into connected subgroups and guarantees that message delivery in each subgroup is totally ordered; whereas, OGS attempts to preserve a unique (majority) subgroup; so, when failures increase beyond a threshold, OGS blocks message delivery until it becomes possible to form the unique subgroup again.

3 NEWTOP OBJECT GROUP SERVICE

3.1 OVERVIEW

An application developer creates potential group members as CORBA objects. Such objects are defined by an IDL interface. A single group member is addressable via a single object reference (IOR). This enables the Newtop service to identify group members and associate them to groups. A group member will also be referred to as a *client* of the Newtop service.

The Newtop service is a distributed service and achieves distribution with the aid of the *Newtop service object* (NSO). Each client (group member) is allocated an NSO. Group related communication required by a client is handled by its NSO. Only one NSO is required by a client, irrespective of how many groups the client participates in. Communication between a client and its NSO is handled by the ORB. Therefore, the NSO may reside within the same address space, in a different address space, or on a different node in the network to the group member associated with it. The most efficient configuration would be the client and its NSO within the same address space. Fig. 3 shows the communication relationships between Newtop service clients and their NSOs.

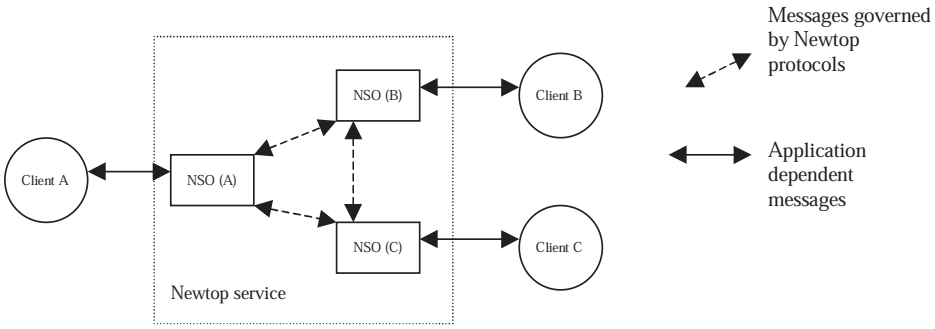

Fig. 3: Clients of the Newtop service and associated NSOs.

The *Newtop service object factory* (NSOF) supports the function of creating NSOs and associating them to potential group members.

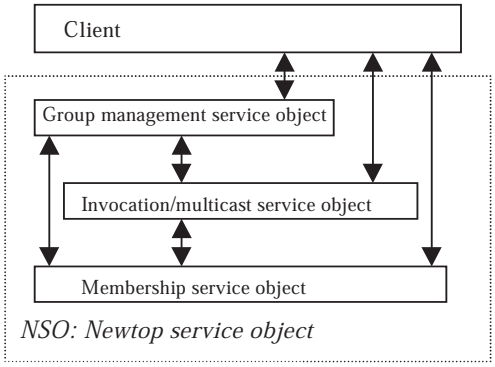

Fig. 4: Newtop services

The Newtop service itself consists of three services implemented by corresponding objects within the NSOs: (i) membership; (ii) invocation/multicast; and (iii) group management (see fig. 4). The management service provides clients with create, delete and leave group operations. The invocation/multicast service provides three group invocation operations (wait for responses from all, from one and an asynchronous, no wait invocation). The membership service maintains the membership information and ensures that this information is mutually consistent at each member. This is achieved with the help of a failure suspector that initiates membership agreement as soon as a member is suspected to have failed. The client can obtain the current membership information by invoking 'groupDetails' operation. Fig. 5 summaries the main operations provided by an NSO.

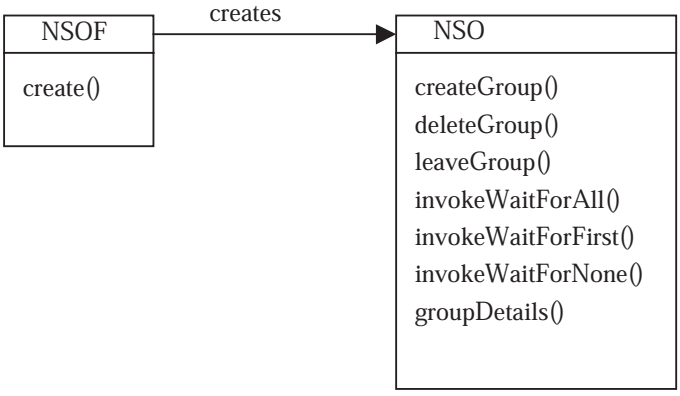

Fig. 5: Summary of NSO operations

We now describe each service in more depth.

3.2 MANAGEMENT SERVICE

The management service manages the creation of new groups, the deletion of existing groups, and the change of membership for existing groups.

- *Creating a group* – The creation of a group is initiated by a potential client of the Newtop service; it is assumed that the relevant NSOs have already been created. The client is required to give the group an identifier that can aid the client and the Newtop service in differentiating between groups. This identifier should be unique and consists of a string of ASCII characters. The client is also required to supply an initial member list containing the IORs of the NSOs. The objects identified by the list are considered group members at the start of a group's life.

- *Deleting a group* – A client may specify, at any time, that a group is to be deleted. The deletion of a group does not result in the deletion of the individual group members, but only of the abstract group entity. When a group has been marked for deletion all group members are told by the management service to voluntarily leave the group. The group membership service may, due to members leaving and/or members failing, indicate to the management service that the membership of a group has reduced to a singleton. This results in the management service deleting this group.

- *Leaving a group* - At anytime during the lifetime of a group a member may request to leave the group.

A note on joining a group. Since members are permitted to belong to several groups, it turns out that there is no need for supporting an explicit facility for joining a group, as a similar effect can be obtained by members forming a new group and exiting the previous ones (as illustrated by the online migration example, figs. 2(a, b)). Joining a group in this manner results in the name of the group changing after each join is accomplished. This may at first appear to be a drawback. However, a member join usually requires some application specific processing (e.g., state transfer as described in the online migration example), which is very easy to support if the Newtop way of group joining is employed.

3.3 INVOCATION/MULTICAST SERVICE

The Newtop service relies on the message passing capabilities of the ORB for enabling multicast communication between group members. Since at present ORBs only provide one to one synchronous communication, multicasting has been implemented by making invocations in turn to all the members. Multiple threads of execution are used to obtain parallelism and prevent client blocking. It is expected that in the near future asynchronous messaging service will be available on ORBs; we can easily exploit this service to gain efficiency. Three types of invocations are currently supported by the NewTOP service: two synchronous invocations (wait for responses from all the members, wait for a response from any one) and an asynchronous, wait for no response invocation.

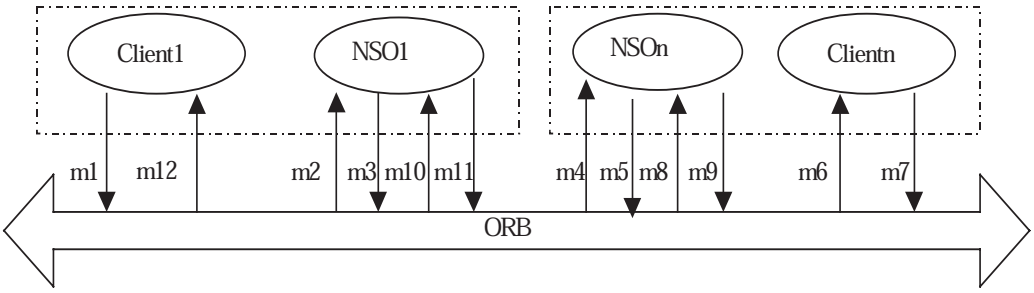

Fig. 6: Message interactions in a group multicast

We describe now the principal message exchanges involved in making a group invocation. Assume a group on n identical objects and object1 wants to make a synchronous group invocation on some operation of the objects; this invocation will have to be made via the group service. Fig. 6 shows two of these objects and their respective NSOs. The client of the Newtop service making the invocation is required to marshall the invocation request, consisting of the name of the function and associated parameter list, into a single structure and send it to its NSO. Message 1 (m1 for short) is such a message; m2 is its reception. As a result, NSO1 sends Newtop specific messages to other NSOs; in fig. 6, m3 is such a message and m4 is its reception at NSOn. NSOn responds by composing and sending the appropriate invocation message, m5, to its target object (objectn); m6 is its reception at objectn. The response from objectn (m7) is received by NSOn (m8); NSOn then sends Newtop specific message (m9), it is received at NSO1 (m10), from here m11 and m12 indicate the final journey back to the invoker. An NSO (such as NSOn) that is receiving an invocation on behalf of its target object must be able to compose the type specific invocation on the fly; this is made possible by making use of the Dynamic Invocation Interface (DII) feature of the ORB (in the fig., the invocation represented by the message pair m5, m8 uses DII).

Clients of the Newtop service are aware of the difference between a request issued to a group and a request issued to a single object. Making a group request appear the same as a singleton request is actually straightforward, and involves inserting a proxy object between the client and its NSO and letting the proxy object do the marshalling.

Message delivery is atomic with three types of ordering guarantees (causal, causality preserving total and arbitrary) and in the case of total order, two types of protocols are supported for enforcing the ordering.

- *Arbitrary ordering* - This type of ordering results in the speediest message delivery. Messages are delivered to group members as they are received.

- *Causal ordering* – Messages are delivered to group members in accordance with the causal dependencies that exist between messages [17].

- *Total ordering* – Members of a group deliver messages in the same order; causal relationships are preserved.

Two types of ordering techniques, *symmetric* and *asymmetric*, are supported. In the asymmetric version, one of the members of the group assumes the responsibility for the ordering of messages within the group. Such a member is commonly termed a *sequencer*. Electing a new sequencer, in case the original one departs from the group, is straightforward as the underlying membership service maintains consistent group views; so any deterministic algorithm can be used. In the symmetric version, all the members use a derterministic algorithm for message ordering. Experimental work has shown that symmetric protocols tend to be more attractive in situations where all

the members are lively, and multicasting regurarly (e.g, a conferencing application) whereas asymmetric protocols are better in other situations [16]. Our performance figures presented in the next section confirm this behaviour. The client of the Newtop service specifies the ordering properties and the type of protocol when creating a group. The created group will then manage message passing and ordering guarantees using the specified protocols.

3.4 GROUP MEMBERSHIP SERVICE

The group membership service maintains a mutually consistent view of a group membership for each member of a group.

- *Detecting member failure* – Like other group communication systems, the Newtop service may suspect member failures with the aid of a timeout based failure suspicion protocol. Suspecting a member of failure results in the execution of the membership agreement protocol; the suspected member will be removed from the group or will remain in the group with all suspicions removed. Whatever the outcome of the protocol, group members will retain mutually consistent views of the group membership.

- *Single group membership* – When membership of a group falls to singleton the group is marked for deletion, the management service is informed and all information relating to the group is removed.

In a group communication system a member is often required to stay *lively* within a group to avoid being suspected by other members. This usually takes the form of a member periodically sending “I am alive” or “NULL” messages during periods it has no application level messages to send. In Newtop, after a member has negleted to send a message for a period of time, the Newtop *time-silence* mechanism will send a “I am alive” message. A client of the Newtop service creating a group may decide if the group is to be *lively* or *event* driven:

- *Lively* – time-silence mechanism and failure suspicion is active throughout the lifetime of a group; the duration of the time-silence period is specified at the creation time.

- *Event* – The time-silence mechanism is only active when application dependent messages exist within the Newtop service environment. Once all these messages are delivered to group members the failure suspicion and time-silence mechanisms are shutdown. The appearance of further application dependent messages wakes up these mechanisms.

4 PERFORMANCE

Performance figures were obtained using the C++ version of the Newtop service, compiled with omniORB2 [18]. Group members reside in the same address space as their NSOs. Each group member is hosted on a different node in the network. The system consisted of 8 Pentium II PCs running Red Hat Linux 2.0.34, each with 64 megabytes of RAM, connected together using 100 Mbit fast Ethernet. The marshalled message used in all experiments was a CORBA string consisting of 100 characters.

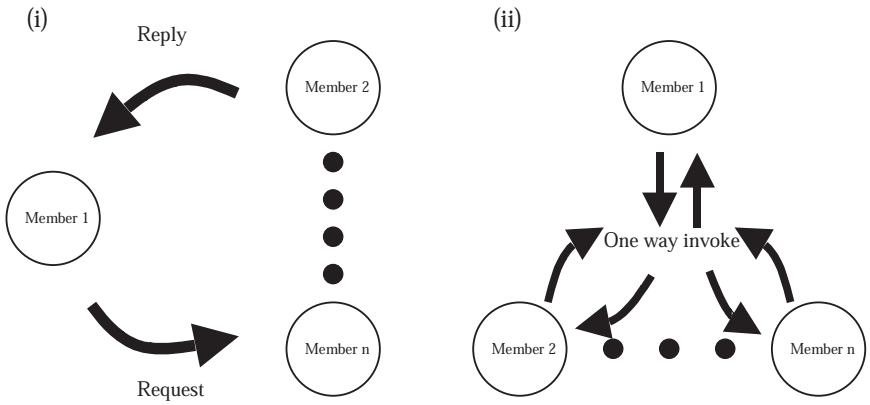

fig. 7: Groups for performance evaluation

Two experiments were carried out. In the first one (fig. 7(i)), member 1 multicasts a request to the group and waits for replies; this represents a commonly occurring scenario when services are replicated. The group was configured to be event driven. The asymmetric protocol is expected to perform better here. The time interval between the client issuing the request and receiving replies from all the other members was measured in milliseconds and shown in table I for various group sizes, for both types of the order protocol. The cost of making a unicast invocation in omniORB2 is 0.77 msec, whereas the same unicast call made via the NewTOP group service doubles the cost. For group invocations to n>1 members, we measured the cost of making n such calls directly using omniORB2; these figures appear under the column 'unreliable invocations'. The sequencer based scheme clearly outperforms its symmetric counterpart. The cost of obtaining atomicity and ordering properties are high, but the figures for small groups are still around 10 msecs which would be acceptable in a number of applications.

The second experiment (fig. 7(ii)) involves a group where all the members are regularly multicasting by using the asynchronous method invocation operation (as in say, a teleconferencing application). The group was configured to be lively; here the symmetric protocol is expected to perform better. The time interval between a member sending a message to that message becoming deliverable at all other members was measured; table II gives the figures. We see that the symmetric protocol is superior.

Table I (units of measurement = milliseconds)

Number of members	Asymmetric	Symmetric	Unreliable invocations
2	5.400	5.400	1.100
3	8.200	13.230	1.490
4	12.000	28.910	1.800
5	17.900	48.500	2.300
6	25.500	78.800	2.700
7	33.200	120.170	3.100
8	42.900	173.400	3.500

Table II (units of measurement = milliseconds)

Number of members	Asymmetric	Symmetric
2	5.300	4.100
3	8.900	7.500
4	11.900	9.500
5	21.600	15.000
6	31.200	23.000
7	44.200	35.000
8	66.900	43.000

5 Concluding Remarks

The Newtop object group service is intended for a wide variety of group based applications; objects can simultaneously belong to many groups, group size could be large, and objects could be geographically widely separated. The service implementation is portable; currently it runs on Orbix and OmniORB2. For future work we intend to build a number of group based fault tolerant applications; one of our aims is to use the service for supporting replication of transactional objects for high availability and evaluate against schemes that do not use groups [19].

6 Acknowledgements

G. Morgan was supported by EPSRC CASE PhD studentship with industrial sponsorship from HP Laboratories, Bristol.

References

[1] R. Orfali, D. Harkey and J. Edwards, “The essential distributed objects”, John Wiley and Sons Ltd., 1996.
[2] http://www.omg.org/techprocess/meetings/schedule/Fault_Tolerance_RFP.htm
[3] Birman, K., Schiper, A. and Stephenson, P.,"Lightweight Causal and Atomic Group Multicast", ACM Transactions On Computer Systems, Vol. 9, No. 3, August 1991, pp. 272-314.
[4] L.E. Moser, P.M. Melliar-Smith et al, “Totem: a Fault-tolerant multicast group communication system”, CACM, 39 (4), April 1996, pp. 54-63.
[5] Amir, Y., et al, “Transis: A Communication Sub-system for High Availability”, Digest of Papers, FTCS-22, Boston, July 1992, pp. 76-84.
[6] D. Dolev and D. Malki, “The Transis approach to high availability cluster communication”, CACM, 39 (4), April 1996, pp. 64-70.
[7] Chang, J. and Maxemchuk, N. F., "Reliable Broadcast Protocols", ACM Transactions on Computer Systems, Vol. 2, No. 3, August 1984, pp. 251-273.
[8] P. Ezhilchelvan, R. Macedo and S. K. Shrivastava, “Newtop: a fault-tolerant group communication protocol”, 15th IEEE Intl. Conf. on Distributed Computing Systems, Vancouver, May 1995, pp. 296-306.
[9] C. T. Karamanolis, "Configurable Highly Available Distributed Services", PhD thesis, Imperial College of Science, Technology and Medicine, June 1996
[10] S. Maffeis, "Run-time support for object-oriented distributed programming", PhD thesis, University of Zurich, February 1995.
[11] P. Narasimhan, L, E. Moser and P. M. Melliar-Smith, "Replica consistency of CORBA objects in partitionable distributed systems", Distributed Systems Eng., 4, 1997, pp. 139-150.
[12] P. Felber, R. Guerraoui and A. Schiper, "The implementation of a CORBA object group service", Theory and Practice of Object Systems, 4(2), 1998, pp. 93-105.
[13] P. Felber, "The CORBA Object Group Service: a Service Approach to Object Groups in CORBA", PhD thesis, Ecole Polytechnique Federale de Lausanne, 1998.
[14] Fischer, M., Lynch N., and Paterson, M., "Impossibility of Distributed Consensus with One Faulty Process", J. ACM, 32, April 1985, pp 374-382.
[15] T.D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems", J. ACM, 43(2), 1996, pp. 225-267. (prelim. Version in 10th ACM PODC, August 1991, pp. 325-340)
[16] L. Rodriguez, H. Fonseca and P. Verissimo, "Totally ordered multicasts in large scale systems", 16th IEEE Intl. Conf. on Distributed Computing Systems, Hong Kong, May 1996, pp. 503-510.
[17] L. Lamport, "Time, clocks, and ordering of events in a distributed system", Commun. of ACM, 21, 7, July 1978, pp. 558-565.
[18] http://www.orl.co.uk/omniORB/omniORB.html
[19] M.C. Little and S K Shrivastava, “Understanding the Role of Atomic Transactions and Group Communications in Implementing Persistent Replicated Objects”, POS98, Eighth International Workshop on Persistent Object Systems: Design, Implementation and Use, Tiburon, CA, September 1998.