PDPTA'2000
Paper : 1472P

A Reference Design Model for Group Communication Systems

Saul POMARES HERNANDEZ, Khalil DRIRA, Jean FANCHON
Laboratoire d'Analyse et d'Architecture des Systèmes
7, Av. du colonel Roche - 31077 Toulouse - France

Abstract

The paper proposes a reference design model that identifies groups, and organizes the main capabilities or functional components of group communication systems. These components or capabilities are organized in layers which reflect their relationships. Each layer provides certain services to the upper layers. The main purpose of the model's layered structure is to reduce the design and implementation complexity of group communication protocols.

Keywords : Group communication systems, Group communications protocols, Distributed and parallel systems, Design models.

1. Introduction

Today, different group communication protocols for distributed and parallel systems are available [1], [2], [3], [4], [5], and their development has reached a good degree of maturity. They can be used to construct applications with highly available security domains [6], real-time responsive requirements [7], and group mechanisms that fit well with an emerging generation of intelligent network and collaborative work applications [8]. In addition, they provide availability, dependability, and timeliness properties to applications.

Unfortunately, neither their design nor their structure is clear. For this reason, a motivation has arisen to create a reference design model that allows one to identify, to classify, to group, and to order those parts that are fundamental for an efficient and reliable group communication.

The rest of this paper is structured as follows : Section two briefly mentions some of the more important issues in group communication. The presumed reference model is described in more detail in Section three, and finally Section four provides some conclusions.

2. Design Issues for the Group Communication Layers

Below, we will briefly mention some of the more important issues in group communication.

Every group member needs a mechanism for identifying the sending and receiving process groups. Since a group normally has many members, some which belong to several groups, a mean is needed for a process to identify the group with whom it wants to communicate. As a consequence of having multiple process group destinations, some form of *BS* (Bearer Service) *addressing* and *naming* (logical addressing) is needed in order to specify a specific group destination.

Another set of design decision concerns the *flow-control*. This is very important; for example, in order to implement the atomicity property (either all correct group members receive a message *m* or none do), each group member stores every message it receives in a local buffer until it learns that the message is *stable* (i.e. the message has been received by all correct members). In the absence of any specific rules, the local buffer can grow very large and may overflow, thus causing a member's failure. In order to avoid the overflow, some flow control techniques exist which will be discussed later.

In asynchronous systems the process groups do not share storage. They exchange messages via a datagram communication service. In networks WAN, messages can arrive in different order to group members. At times, it is important to preserve the order of messages sent to the group; other times, this is not necessary. It depends on the application. Several types of *orderings* of messages have been proposed, each one of them is applied to different situations. Each situation and orderings will be studied more ahead.

3. The Reference Model.

In order to reduce their design complexity, the model is composed of five layers; each level built upon the one below it. Each layer provides certain services to the upper layers, shielding those layers from the details of how the offered services are actually implemented.

Layer *n* of a process carries on a conversation with layer *n* on another process. The rules and conventions used in this conversation are collectively known as the layer *n* micro-protocol. In reality, no data are directly transferred from layer *n* of a process to layer *n* on another process, each layer passes data and control to the layer below it, until the lowest layer is reached.

Between each pair of adjacent layers there is an *interface*. The interface defines which primitive operations and services the lower layer offers to the upper one.

A set of layers and micro-protocols is called a *Group Communication Architecture*. The specification of an architecture must contain enough information to allow an implementer to write the program for each layer so that it will correctly obey the appropriate protocol.

The model is shown in Fig. 1. The model is organized in five layers or levels. The model is called GCRM (Group Communications Reference Model). The principals that were applied to arrive at the five layers are as follows:

1. A layer should be created where a different level of abstraction is needed.
2. Each layer should perform a well-defined function.
3. The layer boundaries should be chosen to minimize the information flow across the interfaces.
4. The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.

Below we will discuss each layer of the model individually, starting at the bottom layer. Note that our model itself is not a Group Communication Architecture because it does not specify the exact services and protocols to be used in each layer. It just tells what each layer should do and it shows some alternatives.

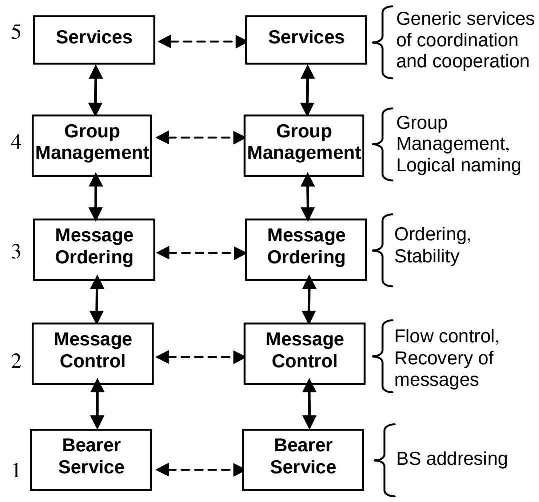

Fig. 1 The Reference Model GCRM

3.1 The Bearer Service Layer

The Bearer Service Layer is concerned with the *BS addressing* that identifies process groups over a network.

There are two methods of group *BS addressing*. One method is to use a *single* address for the whole group [5]. The second method requires sending individual packages to each one of the group members. The main advantages to having a single address that identifies the group are that it saves bandwidth and it allows a process to send a message without knowing which processes are members of the group.

The implantation of the method depends on the capabilities of the network. In certain networks, it is possible to create a special address that can be "listened" by several machines. When a message is sent to one of these addresses, an automatic delivery to all the machines that "listen" at that address is made. This technique is called *multicast*. In other words, multicasting describes data transmission in which a source sends a single copy of data to multiple recipients simultaneously. The implantation of the groups by means of multicast is direct: it is only necessary to assign to each group an address of different multicast. Some examples of networks that support multicast are: IEEE Standard 802.3 (Ethernet) [20] and IP Multicast [19]. Ethernet provides 2^{47} multicast addresses and IP Multicast 2^{28} multicast addresses, so in IP Multicast over 250 million groups can exist at the same time and even more on Ethernet.

Another technique is to *broadcast*, which means that the addressed frames are given to all the machines of the network. Each machine receives the transmission; for this reason, its software must verify that the package is directed to the machine. If not, the package is discarded. A certain amount of time is lost during the processing of the interruption. Nevertheless, the package arrives to all the members of the group.

If neither multicast nor broadcast is available, the group communication can be made (by the transmitter) by sending individual packages to each one of the members of the group. For a group with *n* members, *n* packages are needed, instead of only one package if multicast or broadcast is used. Although this method is less efficient, its implantation will work.

3.2 The Messages Control Layer

The message control layer is concerned with the flow control techniques and the aspects that take part in the recovery of messages.

3.2.1 Flow Control Techniques

The report buffer will overflow when the rate at which the messages become stable is smaller than the rate at which the members of the group initiate the new multicast. This can occur when one or more members launch the new multicast at a very fast speed, or when a communication or a process breakdown occurs. Thus, the principal condition to reduce or avoid the possibility of a report buffer overflow is to limit the number of messages from group members that will receive the new request.

S. Mishra and Lei Wu propose [11] two generic flow control styles: *Conservative Flow Control* and *Optimistic Flow Control*. The goal of the conservative flow-control techniques is to prevent the buffer overflow. In these techniques, flow control decisions are made by each group member by using information on the locally launched multicasts. Since each member of the group has precise information about what messages it has multicast, these flow control decisions are based on precise information. Consequently, the buffer overflow can be completely prevented. In these techniques, each group member is restricted to receive new messages from application clients when the number of locally originated multicasts (not still known to be stable) is a certain constant *J*. With this restriction, each group member can have at most *J* unstable locally originated updates at any time. Buffer overflow is prevented by ensuring that each group member reserves enough buffer space. The conservative flow-control techniques are used in Delta-4 [12]; the overflow avoidance techniques of the Pinwheel protocol are conservative techniques [13].

The conservative techniques are designed to prevent overflow in the worst cases; the *optimistic* techniques are designed to reduce the possibility of buffer overflow by posing fewer restrictions on group members. In these techniques, flow control decisions are taken by each group member using some global information. In an asynchronous multicast protocol, group members may not always have the most recent global information. Consequently, these decisions are based on approximate information. As a result, buffers may overflow in some very unlikely cases. These techniques also provide mechanisms to tolerate overflow, ensuring correctness and progress of the multicast protocol. Optimistic techniques are used in Horus [8], Amoeba [5] and Totem [3].

3.2.2 Loss, and Recovery of Messages.

The services of the messages control layer include: the detection of loss of messages, and the recovery of them. All these services are very important to obtain a reliable group communication system. Almost all the systems studied in this work provide reliable communication.

3.3 The Messages Ordering Layer

The main task of the message ordering layer is to provide mechanisms that ensure the same order on the delivery of messages to the group. Another issue that arises in this layer is to provide services that assure the stability property (delivery and response semantics).

3.3.1 Ordering

Roughly speaking, there are four possible *orderings*: no ordering, *FIFO* ordering, *causa*l ordering, and *total* ordering. Group communication without an order among messages is easiest to implement but unfortunately makes programming for application builders harder. *FIFO ordering* guarantees that all messages from a member are delivered in the order in which they were sent [5]. A FIFO order is appropriate when the context of a message *m* only consists of the messages emitted previously by the same transmitter of *m*. Nevertheless a message also can depend on messages sent by other transmitters. In this case, the order of delivery guaranteed by a FIFO transmission is not sufficient. *Causal* ordering guarantees that if the send events for two messages are ordered by causality and the messages have a common destination, then the messages are delivered to that destination in the causal order of their send events [2].

The Causal ordering provides a built-in message synchronization and reduces the non-determinism in a distributed computation. Causal ordering provides an equivalent of the FIFO channel property at a global (communication network) level where sequential send of messages along a communication channel is replaced by causally related send of messages going to the same destination over a communication network. Note that the causal ordering is strictly stronger than the FIFO ordering [14]. The concept of causal ordering is of considerable interest to the design of distributed systems and can be found in applications of several domains such as teleconferencing [16], multimedia systems [17], mobile computing [18], and security domains [6].

In **total** ordering, each member receives all messages in the same order. This ordering is stronger than causal orderings; it simplifies the application programming by reducing the risk of inconsistency when replicated data are updated and by resolving the contention for shared resources within the system, such as the claiming of locks [3]. The main disadvantage that it presents is that when forcing the synchronization at delivery of messages, the system loses a high degree of concurrence (important characteristic of a distributed system). Examples of systems that include total ordering delivery are: Amoeba [5], Horus [8], Isis with ABCAST [2] and Totem [3].

Other types of orderings do exist. They are a combination of previous ones: for example, *FIFO-Total* ordering and *Causal -Total* ordering [21]. The last ordering is stronger than any of the other orderings, but is harder to implement.

3.3.2 Stability

Stability is a message property that determines when a message is considered successfully delivered to a group. The knowledge that a message has reached all its destinations is useful. A multicast *m* is *k-stable* when it is known that the multicast has been received at *k* destinations. When k = $|dest(m)|$ we will say that *m* is *fully stable*[2].

There are two aspects that take part in the stability: message *delivery* and *message response* semantics.

There are three common *delivery* semantics [5]: *k*-delivery, *quorum* delivery, and *atomic* delivery. With *atomic* delivery either all correct members receive the message multicast *m* or none do. Atomic delivery is the ideal semantics, but is harder to implement since processors can fail. With *k-delivery* and *quorum* delivery, a multicast is said to be successful when *k* or a majority of the current membership has received it.

Response semantics deals with what the sending member expects from the receiving process. There are three broad categories of what the sender can expect [2], [22]: no responses, *k*-responses, *all* responses.

3.4 The Group Management Layer

The group management layer includes all those functions to provide the group management (group structure, membership, detection of processor failure, etc.) it is also concerned with the logical naming.

3.4.1 Group Structures

The development of a cooperative work requires communication and organization between its members and the exterior world. A suitable organization and communication are the support for the cooperative work. In effect, the correct definition of the relations between the members of a group and external process facilitates the coordination, the negotiation and the publication of results. For these reasons we classified the group structures in:

1. *Communication (Inter-group*, *Intra-group* communication) and,
2. *Organization.*

The *inter-group* communication is the communication between external groups or processes, while *intra-group* refers to the internal communications among group members. Keeping this in mind, there are *closed* and *open* groups. A group is considered *closed* only when its members are allowed to send messages to it; otherwise, it is *open* [23].

Organization group structures include four common styles: *centralized*, *hierarchic*, *cooperative* and *hybrid*. In a *centralized* group, the decisions are concentrated in a single member of the group, named *coordinator*; the remaining members are denominated s*ubordinates*. In this scheme only a communication between the Coordinator towards the other members of the group and vice-versa exists [27]. No communication between the subordinated members is allowed. A *hierarchical* group is a structure built from multiple centralized component groups, in this scheme the communication between groups is made through the coordinator. By *cooperative* group we refer to the coordinated work of a certain number of participants, all with the same degree of authority. In this type of scheme the coordination can only be obtained by consensus. In this structure it is necessary that all the members of a group can communicate to each other. Therefore the topology more commonly found within a cooperative group is a completely interconnected topology. Nevertheless, the work group is frequently subdivided into groups, in which case the number of communication leagues is reduced. On the one hand, the hierarchic structure in a group imposes very rigid relations in the communications within the group, limiting with this the interchange of information between the members of the group. On the other hand, a cooperative structure promotes the interchange of information between the members of the group, without imposing restrictions in the cooperation styles. Either scheme can be useful (according to the application).

There is an intermediate structure of work between these two structures, which at the same time facilitates the coordination (like in the hierarchic scheme) that does not limit the information interchange. This scheme is called *hybrid* structure (Fig. 2). It divides the group into sub-groups with one of its members assuming the coordinator functions. Unlike the hierarchic scheme, the remaining members do not act like subordinates. The communication can take place among them without taking place through the coordinator.

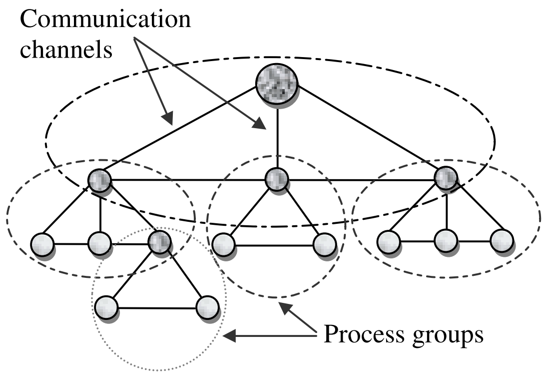

Fig. 2 Hybrid group structure

In addition, in both classifications, groups can be either *static* or *dynamic*. In dynamic groups the processes can leave or join, varying the number of members over time. In static groups a process remains in the group for all its lifetime.

ISIS Group Structures.

As shown in Fig. 3, ISIS proposes four group structures(styles) based on the group communication (inter-group and intra-group communication) [24] : *peer* group, *client-server* group, *diffusion* group and *hierarchical* group. In the *peer* group, processes cooperate as equals in order to get a task done. In the *Client-server* group, a peer group of processes act as server to a potentially large set of client processes. Clients interact with the servers in a request-reply style. Here it is possible to indicates, that ISIS presents the user with open groups, but implements it using closed groups [25]. When a process that does not belong to a group wants to communicate with it, the system either performs a join or an RPC with one of the group members. In the latter case, the member multicasts the message to the whole group. Thus, to the user this is transparent, he has the illusion to work with open groups. A *diffusion* group is a type of client-server group in which the servers multicats messages to the full set of servers and clients. A *hierarchical* group is a structure built from multiple component groups. Applications that use the hierarchical group initially contact its *root* group, but are subsequently redirected to one of the constituent subgroups.

3.4.2 Logical Naming / Addresing.

An important and complicated issue in the multi-participant communication field is the naming and logical addressing of the processes groups, the binding of those names in address, and the address into routes to the group members. With the aid of member management, one can resort to global and logical naming/addressing by using short-addresses and performing selective addressing. These guidelines simplify and improve the addressing in the communication of the multi-participant, this is typical of modern distributed systems.

The group communication implies sites and multiple participants. The group can be named logically with a single name and address; both, the sites and multiple participants, must identify to the group in their totality, without the explicit knowledge on the number and location of the final points[26].

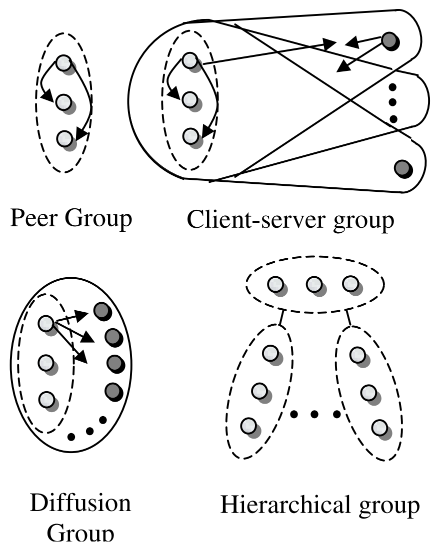

Fig. 3 ISIS group structures

Logical addressing/naming is also very important, for example in the cases that use multicast communication protocols and the network nonsupport multicast transmission. This last one can be implemented through point to point messages . In this way the user is under the impression that always works at group level and not at participant level.

3.4.3 Processor Failures Detection.

This is one of the most important services in the *group management layer,* because in fault tolerant systems it is necessary that all members of the group have the same view of which members are operational. In the presence of processor failures the system must be able to rebuild the group with the surviving members. The objective is to assure that all members in the rebuilt group receive all the messages successfully sent by any member of the original group before the failure, and that all surviving members of the rebuilt group will receive all messages successfully sent by any member of the new group after the failure. Some examples of systems with processor failures detection are Amoeba [5], Isis [2] and [1].

3.5 Services Layer.

The services layer includes all those functions that are not involved in message transmission. These services are oriented to manage the resources in a group communication system. It provides all the cooperation and coordination services in order to achieve a common final goal, for example: resources management, event management, and scheduling of activities.

4. Conclusions.

In this paper all the different issues that comprise group communication were presented in depth. Through the study of the current group communication systems, a reference design model was created to facilitate its design and its construction. According to the proposed model, systems are cast into five-layers. The layered structure of the model improves modularity.

5. References

[1] J. Chang and N. Maxemchuk, “Reliable broadcast protocols”, ACM Trans. Compt. Syst., vol. 2, No. 3, pp. 251-273, Aug. 1984.

[2] K. Birman, A. Schiper, and P. Stephenson, “Lightweight causal and atomic group multicast”, ACM Trans. Compt. Syst. Vol. 9, No. 3, pp. 272-314, Aug. 1991.

[3] Y. Amir, L. Moser, P. Melliar-Smith, D Agarwal, and P Ciarfella, “The totem single-ring ordering and membership protocol”, ACM Trans. Compt. Syst., vol 13, No 4, pp. 311-342, 1995.

[4] A. Mostefaoui and M. Raynal, “Causal multicast in overlapping groups: towards alow cost approach”, IEEE Workshop on Future Trends of Distributed Computer Systems, pp 136-142, Sept. 1993.

[5] M. Kaashoek and A. Tanenbaum, “Efficient reliable group communication for distributed systems”, Technical report, Dept. Of Math and Comp. Science, Veije Universiteit, The Netherlands.

[6] F. Schneider, “Implementing fault-tolerant services unsing the state machine spproach: A tutorial”, ACM Compt. Surveys, vol 22, No. 4, pp 299-319, Dec. 1990

[7] P. Veríssimo. “Causal Delivery Protocols in Real-time Systems: a Generic Model”, Technical Report, INESC-IST, Technical University of Lisboa.

[8] R. van Renesse, K. Birman, and S. Maffeis, “Horus: A flexible group communication system”, Commun. ACM, vol. 39, No. 4, pp 76-83, Apr. 1996.

[9] R. Gueth, J. Kris, and S. Zueger, “Broadcasting Source-Addressed Messages”, Proc. Fifth International Conference on Distributed Computing Systems, pp. 108-115, Denver, CO, 1985.

[10] L. Hughes, “A Multicast Interface for UNIX 4.3”, Software-Practice and Experience, vol 18, No. 1, pp 15-27, Jan 1988.

[11] Mishra S. and Wu L., “An Evaluation of Flow Control in Group Communication”, IEEE ACM Trans. On Networking, vol. 6, No. 5, October 1998.

[12] P. Verissimo and J. Marques, “Reliable broadcast for fault-tolerance on local computer networks”, in Porc. Ninth Symp. On Reliable Distributed Systems, Huntsville, AL, Oct. 1990, pp. 54-63.

[13] F. Cristian and S. Mishra, “The pinwheel asynchronous atomic broadcast protocols”, in Proc. Second Int. Symp. on Autonomous Decentralized Systems, Phoenix, AZ, Apr. 1995, pp. 215-221.

[14] Kshemkalyani A. and Singhal M., “Necessary and suffcient conditions on information for causal message ordering and their optimal implementation”, Distrib. Comput. 11: 91-111, 1998.

[15] Schwarz R. and Mattern F., “Detecting causal relationships in distributed computations: in search of the holy grail”, Distrib. Comput. (1994) 7:149-174.

[16] Ravindran K., Prasad b., “Communication Structures and paradigms for distributed conferencing applications”, 12th IEEE Int. Conf. On Distributed Computing Systems, May 1992.

[17] Baldoni R., Prakash R., Raynal M., Singhal M., “Efficient causally ordered communications for multimedia real-time applications”, In Proceedings of the 4th International Symposium on High Performance Distributed computing, pp 140-147, Whasington, D.C., Aug. 1995.

[18] Prakash R., Raynal M., Singhal M., “ An effient causal ordering algorithm for mobile computing environments”, Technical report, Parallel Architectures Dept., Inst. Nat. de Rech. En Inf. et en Aut., France, 1995.

[19] Deering S, “Host Extensions for IP Multicast”, Internet Request for Comments 1112, August 1989.

[20] Digital Equipment Corporation. Intel Corporation, and Xerox Corporation, “The Ethernet. A local Area Network: Data link layer and physical layer specifications”, Version 1.0, Sept. , 1980.

[21] Mullender S., “Distributed Operating Systems”, 2nd ed., ed. S. Mullender, Addison-Wesley, pp 113-124, 1994.

[22] Hughes L., “Multicast Response Handling Taxonomy”, Computer Communications, Vol. 12, No. 1, pp. 39-46, Feb., 1989.

[23] Liang L., Chanson S. T., and Neufeld G. W., “Process groups and group communication: Classification and Requirements”, IEEE Computer, Vol. 23, No. 2, pp. 56-68, Feb., 1990.

[24] Birman K., “The process group approach to reliable distributed computing”, Communications of the ACM, vol. 36, No. 12, pp 36-53. 1993

[25] Birman K., Joseph T., Schmuck F., “ISIS documentation: release 1”, Technical report, Cornell University, Ithaca, NY, July, 1987.

[26] Veríssimo P., Vogels W., Rodrigues L., “A Framwork for structuring group support in LSDCs”, Broadcast Technical Report, INESC-IST, Technical University of Lisboa, August, 1993.

[27] Tanembaum A., “Modern operating systems”, 2nd ed, Prentice Hall Hispanoamericana. 1993.