Groups Partitioning over CORBA for Cooperative Work

E. Garcia, H. Guyennet and J-C. Lapayre
[garcia,guyennet,lapayre]*@univ-fcomte.fr*

Laboratoire d'Informatique de l'université de Franche-Comté
16, Route de Gray
25030 BESANCON cedex - FRANCE

Abstract

Many examples illustrate the usefulness of group partitioning in distributed systems. The process group abstraction is a powerful tool for the development of fault-tolerant distributed applications. This solution is also particularly relevant to group entities sharing similar properties or stakes when applied to the field of cooperative work. When dealing with distributed applications, communication standards have evolved from socket-based interfaces to Remote Procedure Calls and nowadays, towards distributed object platforms like CORBA. This standard from the Object Management Group masks distribution and heterogeneity and provides a good basis for distributed application development. This paper presents a review of group mechanisms found in the literature then focuses on the implementation of groups over CORBA. An implementation of such a service over CORBA illustrates the impact of group auto-organization. Performance we obtain show that our service is efficient and provides good mechanisms to manage operations in a group.

Keywords: group service, CORBA, cooperative work, auto-organization, performance

1 Introduction

People get together in groups to share common interests or to work on collaborative projects. Similarly, groups can be used in distributed systems to facilitate communication, to solve heterogeneity problems, to improve security and performance. Many examples illustrate the usefulness of group partitioning in distributed systems. The process group abstraction is a powerful tool for the development of fault-tolerant distributed applications. It can also be exploited to adapt the transmission rate for receivers by multicasting the video simultaneously to different groups of recipients having different QoS capabilities. Receivers can dynamically switch from group to group, depending on the available bandwidth [Chockler96]. When load balancing of processes between processors is enabled, the partitioning into groups [Guyennet00, Evans94] allows communication costs to decrease and solves problems of heterogeneity. In wide area networks, grouping nodes that are within a company or a region eases communication and allows scalability for meta-computing like in the Globus [Foster01], BetaGrid, EuroGrid and DataGrid projects. Many collaborative multimedia applications require support to address multiple levels of quality of service. One can imagine configurations in which subsets of participants would maintain their own private chat groups or side-band conferencing sessions like in the Maestro project [Birman97]. This kind of organization naturally follows the way people form groups in a common project. Communication delays are alleviated whilst security, privacy and resilience are improved.

The CORBA middleware we use, achieved standardization of the communication layers and handles heterogeneity problems either at the language, system or hardware level. However, it does not manage object groups. Having a group service on such a standardized layer would greatly facilitate the building of cooperative applications. This paper is organized as follows: The first part of this article reviews the literature about group partitioning in distributed systems. Then, we will present group communication protocols and will consider the dynamicity of the group structure. In section 5, we will focus on group services developed over CORBA. Section 6 will present an implementation of a group service over CORBA allowing group auto-organization. Different tests are performed on this CORBA service and are presented in this section.

2 Groups in Distributed Systems

The group concept allows us to define a group of entities as one virtual entity, thus making it possible to give only one and the same name to each member of a particular group, and to communicate with them using only one address. In wide area distributed systems, acquiring information, moving objects or processes, and making a decision are difficult problems, which cannot be solved for all the nodes. Therefore, in order to deal with these problems, subsets can be formed called *groups* [Birman91, Balayer95], *domains* [Raymond96], *clusters* [Zhou92], *partitions* [Rosti94], and *territories* [Raverdy96]. These groups are made up of members. In each group, one member plays a particular role. It is referred to as a *leader* [Balayer95], *manager* [Evans94], *Interlevel Contact Point* [Vaughan95], or *coordinator* [Birman98], and is responsible for managing the communication between groups or levels, receiving information and sending it to other members, and supervising internal group organization. The notion of hierarchy is very important in this concept since it allows us to order the various elements of a group. Scalability problems are replaced by a different problem related to the choice of elements making up the group. With protocols of group self-organization, the group structure can adapt to the dynamics of the system. When a group is too large, these protocols allow it to be divided into several smaller groups. Moreover, when group members have the same location, they can be better distributed in order to improve resilience.

2.1 Process Groups

The notion of group makes a set of processes appear as a single abstraction. A group is a number of processes that cooperate in a manner specified by the system or the user. A message can then be sent to the group without the address of the members being known. Furthermore, these can be members of several groups. They can be dynamic. New groups can be created while others can be destroyed.

Open groups exist in which each member can receive messages from outside. On the contrary, in *closed groups*, the members can only receive messages from another member belonging to the same group. Groups are composed of equal members and decisions are collective. Other groups are structured hierarchically. These two types apply to specific situations dependent on the application. The groups must possess their own address in order to receive messages. Either all members of a group can receive the message or none of them can. This characteristic of all or nothing reception is called *atomic broadcasting* and it is useful for distributed systems. The sender does not have to concern himself with the reception problems of the different members of the group. However, at the implementation level, each process must send an acknowledgement as confirmation. This is especially useful in the case of fault tolerance. Atomic property must be respected. A second property, global sequencing, ensures that the order of reception is the same as that of transmission.

A group server is used to manage these groups and it can be either centralized or distributed. Complex problems may occur, especially when a member suddenly breaks down or when a message is sent when a member withdraws or when a large number of members are in difficulty.

In AMOEBA [Tannenbaum92], a group is composed of one or several processes that work together to carry out a task to provide a service. The groups are closed. Users can access a member of the group via RPC. This member will then use the communication in the group to broadcast the message. Communication primitives are used to create a new group, enter or exit an existing group, send, receive and repeat in the case of failure.

In the Isis system [Birman91, Birman93], a distributed programming environment is presented, based on virtual synchronous process groups and communication groups. Isis takes fault tolerance into account to achieve reliable broadcasting. From experience with Isis users, four group structures appear in Isis programs: the peer group, the client-server group, the diffusion and the hierarchical group.

RM-ODP [Raymond96] deals with three organizational concepts referred to as groups. To characterize sets of objects, this standard talks about groups, configurations and domains. The group is a set of objects with a particular characterizing relationship. This relationship can be a structural relationship or an expected common behavior of the objects. For example, an addressed group where all the objects are addressed in the same way. The configuration is a more specialized term used to talk about a collection of objects able to interact at interfaces. The domain is also a set of objects but, compared to groups, each of these objects is related by a characterizing relationship to a controlling object.

In PVM [Geist94] and MPI [Bangalore94], the process groups can be used to specify those that are involved in global communication or to introduce the notion of task parallelism in an application. Dynamic process groups are installed over PVM. In this installation, a process can belong to multiple groups and the groups can change dynamically at any moment during execution. Functions, which logically use groups of tasks such as broadcast and barrier, use the names of groups explicitly defined by the user as arguments. Routines are provided to join or quit a designated group.

2.2 Groups of Processors

Cooperation also develops within a group of processors. These groups are created to improve performance in distributed systems. Hierarchical processor organizations have been presented for both multiprocessing and distributed configurations. Ahmad and Ghafoor [Ahmad91] give a two-level organization of a multiprocessor system for load balancing algorithms.

In Utopia [Zhou92], machines belonging to the same group share the same resources. The choice of grouping occurs according to the topology of the network (physical groups of machines) or by creating virtual groups of machines (in order to take into account constraints other than localization). The load of all machines is centralized around one server per group. The load can therefore be distributed over the network to the other groups. Centralized algorithms are used among the groups to make Utopia migratable to thousands of machines.

The notion of geographical grouping has also been studied [Evans94, Guyennet00]. All the machines within the same geographical area belong to the same group. The load distribution algorithm considers that a machine is labeled heavily or lightly loaded on the basis of a load threshold. Authors define an intra-group load distribution strategy that consists of a local distribution of the load, i.e. between the machines within a group. A second inter-group load distribution strategy takes into account the load distribution within a given group, and extends the action of distribution throughout the entire network.

Billard [Billard93] is interested in decision-making agents that have access to resources in a distributed system and whose goal is to improve performance in some application domain. For example, job schedulers can share a set of processors where some processors may be lightly loaded while others are heavily loaded. The goal is to obtain systems where some group formations are more advantageous than others, where agents have to search for the optimal group formation and where communication delays can affect the results of the search. If communication is too restricting, the agents will work alone, or they will form the most optimal group.

Vaughan [Vaughan95] organizes the processors in a virtual group structure so that individual processors are grouped into neighborhoods for information gathering. A hierarchy of rings is formed, at the summit of which is one ring which joins the system together and enables overall cooperation. These rings can be configured dynamically and patched or reconfigured in case of processor or link failure and repair.

Raverdy [Raverdy96] proposes a generic model for application oriented resource management based on dynamic and hierarchic partitioning. The global network is divided into small-scale entities: Computation Domains (CD). Within a CD, application processes execute inside Execution Territories (ET) and resources belong to Resource Territories (RT). Territories (ET and RT) cooperate inside a CD to provide needed resources to applications. In this model, resource management is separated from process management. This model is well adapted to the execution of heterogeneous applications in the same environment.

2.3 Group Communication

At network level, the IP protocol was extended to integrate the notion of group and it also specifies the adaptations required for the interface to manage groups [Benford93]. The extension of the protocol must enable the transmission of a packet to a group in an unreliable way. This operation is called IP *multicasting*. The groups are managed by broadcasting agents, *multicast routers*. A host transmits an IP multicasting packet to a group by encapsulating it in a selective frame on the network. IPv6 includes multicast for routers and workstations. An IPv6 multicast address is an identifier for a group of nodes. A node may belong to any number of multicast groups. The IGMP (Internet Group Management Protocol) protocol informs the neighboring multicast routers of the host's membership to the group. XTP (Xpress Transfer Protocol) is a combined network and transport level protocol that offers significant support for multicast transfers. In contrast to *IP Multicast*, XTP does require explicit handshaking between the sender and receivers that wish to join an existing group [Braudes93].

The V system exploits a group communication service to minimize communication costs. It was the first to utilize hardware multicast to implement process group communication. The Newtop protocol [Macedo94] replaces the context graph by the notion of causal blocks. Each causal block includes a set of messages that are causally independent. The Horus project [Renesse95] provides unreliable or reliable FIFO, causal or total multicast services. Horus is extensively layered and highly configurable. The Transis project [Dolev96] uses group communication services in a partitionable network. Three multicast primitives are implemented according to the extended virtual synchrony semantics: causal multicast, agreed multicast and safe multicast. The Totem system [Amir95] proposes the Multiple-Rings protocol, which provides reliable delivery and ordering across the entire network. Gateways interconnect broadcast domains and participate in the ring protocol.

3 Group communication protocols in cooperative work

Users of cooperative software are gathered following different criteria such as distance, competency, the task at hand, … For example, during the design of a building people can be grouped either by corporation or according to the place or the time at which they will have something to do. Users' manipulations of the design must be broadcasted into the group. [Budhia96] distinguishes three types of protocols for group communication:

- **Asymmetric protocols** where messages are transmitted to a central coordinator that orders and broadcasts them. In this case, centralization leads to a bottleneck and provides a single point of failure;
- **Symmetric protocols** where each member can emit and receive messages. The disadvantages of asymmetric protocols are overridden but message ordering becomes costly;

- **Rotating sequencer protocols** where the right to emit is given to a member temporarily. In this case, message ordering keeps simple and there is not a single point of failure.

Works of K. Birman on the Isis [Birman94] and Maestro [Birman97] systems deal essentially with symmetric protocols that are quicker in communication time but more costly in terms of coherence management.

In case of rotating sequencer protocols, many protocols rely on the use of a token: possession of the token gives the right to emit. Rajagopalan and McKinley have described the use of such a protocol for message ordering for the first time with the TPM protocol (Token-Passing Multicast) [Rajagopalan89].

The Pilgrim algorithm [Guyennet97] is used to maintain consistency in a shared memory for cooperative applications. It uses a token on a virtual ring topology. The token holds information about the modifications of the shared objects

The use of a token on a ring has also been used in Totem [Agarwal94]. This system provides a total ordering of messages in process groups in a local area network but also over multiple interconnected networks.

The two main protocols inheriting from TPM are Total in Horus and On-Demand. Total [Renesse96] proposes the use of micro-protocols so that the token can adapt its way through the sites according to the applications' needs. The On-Demand protocol [Alvarez95] has been designed for applications with traffic peaks such as video applications. Only the member holding the token can broadcast his information. He passes the token only if another member reclaims it. This avoids the cost of useless token circulation. This protocol has inspired Rodrigues, Fonseca and Verissimo, which propose a hybrid protocol mixing the symmetric approach and token-based approach [Rodrigues96]. The Newtop protocol also mixes the two approaches so as to adapt to networks where some links are slower than others and where communication topology is not known in advance [Ezhilchelvan95].

The use of multiple types of protocols seems a good way to adapt to the needs of cooperative applications. Many systems build a new topology when changing protocols, which is costly and limits the possibility of reconfiguring exchanges in the group [Brattli98]. However, the use of multiple protocols poses the problem of the integrity of the system. Indeed, if two members do not use the same protocol, message ordering becomes difficult.

4 Dynamic group behavior

The group decomposition has already often been used in distributed applications. Currently, multimedia applications are appearing, such as cooperative work, videoconference, tele-diagnosis, tele-teaching and global-computing applications as the GLOBUS project. All these applications use the group notion. Groups have to be able to evolve dynamically and to structure themselves during applications runtime without interfere with their behavior.

This auto-organization property allows the system to organize itself in an autonomous way in group tree diagram. It also allows a group with a too important size to decompose itself in a tree diagram of smaller groups. Some constraints have to be respected for a consistent structuration and for the permanence of the architecture and of the links between members. Groups auto-organization relies on the consistency of information shared by group members [Malville-98]. With mobile terminals, for example with fixed servers and mobile clients, client groups will evolve continuously.

4.1 Dynamicity definition

Groups auto-organization protocols rely on some global constraints that allow agents to have autonomous and consistent behavior. In particular, all group members have to share consistent information.

4.2 Group view

The group view [Birman-91] [Malville-98] contains information relating to all group members. In particular, each member knows as well the list of all members as their structural characteristics CS_i. The $_G$ view of a G group is defined by the following equation:

$\text{view}_G=$ Members = $\{S_1,\ldots..S_n\}$

$CS=\{cs_1,\ldots\ldots.cs_n\}$

Groups auto-organization protocols rely on the members consistency. We can add, remove or modify the structural characteristics of a member.

Protocols are required to manage the modifications of structural characteristics and to guarantee the view consistency

4.3 Constraints

The group decomposition has to guarantee the view consistency of the members of the new groups. The protocol managing this decomposition has to meet three requirements:

- **When** performing the group decomposition. All servers can access a deterministic decision function. This function depends on the critical size of the initial group,
- **How** to decompose a group from there own view and information on servers,
- **How** to decompose a group while keeping a view consistency of members.

These constraints allows the members of a group to decide in an autonomous and uniform way *when* and *how* to decompose the original group.

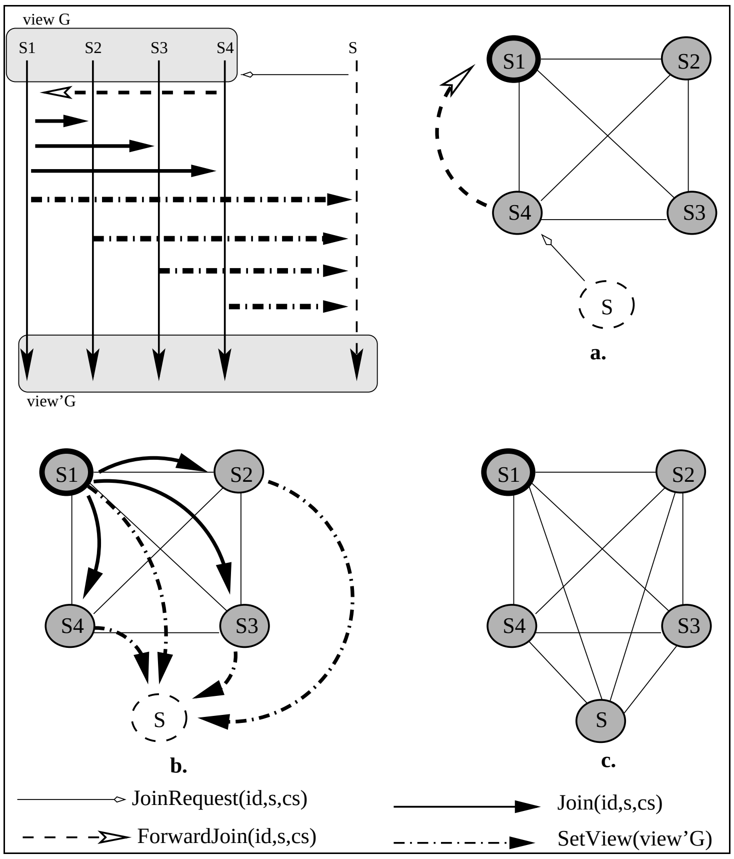

Fig 1. Inscription of a new member in a simple system

4.4 Inscription of a member in a group

In a simple system (composed of only one group, Fig. 1), when a server wants to join a group, it sends an inscription request (JoinRequest message) containing its structural characteristics, to any group member, this last transmits this message (ForwardJoin message) to the group leader which is the first member joining the group. The leader broadcasts to the other members, which allows them to update their view (Join messages) and returns the new view to the new member (SetView message).

With a system composed of several groups (Fig. 2), one step more is necessary : the JoinRequest must be sent to the superleader (site which is the leader of the leaders group).

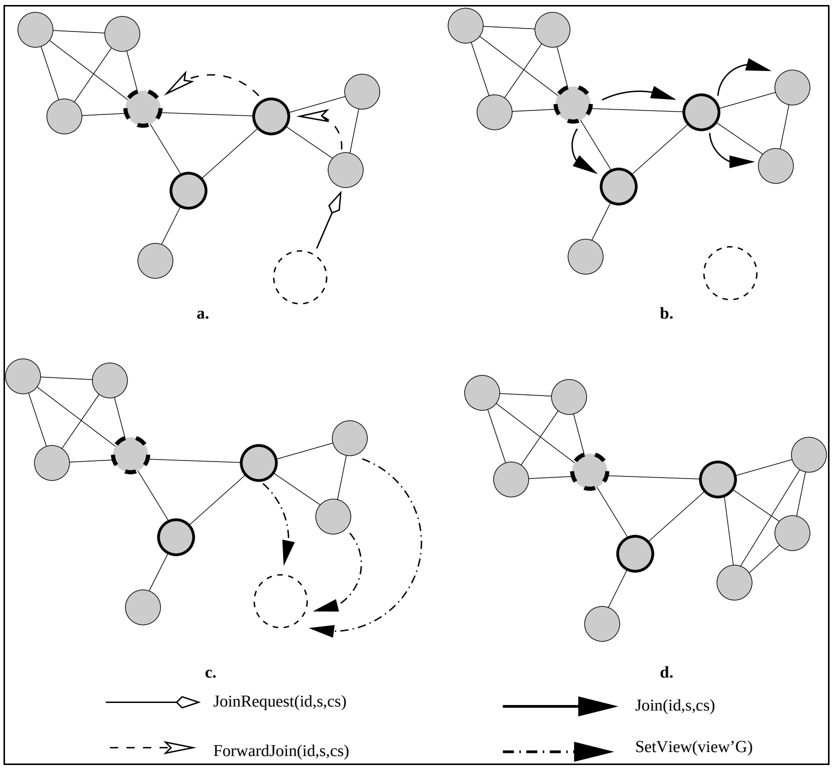

Fig 2. Inscription in a system composed of several groups

4.5 Leaving of a member

The server can decide to leave the group. It sends directly to the coordinator a leaving message containing its identifier. This last one broadcasts this information to the other members. When there is no member any more, a group can disappear.

4.6 Group decomposition

The group decomposition protocol is based on three global constraints:

- the views consistency
- global constants ($T_{max...}$)
- a deterministic decomposition function

It ensures an autonomous and consistent behavior of groups members.

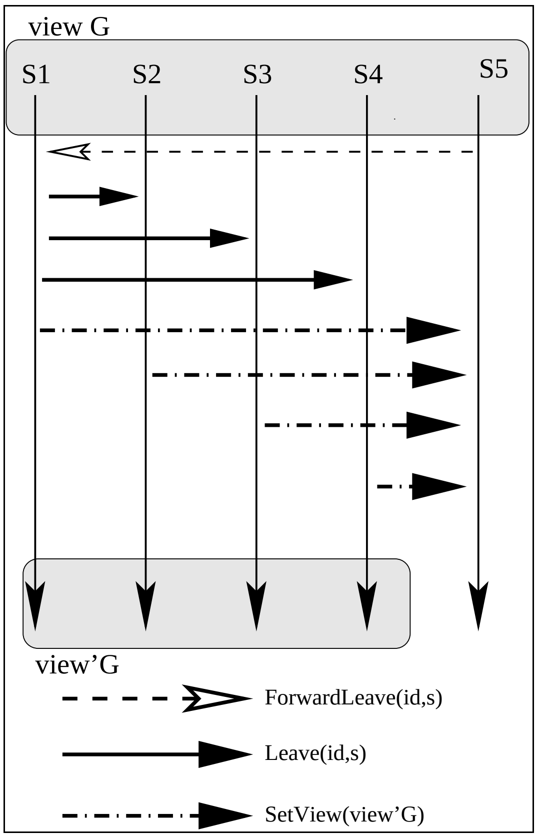

Fig 3. Leaving of a member

The group decomposition (if it takes place) is realized when a new member requests to join the group (Fig. 4). The inscription protocol has to preserve the views consistency of initial group members and of possibly new created groups. For example, if a server *s* wants to join a group G: when the leader *S1* receive a ForwardJoin message containing its structural characteristics, it updates its view and decides if the group has to be decomposed. If it is the case, it applies the decomposition function to it new view to determine which are the new groups to be created, as well as their contents. We suppose that the initial group *G* is decomposed in two groups *H* and *I*. *H* contains *S1* and *S2* severs and *I* contains *S3*, *S4*, *S*. The decomposition function allows the leader *S1* to determine the new view (*viewH*) and *viewI* of each new group. In the group *I*, *S3* becomes leader.

The *S1* leader role is to create the new *S3* leader of the second group and to broadcast a Join message to each member of the initial group. This message contains information relating to the new member, but also an ordered (by creation order) list of created groups leaders. The leader does not need to transmit other information as other members can also determine them from the current view of the group the can have by applying it the decomposition function. They only receive information they cannot determine in an autonomous way.

When a member receives a Join message, it sends it to its successors (to guarantee an atomic broadcast) and then it determines the created groups views, thanks to its own group view and other information contained in the message. So, it can update its own view and initialize the new member and new leaders one (SetView message).

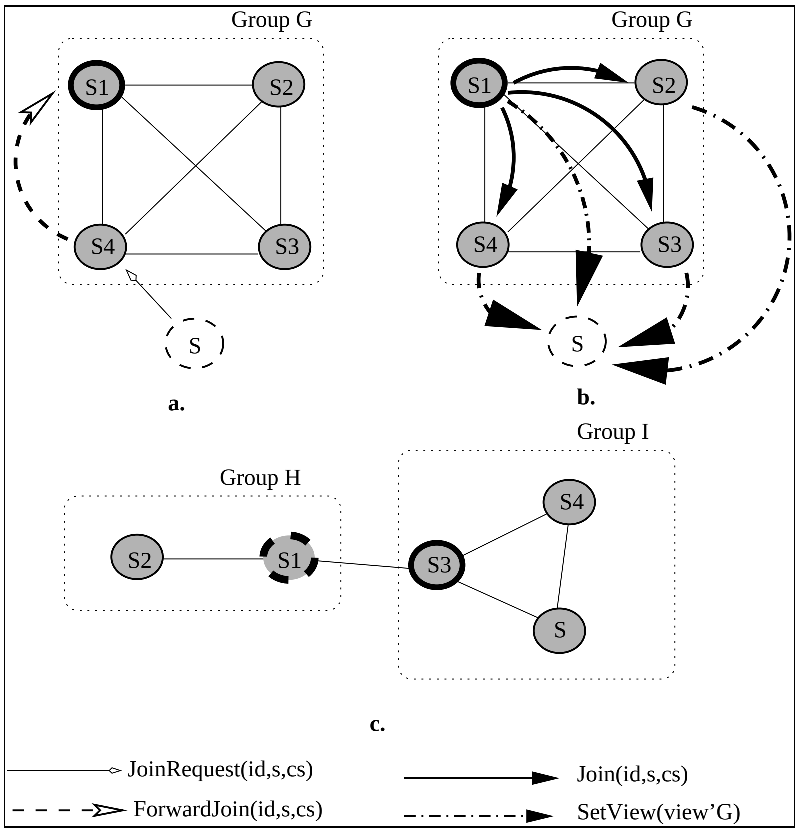

Fig 4. Functional decomposition example

5 Group services on top of CORBA

Several systems using group communication have been successfully experimented and are cited in previous paragraphs. Since CORBA, group communication systems using ORBs have been developed: Isis, Electra [Maffeis96], Eternal [Narasimhan97], OGS [Felber97], Newtop [Morgan99], TRM [Mishra01]. Three main approaches are implemented. Some systems use an integration approach, like Isis and Electra. In this approach, an existing ORB is modified and extended to include group communication mechanisms. In the second approach, this mechanism is implemented below the ORB in the lowest layer as in Eternal. In the third approach, group communication is provided by a service on top of an ORB. This last approach includes the most recent systems: OGS (CORBA compliant), TRM, The Pilgrim.

The Object Group Service (OGS) enables to view a group of CORBA objects as a single entity despite concurrent invocations and failures and thus provides an adequate support for the construction of highly available distributed applications with replicated critical components. The OGS manages groups of CORBA objects and provides primitives to communicate with these groups. Clients do not need to know the number, the identity, or the location of the members of a group. Similarly to the CORBA event service, the OGS provides two types of communication: untyped and typed communication. Untyped communication enables clients to send only values of type *any* as messages. Typed communication enables a client to directly invoke an operation of the server interface. Typed communication requires a specific binding phase executed by each client willing to get a reference to a group. OGS is modular and consists of a set of services:

- A Messaging Service that provides non-blocking reliable point-to-point channels.
- A Multicast Service that provides reliable multicast communication.
- A Monitoring Service that monitors objects to detect failures.
- A Consensus Service used to guarantee that only one response is returned for a request to the group.

Newtop contains several features not supported by other object group services. Objects can simultaneously belong to many groups, group size can be large. Group members can be geographically scattered, say communicating over the Internet. 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 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.

The most recent system is TRM [Mishra01]. Authors investigate the design, implementation, and performance of a group communication service using the Common Object Request Broker Architecture (CORBA). The results presented are based on an implementation of the timewheel atomic broadcast and the three-round, majority agreement group membership protocols. This paper provides a detailed comparison of the performance measured from three implementations of these protocols. Two of these implementations use CORBA, while the third uses UDP sockets for interprocess communication. The socket implementation provides the best performance, followed by the CORBA implementation. The message sizes in a group communication service are generally small and, so, the one-way communication delay will be significantly higher in a CORBA implementation than in a UDP socket-based implementation of a group communication service. The main conclusion is that the current CORBA technology can be used to provide heterogeneity properties to group communication services at some performance cost. However, a naive implementation of a group communication service in CORBA incurs substantial performance overhead.

The Newtop service offers a more comprehensive set of group management facilities than OGS. OGS does not support objects belonging to multiple groups like Newtop and supports only the asymmetric way of total ordering. 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. OGS attempts to preserve a unique subgroup ; so, when failures increase beyond a threshold, OGS blocks message delivery until it becomes possible to form the unique subgroup again.

6 Group service implementation on top of CORBA

In section five, we have seen there are three possibilities to implement a group service in CORBA: integration, interception and as a service. Even though the latter choice is not the most efficient, it provides portability and conformance to the CORBA standard. In this section, we describe the implementation of a group mechanism as a service on top of CORBA. While implementing a group service, one has to decide how the service will behave but also to make technical choices depending on the platform used which in our case is CORBA. The description will allow us to point out the relevant questions to answer while building a group service. The implementation will also be used in the next section to study the behavior of group mechanisms.

6.1 Service presentation

We have developed a group service adapted to cooperative work. We use distributed shared memory to manage the consistency of discrete media. The main parts of the service are: group membership management, internal group organization, communication management. This service allows cooperating members to broadcast their information.
All modified objects of the distributed shared memory are transported by way of this service. A consistency service interacts with this service to maintain the integrity of the application.
We chose to develop a new token strategy based algorithm, this kind of algorithm is efficient to manage objects consistency in cooperative work [Garcia00]. The originality of this algorithm is to allow the virtual topology of the group to be reconfigured using two communication techniques: a rotating sequencer for one site and a symmetric approach for another that is not very active, for example (figure 7). Another important characteristic of this algorithm is that the representation of the virtual topology (the *group view*) is a distributed shared object and is transported by the token.
The token is not a simple tool allowing a site to broadcast, it contains the virtual topology representation: the IDL definition of Token in figure 5 contains the GroupView. The consistency of this topology has to be maintained, to ensure that all the sites have the same view of the system. The nature and the treatment of the token depend on the state of the sites that receive it

(leader, slave…). All members can change of state (potential leader, slave…) according to their activity rate, and then the group topology can be dynamically modified with active and low active sites (figure 9).

6.2 Group organization management

6.2.1 Group membership

Group membership management deals with the inscription and withdrawal of members of the group. The group manager described in the next section handles this. Group membership is closely related to the organization of the group since adding or removing a member can trigger a change in the topology according to the organization policy.

```
struct Identifier {                 // a group member
   string Host;
   string Name;
   short Port;
   short State;                     // Potential leader, slave…
   short NbSlaves;                  // if leader
 };
 typedef sequence<Identifier> ListIdentifier;

 struct Change {
   short Index;
   short CodeChange ;
   string NameChange;
   short NbViewed;
   ListeIdent ListViewer;
 };
 typedef sequence<short> ListeIndex;
 typedef sequence<Change> ListeChange;

 struct GroupView {
   ListeIdentifier  ListId;         //list of members
   ListeChange ListeIdChange;
 };
 struct Token {
   GroupView the_groupview;
   string message;
   … }
```

Fig 5. IDL interface

6.2.2 Group Manager

The group manager is at the same time the representative of a group from the client's point of view and the manager of the group's behavior. In our service, group managers are active sites that own the token. If a slave site receives a joinResquest message from a potential new member, it has to forward this message to an active site.

This last site can treat this request when it will receive the token making it group manager or group leader. An active site (rotating leader) uses the SendTokenRotating method (Fig 6.) to broadcast its information to the other active sites and it uses the SendTokenSymmetric method to communicate with its slaves (low active sites).

6.2.3 Group Behavior

In section 4 we have seen that group auto-organization allows the system to organize itself in an autonomous way, and that it relies on the consistency of information shared by group members. Our service ensures the consistency of the group view on each site involved in the cooperative work. Group auto-organization can use numerous criteria: number of members (section 4), but also distance between members, cooperation rate, activity…), in our case we use the activity rate of members to decompose a group in sub-groups.

If all members of a group are active, they communicate with a rotating sequencer protocol (section 3), a token ring strategy based algorithm. All active sites are potentially leaders, and they become effectively leaders when they receive the token, they can then act as group manager (figure 9a).

A group is decomposed if one or more sites become low active sites. In figure 2 we see that the setInactivityLimit method allows us to determine when a site becomes low active. At this time, one or more sub-groups composed of an active site (leader) and low active sites (slaves) are created. A symmetric communication protocol is used between a leader and its slaves, and the rotating sequencer approach continues to be used between all potential leaders (Figure 9b, 9c, 9d).

If a member whishes to join or leave the group, it can send its request to any member of the groups. But, the effective modification of the topology has to be performed by a leader possessing the token (the super-leader). So, if a slave site receives such a message it has to forward it to its leader that has to wait for the token to perform the modification. Principles of our service are quite close to principles described in section 4.

```
interface Site {
   …
oneway void join(in Identifier id_NewSite,in Site site);
oneway void leave(in short index, in Identififier id);

void SendTokenRotating (in Token the_token);
void SendTokenSymmetric (in Token the_token);

void zeroNbInactivity();
void setInactivityLimit(in short v);
          …
```

Fig 6. Site IDL interface

6.3 Communication management

The communication management part is responsible for the delivery of data to the members according to a communication policy (e.g., FIFO, causal ordering, ...).

In section 3, we have seen that there are 3 types of group communication protocols used in cooperative work: Rotating sequencer, Symmetric and Asymmetric.

Our service starts with a rotating sequencer approach when sites are active: ring topology is more efficient than a full-connected network topology for a multi-broadcast operation; however this tendency is reversed when the number of active sites decreases. So, we use the second method (SendTokenSymmetric in figure 6) for the sites that do not participate actively to the cooperative work to complete the ring which links active sites.

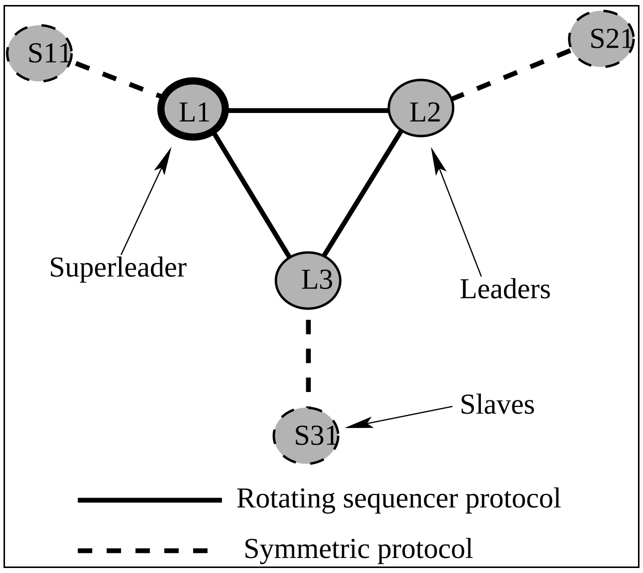

Fig 7. Balancing of slave sites

6.4 Performance

The group service of our platform is implemented with Orbacus [ORBACUS99] and C++. Tests are performed on a 100 Mbit LAN with a PentiumIII PC cluster.

In a first test (figure 8) we study the effect of the number of low active sites and the choice of their leader on the performance. In abscissa, we can see the number of low active sites in the topology. In ordinate, we can see the time in second for 3000 ring turns. In case of a decomposed group, a ring turn corresponds to the time taken by the token to visit each leader site (rotating sequencer protocol) and also to the additional time taken by each leader to broadcast its information to its slave sites (symmetric protocol). Results are note the sum of these 2 times, indeed, broadcast to slaves performed by leaders can be partially or totally done during the routing of the token between leaders.

We work with a virtual topology with 6 sites, and can see that the increase of the slave site number improves performance. The optimal number is not, however, the maximum one. Indeed, when it becomes greater than the number of active sites, there is a problem of congestion. The token comes back to a leader site before the slave sites have received and treated their previous copy. As broadcast is a blocking operation (for fault tolerance and synchronization), the leader site has to wait for the slave site release. If multicast is not implemented in the system used, the balancing technique is better than the centralized one. With 3 group leaders and 3 slaves sites (figure 7) the centralized technique is 25 percent slower than with balanced leaders: this situation corresponds to the lower point of the *Balancing* curve in figure 8. Indeed, when there is only one leader it has to wait one stage more for its slave site release.

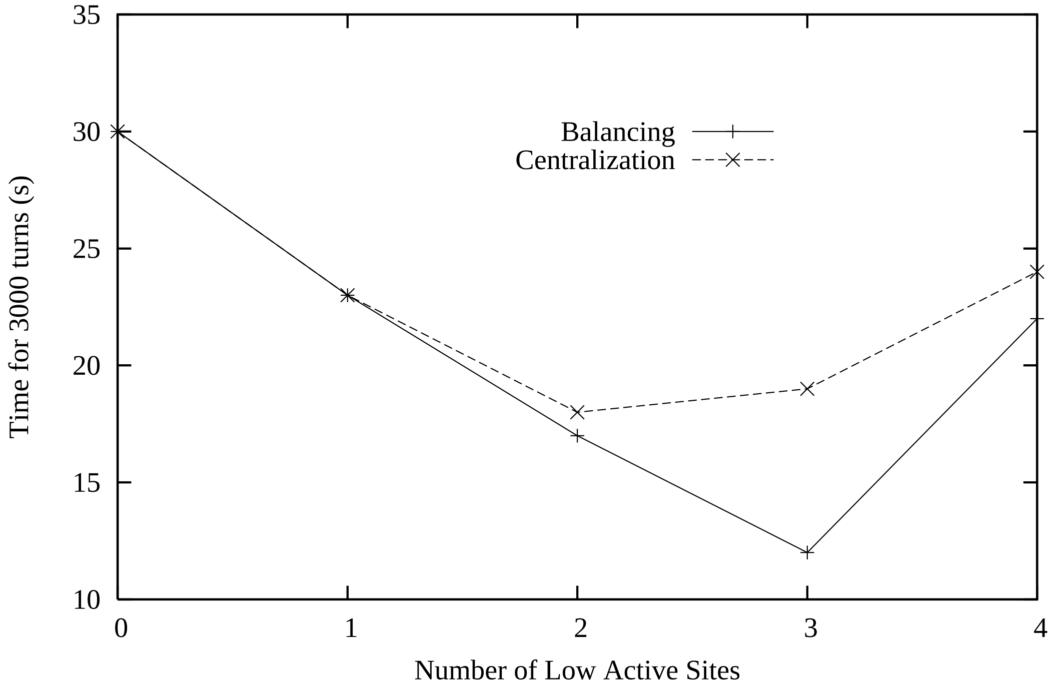

Fig 8. Comparison between balancing and centralized techniques

These tests allow us to answer the question "how to decompose a group ?" (section 4.3), to obtain the best group topology according to site states.

A second test shows the performance of our group service with different group topology. Figure 9 represents the different topology used and figures 10 shows their respective performance in function of the token size shown in abscissa. Figure 9a represents a topology with 8 active sites using the rotating sequencer technique and corresponds to the curve *8A* (8 active sites) of figure 10. Figures 9b, 9c, 9d represent topologies with actives and low actives sites, respectively with 4, 3 and 2 groups. The tests about the comparison between balancing and centralized techniques (figure 8) have allowed us to choose how to decompose a group: we use a balancing. In figure 9b we can see 4 low active sites with 4 leaders, it corresponds to the curve 4A/4LA (4 active sites and 4 low active sites) of figure 10. In figure 9c there are 5 slaves sites with 3 leaders (3A/5LA) and in figure 9d, 6 slaves with 2 leaders (2A/6LA).

We can see, in figure 10, that the best results are obtained when the group auto-organization leads to topologies *b* or *c*. For any token size these two topologies perform 3000 ring turns in less than 17 seconds. Communications are twice faster for these topologies compared with the other topologies with 8 active sites or with 2 group containing a leader and 3 slaves. We observe the same behavior than for the first test where the best results are obtained with an average number of slave sites and not a maximum one.

Group auto-organization with a dynamic topology reconfiguration is efficient if we use an appropriate heuristic to determine if decomposition is necessary. These tests can allow us to answer the question "when to decompose a group ?" , and to set an efficient inactivity limit. We can observe the behavior of cooperative application users and use our tests to set a limit that meets our requirements.

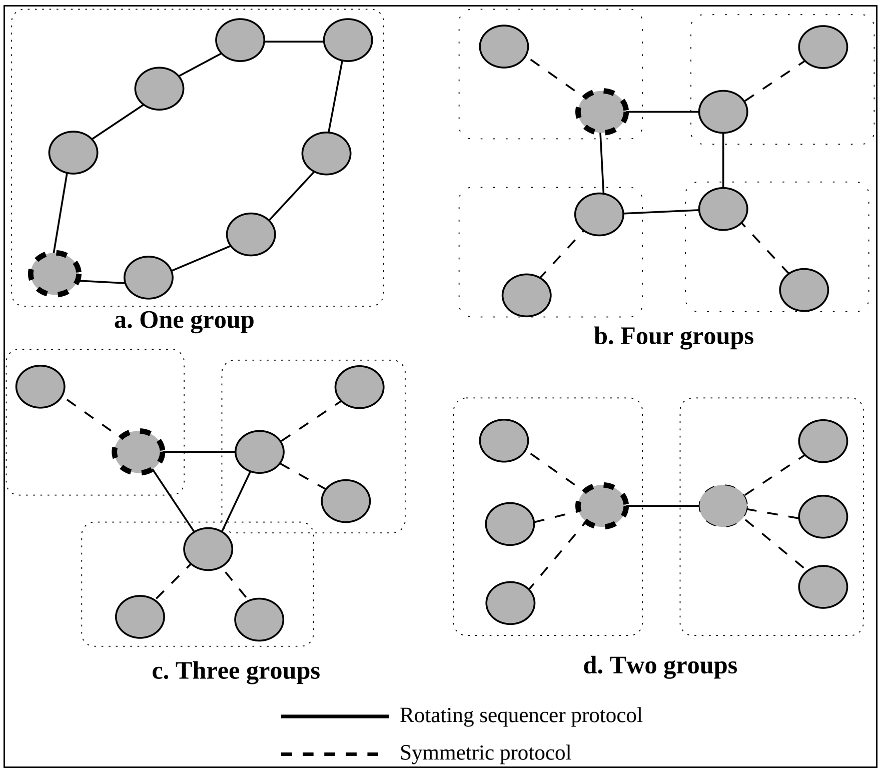

Fig 9. Auto-organization of a group

A third test presents gains obtained when an active site becomes low active, and allows us to see the impact of site speed on the group decomposition (Figure 11). In this test sites called Fast sites are equipped with 100Mb/s Ethernet cards and slow sites with 10Mb/s cards.

For the start configuration with 4 fast active sites, we see that the time taken by the token to perform 3000 turns increases with its size. For a token containing 2500 elements of 4 bytes (10 Kbytes) it is 12.5 seconds. When one of the active sites becomes low active, performance is improved from 8 percent for a 10-element token to 20 percent for a 2500-elements token. The gain depends on the token treatment time. If this time is great (in terms of the token size), the obtained gain will also be greater because the treatment of the token is performed in parallel on two sites.

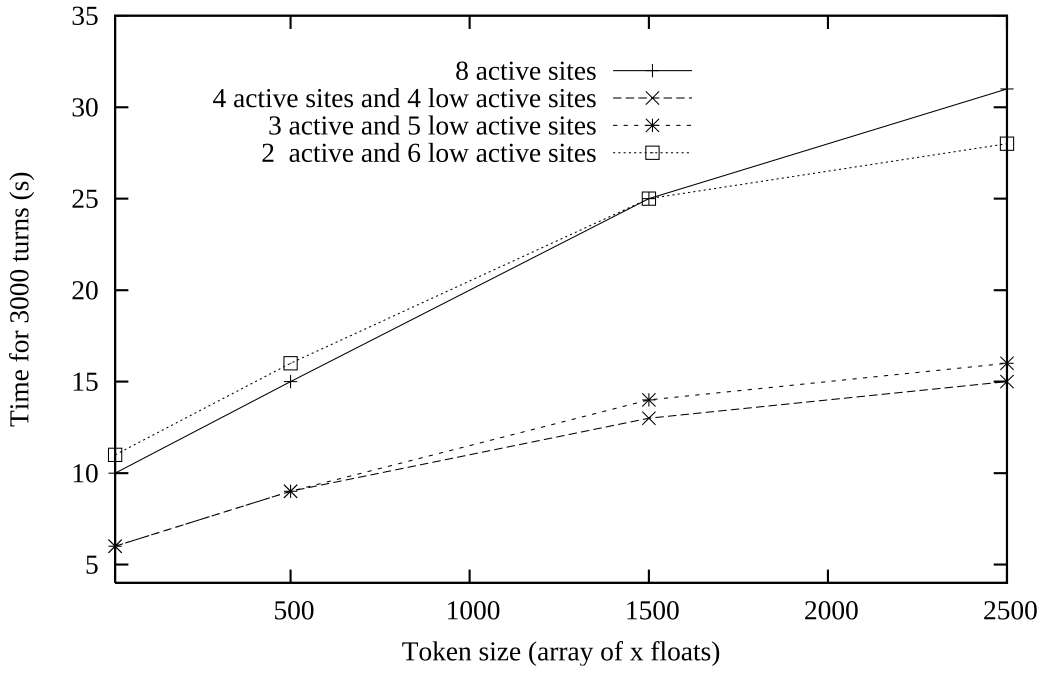

Fig 10. Performance of group auto-organization

For the start configuration with 3 fast active sites and 1 slow active site, it is efficient to place the slow site in low activity for a token size below 1500 elements (6Kbytes). Beyond that point, performance decrease due to the congestion point represented by the slow site. Indeed, the leader of the slow site has to wait that this last has performed its treatment before receiving the new token.

All tests give good results considering that most cooperative applications use a small (less than 5 Kbytes) token to manage discrete media consistency. Group auto-organization needs to answer several questions, "when and how to decompose a group?…", and needs also to know the impact of a decomposition according to site power, active/low active sites ratio… So, we can propose a group service on top of CORBA, which offers functions to manage operations in the group, but some parameters have to be adjusted to meet particular cooperative application requirements.

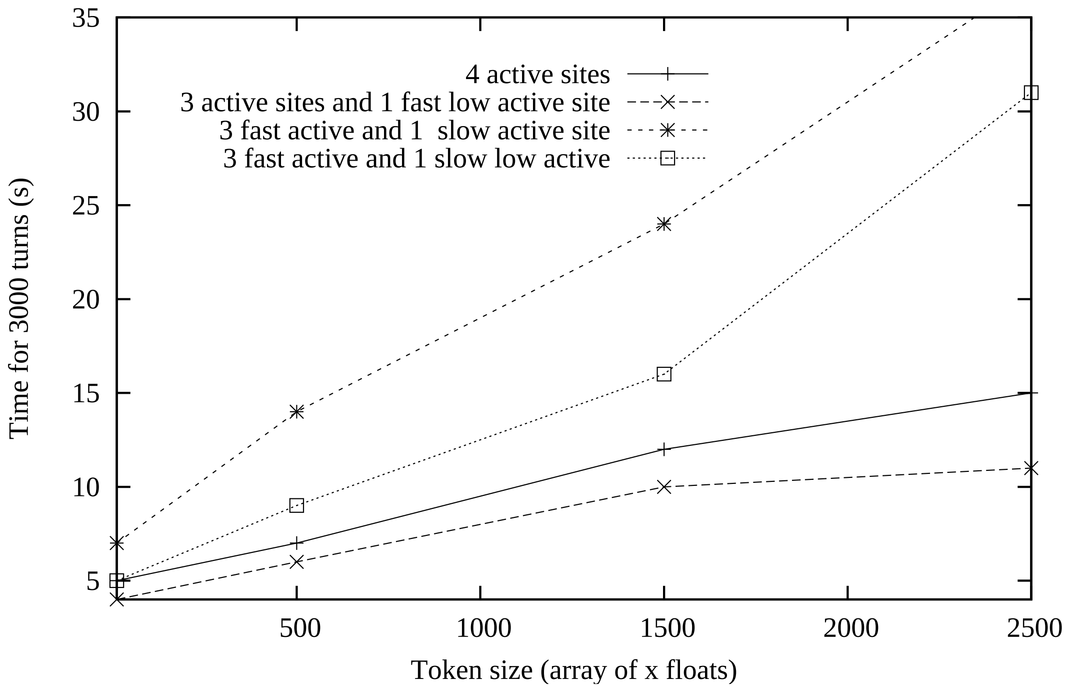

Fig 11. Introduction of different kinds of slave sites

7 Conclusion

Group mechanisms have long been considered a useful solution to handle large collections of entities such as processors or processes. It is a tool to share a common name for them and to handle group communications. It is also a good solution to group entities sharing similar properties or stakes. This is particularly accurate when applied to the cooperative work field. Group partitioning of processes or processors is also an approach that is of increasing interest, especially with the emergence of nomadic applications. It is the only approach to achieve scalability by grouping members into domains, while inter-domain communications are done only by domain coordinators.

In the area of distributed applications, communication has evolved from socket based interfaces to Remote Procedure Calls and nowadays, towards distributed object platforms like CORBA. This standard from the Object Management Group hides distribution and heterogeneity and provides a good basis for distributed application development. In this paper, we review existing projects dealing with the realization of a group mechanism for CORBA environments. Then we highlight the concepts to consider and the decisions to make in order to develop a group service over CORBA. This is illustrated by an implementation for such a service over CORBA. This service performs group auto-organization according to sites activity rates. Tests show that our service is efficient and provide us some bases to determine when and how to decompose groups to obtain optimal performances.

References

[Agarwal94] D. A. Agarwal, *Totem: A Reliable Ordered Delivery Protocol for Interconnected Local-Aera Networks*, PhD Thesis, University of California, Santa Barbara, August 1994.

[Ahamad91] M. Ahamad and M.H. Ammar and S.Y. Cheung: *Multidimensional Voting*, ACM Transactions on Computers Systems, 9(4), 399-431, November 1991.

[Ahmad91] I. Ahmad and A. Ghafoor: *Semi-distributed load balancing for massively parallel multicomputer systems*, IEEE Trans. on Soft. Eng., 17, 987-1004, 1991.

[Alvarez95] G. Alvarez and F. Cristian and S. Mishra*, On-demand asynchronous atomic broadcast*, Proceedings of the 5th IFIP International Conference on Dependable Computing for Critical Applications, pp 68-78, Urbana-Champaign, 1995.

[Amir96] Y. Amir, D. Breitgand, G.V Checkler, D. Dolev: *Group Communication as an Infrastructure for Distributed System Management* The International Workshop on Services in Distributed and Networked Environment (SDNE), Macau, pp 84-91, June 1996.

[Amir92] Y. Amir, D. Dolev, S. Kramer, D. Malki: *Membership Algorithms for Multicast Communication Groups* 6th Intl. Workshop on Distributed Algorithms proceedings (WDAG), LNCS 647, pp 292-312, November 1992.

[Amir95] Y. Amir and al.: *The Totem Single-Ring Ordering and Membership Protocol*, in ACM Transactions and Computer Systems, 13(4), pp 311-342, 1995.

[Amir98] Y. Amir, J. Stanton: *The Spread Wide Area Group Communication System* Technical report CNDS-98-4, The Center of Networking and Distributed Systems, the Johns Hopkins University, 1998.

[Anker97] T. Anker, G.V. Chockler, I. Keidar, M. Rozman, J. Wexler: *Exploiting Group Communication for Highly Available Video-on-Demand Services* The proceedings of the IEEE YUFORIC on Multimedia Information Systems and the 13th International Conference on Advanced Science and Technology (ICAST97) and the 2nd International Conference on Multimedia Information Systems (ICMIS 97), pp 265-270, April 1997.

[Balayer95] C. Balayer, C. Daval-frerot, H. Guyennet: *The Processor Group Approach to Dynamic Load Balancing* in ISMM Parallel and Distributed Computing and Systems, Washington, October 1995.

[Bangalore94] P.V. Bengalore, N.E. Doss, A. Skjellum: *MPI++: Issues and Features*, Mississipi State University, U.S.A., March 1994.

[Beier98] I. Beier, H. Koenig: GCSVA - *A Multiparty Videoconferencing System with Distributed Group and QoS Management* International Conference on Computer and Networks. Lafayette, Louisiana. October 12-15, 1998.

[Benford93] S. Benford, J. Palme: *A Standard for OSI Group Communication* Computer Networks and ISDN Systems, 25(8), March 1993.

[Bensaber97] B.A. Bensaber, D Seret: *Protocoles pour les Communications Multicast*, Colloque International sur les Nouvelles Technologies de la Répartition, pp. 303-316, 1997.

[Billard93] E.A. Billard, J.C. Pasquale: *Effect of Delayed Communication in Dynamic Group Formation*, IEEE Trans. on Systems, Man, and Cybernetics, vol 23(5), September 1993.

[Birman87] K.P. Birman, T.A. Joseph: *Reliable Communication in the Presence of Failures* ACM Transactions on Computer Systems, 5(1), pp. 47-76, February 1987.

[Birman91] K.P. Birman, R. Cooper, B Gleeson: *Programming with Process Groups: Group and Multicast Semantics* Papers of the ACM, 37-53, December 1991.

[Birman91-2] K. Birman, A. Schiper, P. Stephenson: *Lightweight Causal and Atomic Group Multicast* ACM Transactions on Computer Systems, Volume 9(3), pp. 272-314, 1991.

[Birman94] K. P. Birman and R. van Rennesse: *Reliable Distributed Computing using the Isis Toolkit* IEEE Computer Society Press, 1994.

[Birman97] K. Birman, R. Friedman, M. Hayden: *The Maestro Group Manager: A Structuring Tool for Applications with Multiple Quality of Service Requirements* Technical Report 97-1619, Department of Computer Science, Cornell University, Ithaca, NY 14850, 1997.

[Birman98] K.P. Birman, R. Friedman, M. Hayden, I. Rhee: *Middleware Support for Distributed Multimedia and Collaborative Computing* In Multimedia Computing and Networking (MMCN98), 1998.

[Birman93] K.P. Birman: *The Process Group Approach to Reliable Distributed Computing* Papers of the ACM, 37-53, December 1993.

[Brattli98] D. Brattli: *A Survey of Dynamic Configurable Protocol Stacks* http://www.cs.uit.no/dagb/dynamic-protocols/dynamic-protocols.html, University of Tromso, Faculty of Science, Department of Computer Science, Norway, 1998.

[Braudes93] R. Braudes, S.Zabele: *Requirements for multicast protocols*, Network Working Group, RFC 1458, 1993.

[Budhia96] R. Budhia: *Performance Engineering of Group Communication Protocols* PhD thesis, University of California, 1996.

[Chandra96] T.D. Chandra, V. Hadzilacos, S. Toueg: *On the Impossibility of Group Membership* ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), pp 322-330, 1996.

[Chockler96] G.V. Chockler, N. Huleihel, I. Keidar, D. Dolev: *Multimedia Multicast Transport Service for Groupware* In Proceedings of the TINA Conference on the Convergence of Telecommunications and Distributed Computing Technologies, September, pp 43-54, 1996.

[Cristian96] F. Cristian: *Synchronous and Asynchronous Group Communication* Communications of the ACM (CACM), Volume 39(4), pp 88-97, April 1996.

[Dolev96] D. Dolev, S. Kramer and D. Malki: *The Transis approach to high availability* cluster communication, Communications of the ACM,39(4), 1996.

[Ellis91] C. Ellis and S. Gibbs and G. Rein: *Groupware. Some Issues and Experiences*, Paper of ACM, 34(1), 35-58, January, 1991.

[Evans94] D.J.Evans, W.U.N. Butt: *Load Balancing with Network Partitioning Using Host Groups*, Parallel Computing, 20, 325-345, 1994.

[Ehilchelvan95] P. D. Ezhilchelvan and R. A. Macêdo and S. K. Shrivastava: *Newtop: A fault tolerant group communication protocol* Proceedings of the 15th IEEE International Conference on distributed computer systems, pp 296-306, Los Alamitos, CA, USA, May 1995.

[Felber96] P. Felber, B. Garbinato, R. Guerraoui: *The Design of a CORBA Group Communication Service* Symposium on Reliable Distributed Systems, pp 150-159, 1996.

[Felber98] P. Felber, R. Guerraoui, A. Schiper: *A CORBA Object Group Service* Theory and Practice of Object Systems, Volume 4(2), pp 93-105, 1998.

[Felber98b] P. Felber: *The CORBA Object Group Service - A Service Approach to Object Groups in CORBA* PhD Thesis, Ecole Polytechnique de Lausanne, Swiss, 1998.

[Foster01] I. Foster, C. Kesselman, S. Tuecke: *The Anatomy of the Grid* Int. Journal Supercomputer Applications, 15(3), 2001.

[Garcia00] E. Garcia, J-C. Lapayre, G. David, *Pilgrim Performance over a New CALiF Communication Layer.* Proceedings of The IEEE International Conference on Parallel and Distributed Systems, ICPADS'2000, pp 203-210, Iwate Japan, July 2000.

[Geist94] A. Geist et al.: *PVM 3 user's guide and reference manual*, Manuel de reference PVM 3, Oak Ridge National Laboratory, May 1993.

[Guo97] K. Guo, L. Rodrigues: *Dynamic Light-Weight Groups* Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97), Baltimore, Maryland, USA, May 1997.

[Guyennet96] H. Guyennet and J-C. Lapayre: *A Co-operative Application Management Platform Based on Shared Virtual Memory*, pp 457-462, IEEE Proceedings of the HiPC'96 International Conference, Trivandrum Inde, December, 1996.

[Guyennet97] H. Guyennet and J-C. Lapayre and M. Tréhel: *Distributed Shared Memory Layer for Cooperative Work Applications*, pp 72-78, IEEE Proceedings of the LCN'97 International Conference, Minneapolis USA, December 1997.

[Guyennet00] H. Guyennet, M. Trehel: *Load Balancing Using Processor Groups,* Parallel Processing Letters, World Scientific Publishing Compagny, Vol 10, N°1, pp 59-72, 2000.

[Guyennet01] H. Guyennet, J.C. Lapayre: *The Group Approach in Cooperative Work and in Load Balancing*, Journal of Parallel and Distributed Computing Practices, Vol 2, N°3, pp 285-297, 2001.

[Kaashoek93] M.F. Kaashoek, A.S. Tanenbaum, K. Verstoep: *Group Communication in Amoeba and its Applications* Distributed Systems Engineering Journal, vol 1, pp 48-58, July 1993.

[Khazan98] R. Khazan, A. Fekete, N. Lynch: *Multicast Group Communication as a Base for a Load-Balancing Replicated Data Service* International Symposium on Distributed Computing (DISC) 1998, pp 258-272.

[Macedo94] R.A. Macedo: *Fault-Tolerant Group Communication Protocols for Asynchronous Systems*, Ph.D., University of Newcastle, 1994.

[Malville98] E. Malville, " Building Groups Dynamically: a Corba Group Self-Design Service ",

[Mishra01] S. Mishra, L. Fei, X. Lin, and G. Xing: *On Group Communication Support in CORBA* IEEE Transactions on Parallel and Distributed Systems, Vol 12, N. 2, pp 193-208, Feb 2001.

[Morgan99] G. Morgan, S.K. Shrivastava, P.D. Ezhilchelvan, M.C. Little: *Design and Implementation of a Corba Fault-Tolerant Object Group Service* Proceedings of the Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems, DAIS'99, Helsinki, June 1999.

[Moser96] Moser, P.M Melliar-Smith, D.A Agarwal, R.K. Budhia, C.A Lingley-Papadopoulos: *Totem: A fault-Tolerant Multicast Group Communication System* Communications of the ACM Vol.39, No. 4 , April 1996, pp 54-63.

[Nakamura92]. Nakamura, T. Tachikawa, M. Takizawa: *Group Communication Protocols: properties and evaluation* TR, Dept of Computers and Systems Engineering, Denki University, Tokyo, Japan, 1992.

[OMG00] Object Management Group: *Life Cycle Service Specification* Version 1.1 April 2000.

[OMG01] Object Management Group: *The Common Object Request Broker: Architecture and Specification* - Revision 2.5, September 2001.

[Powell96] Powell: *Group Communication* Communications of the ACM (CACM) 39(4): pp 50-53, 1996.

[Ramamritham89] K. Ramamritham, J.A. Stankovic, W. Zhao: *Distributed Scheduling of Tasks with Deadlines and Resource Requirements* IEEE Trans. On Comp., pp 1110-23, 1989.

[Rajagopalan89] B. Rajagopalan and P.K. McKinley: *A token-based protocol for reliable ordered multicast communication* Proceedings of the 8th IEEE Symposium on Reliable Distributed Systems, pp 84-93, Seattle, WA, October 1989.

[Raverdy96] P.G. Raverdy: *Gestion de Resources et répartition de Charge dans les Systèmes Hétérogènes à Grande Echelle* PhD Thesis, Paris VI, 1996.

[Raymond96] K. Raymond: *Reference Model of ODP*, CRC for Distributed Systems Technology, University of Queensland, Australia, 1996.

[Renesse95] R. van Renesse and al.: *A Framework for Protocol Composition in Horus* in Proc. of the ACM Symposium on Principles of Distributed Computing, 1995.

[Renesse96] R. Van Renesse, K. Birman, S. Maffeis: *Horus: A flexible group communication system* Communications of the ACM, 39(4): pp 76-83, April 1996.

[Rodrigues96] L. Rodrigues, H. Fonseca and P. Verissimo: *Totally ordered multicast in large-scale systems* Proceedings of the 16th International Conference on Distributed Computing Systems , Hong-Kong, pp 503-510, May 1996.

[Rosti94] E. Rosti et al.: *Robust partitioning policies of multiprocessor systems* Performance Evaluation North Holland, 19(2-3), pp 141-165, 1994.

[Reiter92] M.Reiter, K. Birman, L. Gong: *Integrating Security in a Group Oriented Distributed System* Technical report, Cornell University, 1992.

[Tannenbaum92] A.S. Tannenbaum: *Modern Operating Systems*, Prentice-Hall, 1992.

[Vaughan95] J.G. Vaughan: *A Hierarchical Protocol for Decentralizing Information Dissemination in Distributed Systems* The Computer Journal, Vol. 38, 1, pp 57-70, 1995.

[Verissimo92] P. Verissimo, L. Rodrigues, W. Vogels: *Group Orientation: a Paradigm for Modern Distributed System* In Proceedings of the ACM SIGOPS Workshop, 1992.

[Zhou92] Zhou: *Utopia - A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems* Software: Practice and Experience. TR CSRI-257, Toronto, 1992.