# Group Orientation: a Paradigm for Distributed Systems of the Nineties*

Paulo Veríssimo, Luís Rodrigues
e-mail:...paulov@inesc.pt, ler@inesc.pt

Technical University of Lisboa
INESC†

## Abstract

*Increasing use of distributed systems, with the corresponding decentralization of activities, stimulates the need for structuring those activities around groups of participants, for reasons of consistency, user-friendliness, performance and dependability. Two very diverse fields illustrate this trend: computer supported cooperative group working; distributed computer control.*

*This paper discusses ways for structuring systems and defining building blocks for group-oriented activity. It is felt that efficient abstractions for the design of highly distributed applications should be structured around concepts like object groups. Furthermore, the group concept should pervade the whole architecture, from network multicasting, to group communications and management, and fundamental synchronisation paradigms. Emerging technology will help materialize these concepts.*

## Introduction

Increasing use of distributed systems, with the corresponding decentralization of activities, stimulates the need for structuring those activities around groups of participants, for reasons of consistency, user-friendliness, performance and dependability. The concept appears intuitively in all flavors of distributed actions: when participants cooperate in an activity (eg. management of a partioned database, distributed document processing or distributed process control), compete for a given activity (eg. distributed use of a resource), or execute a replicated activity for performance or fault-tolerance reasons (eg. replicated database server, replicated actuator).

Paradigms, algorithms and technologies to assist the solution of these distributed problems have been presented in the recent years (distributed synchronization, replication and concurrency control, reliable group communication, multicast networking support). A generic systematics of group orientation in distributed systems is yet to be developed.

## Rationale for Group-orientation

The requirements of heavily distributed activities do not conform with raw interfaces traditionally supplied as "distribution support", such as "sockets" or "streams". Building blocks for group activity have been studied in the past in pioneering projects such as the V-kernel [7], ISIS at Cornell [5], the Circus project [8], and also [2,14]. They are currently the subject of great interest, illustrated by projects as the PSYNC/x-Kernel work at Univ. of Arizona [21], the European DELTA-4 project [22], the work on object groups by ANSA [4], the IBM flight control AAS [11], the work of Molina [12].

In order for applications with high levels of concurrency to be correctly designed, have acceptable performance, and remain operational for long enough, whatever distribution support environment to be conceived must combine: encapsulation, modularity and diversity; fault tolerance and timeliness; distributed algorithmics; events and state (i.e. actions and data).

Encapsulation, modularity and diversity may be provided by the combined notion of *object* and *group*. Different object groups in a system may be concerned with different activities, have different methods and properties, solve different problems in a harmonious way, allow for several domains of consistency and ordering to coexist, provide incremental levels of fault-tolerance, real-time hardness, etc.

*A version of this paper was published in the Proceedings of the 3rd IEEE Workshop on Future Trends of Distributed Computing Systems, April 1992, Taipe, Taiwan, © 1992 IEEE

†Instituto de Engenharia de Sistemas e Computadores, R. Alves Redol, 9 - 6º - 1000 Lisboa - Portugal, Tel.+351-1-3100000. Fax.+351-1-525843. This work has been supported in part by JNICT, through Programa Ciência.

It is hard to admit that there is a system not requiring any form of fault tolerance and timeliness. The more computations rely on distribution and interactiveness, the more important these attributes become. Its achievement in an incremental way is eased by the notion of groups.

An existing reluctance to having services with high quality (eg. reliable multicast) provided off-the-shelf in a system, has been related to the literal interpretation often made of the end-to-end argument [23]. In fact, the argument is against providing more functionality than needed at a given layer of a system, because this goes against optimizing efficiency, risks introducing redundant protocol actions like error recovery steps, and ultimately, goes against good sense. Nevertheless, if a class of applications requires a certain functionality (or quality of service), however complex it may be, the lower layer should provide it. It frees the user from programming it, and will probably have been optimized and widely tested when supplied with a system. In fact, a number of classes of distributed applications can be defined whose requirements are solved by a number of distributed algorithms. Then, supplying a suite of such protocols, in a way that the users not requiring them are not penalized by their existence, is in essence a correct interpretation of the end-to-end argument. This reasoning applies to all levels of the architecture, from network hardware through communications to computing.

Event-based [5,26] and state-based[13,16] computational models are sometimes put in alternative. This latter form of representing computations is more mature, based essentially on shared-memory and the handling of *data*. However, when thinking about the relative merits of either approach, several issues lead us into thinking that it is worthwhile to invest more in paradigms oriented to message-passing and groups of *processes* (or active objects):

- real-time — or *responsive* — systems deal with the environment, thus the event-to-state transformation is inevitable. On this matter, the real difference between state- and event-based systems is that the former transformation is performed in the periphery of the system in state-based systems, whereas in event-driven systems, events "travel" further inside the system before being transformed; this said, there is a wealth of such applications being better addressed in the domain of events [26];
- in highly concurrent interactive systems (eg. CSCW or DCCS as discussed ahead) message-passing (generally connected with events) is a very useful and natural paradigm, and should at least be used in combination with shared-memory (generally connected with state);
- in those systems, remote procedure call, being blocking, unilateral and asymmetric (client-to-server), has some shortcomings; it should be complemented with paradigms where groups of entities maintain multilateral, non-blocking and peer-to-peer interactions, according to well-defined sets of rules (such as conversations or casts) [1].

Practically all works cited earlier in the text have addressed a part of the problem, a few of them have tried to systematize solutions. We present next our macroscopic view of how to structure groups in distributed systems. This perspective was largely influenced by the work in DELTA-4, a project started in 1986 and ended in 1991, where the authors in cooperation with other research teams, addressed the problem of groups in the context of distributed fault-tolerance and real-time [22]. It has also benefited from the interaction the authors have maintained with some of the research teams cited.

# Structuring Group Support in Distributed Systems

The necessary building blocks for group-oriented system structuring and programming are represented in figure 1.

The notion of group pervades all layers of a distributed architecture, from multicasting communication infrastructures and group communication protocols with diverse order, agreement and synchronism properties, to group management services, such as membership, replication and cooperation management. Measures of the passage of time are also paramount, taking several flavors from timers to global time services built on top of approximately synchronized local clocks.

## System Architecture Issues

Looking at figure 1, one may wonder how do those blocks map into a real architecture. There is no general solution, but it is intuitive that such an organization is highly related with requirements such as:

- layer-lessness or layer transparency;
- interface recursivity (up- and down-calls);

[1] Engineering-wise, some of these problems have of course been solved long ago one way or the other (replicated RPC, "asynchronous" RPC or RSR, calling process fork, etc.). However, they should be properly addressed at the model level, if possible.

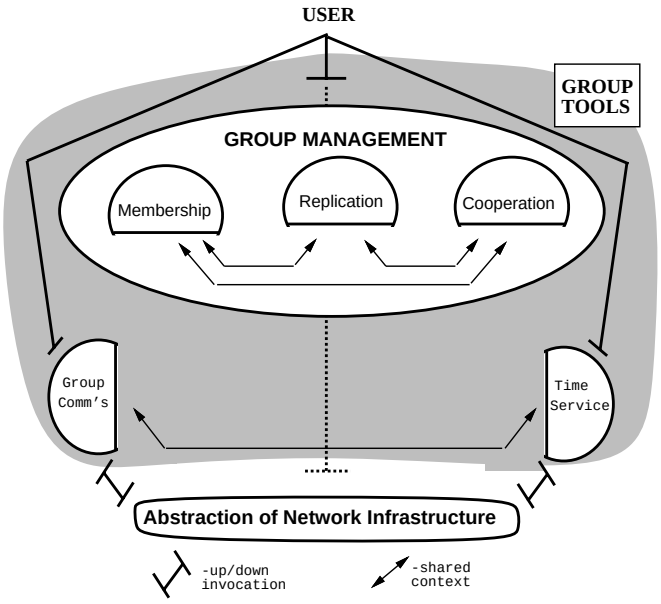


Figure 1: Group Support Building Blocks

- encapsulation using "large" objects;
- operating system support providing threads, efficient IPC, timer and buffer management, fast user-network information path, user-definable scheduling, easy embedding of external protocols.

Two alternative models for operating system support are: (a) top-level opaque interface (eg. Unix) provided by a monolithic operating system; (b) layer-transparent interface (top interface possibly Unix as well), provided by several layers on top of a micro-kernel , allowing visibility of the inner-most layers from user space, and the addition of extra functionality, like the group support protocols. The second is clearly the preferred environment for a group-oriented architecture. It is based on the micro-kernel approach, receiving large support lately.

In this paper we go as far as suggesting that the large building blocks (group management, group communications + time, network) should have a well-defined encapsulation and interface. However, inside these blocks a closer interaction may be desirable.

Certain clock synchronization protocols rely on clock-less reliable group communication. On the other hand, certain reliable group communication protocols (such as the $\Delta$-type protocols [9]) are clock-driven: they rely on the existence of the global clock provided by the time service and make heavy use of it. The group communications and time service should share context. The time service should be implemented at low level, as close to the network as possible. This reduces the errors in clock synchronization and in timestamping.

Group management is concerned with membership, replication and cooperation activities. These are broad designations which illustrate the basic activities concerned with supporting any flavor of distributed computation, in combination with, and making use of, group communication. However, practical algorithms and protocols will often combine processing steps having to do with more than one of these actions. Take the example of the controlling protocol of a replicated database: it will make use of membership management to control whether all the necessary modules are present or to know what is the majority criterium, whereas replication management will assist in maintaining consistency of the replicate set (eg. active or passive replication). In consequence, the group membership, replication and cooperation management should share context.

Nothing prevents group management and group communications from being in the same process. This may ultimately improve efficiency of their mutual interface, but they should remain as separate entities. Programming these facilities as kernel extensions is largely advisable.

## Network Infrastructure

A lot of distributed protocols in the recent years have been designed to be network-independent [17,5,12,1]. However, in trying to be generic and scalable, they do not take advantage of the existence and the emergence of network technologies such as LANs and MANs. In consequence, while these will perform well either in local or wide areas, synchronous or asynchronous environments, etc., users will be less and less prepared to pay the cost of technology independence when technology is there. Most local enterprise, institution or factory settings run over LANs. The near future will see organization-wide distributed systems over metropolitan-area networks. These are reliable both in terms of error rate and availability, display from high to very high speed and bandwidth, have broadcasting/multicasting facilities, some are capable of real-time operation and have reasonably low delivery delays, etc.

We propose that protocols are prepared to take the best advantage of the technology they have available. In the model we follow, there are a few key rules: abstract from particular networks but recognize a few *network classes* (ex. LAN, IP, MAN, etc.); define a set of properties for each class; admit different abstract networks at different layers of the infrastructure [2]; run

[2] There is an analogy with the ISO layering here, though using

parts of the protocol at different abstract network layers, by *delegation* [27].

This way, a distributed protocol like a reliable multicast protocol running over an internetwork, which it sees as a set of abstract properties (representing the internet class), may delegate part of its execution on protocols running local to the LANs, seeing a LAN-type abstract network, and obviously taking advantage of that.

We have proposed a solution to this long-lasting contradiction between the low-level approach (not-scalable) and the high-level one (not efficient). We may now go further in the requirements of the network infrastructure in order to efficiently support groups, knowing that these features may be provided by a part of the infrastructure and still play their role in group communications efficiency:

- *logical group addressing* means recipient number and location transparency, and efficient name-address translation when addressing a named group. It simplifies naming and migration of services and processes.
- hardware *selective* or *multicast* addressing on LANs is a powerful means of supporting addressing of logical groups and when desired, of selecting a sub-set of recipients in a group, i.e. the foundation for sub-group addressing;
- logical addressing combined with selective addressing yields efficient "to-whom-it-may-concern" communication, saving controller power and network bandwidth: messages are sent once, and only received by the concerned group or sub-group members[3]
- *address resolution* is another key issue: working with groups requires the capability to handle a large number of addresses efficiently. It should when possible take advantage of hardware address filtering capability, provided in some VLSI controllers.
- *topology* should ease multipoint communication; broadcast channels are best, but emerging metropolitan networks will also provide multipoint;
- *reliability* and *availability* become more important for very simple reasons: there are more players involved than in point-to-point, communication is more intensive, work is essentially interactive. In consequence, for the same individual network component reliability the probability of failure as perceived by the user is larger than in traditional applications.

different criteria.

[3] Take the example of a replicated server set, accessed by several clients. Maintenance of causal order is desired, so clients and server replicas are in a group for requests. For managing replication and preserving consistency, the reply (provided only by the coordinator) must go atomically to the client and to the other replicas. Normally this would entail the creation of a possibly large number of groups containing each client and the replicas. With selective addressing and sub-groups, that can be done by selecting inside the first and only group.

## Group Communications

Group communications services rely on the low-level network services. They ensure that a group of participants exchange messages following a set rules, without worrying about how they are secured. The semantics of group communications services can be characterized by combinations of agreement, order and synchronism properties.

These elementary properties have been characterized in the literature in the recent years. Surveys can be found in [6,22].

For example, the strongest form of agreement is unanimity, where any message delivered to a recipient, is delivered to all correct recipients. Unanimity may be unnecessary in some situations. For instance, queries to a replicate group need only reach one of replicas, or a quorum of them, it does not matter exactly which. Relaxed forms of agreement apply then, like ensuring delivery to a number of recipients N (N = 0 is the well-known datagram semantics).

In a distributed system, participants must perceive the order in which actions and events take place. The cause-effect relation is the natural ordering of events in a system. It is called a causal order. This order may be relaxed in some cases, for example to a FIFO (first-in-first-out) order, if senders are not causally related. For example, when requests from different clients to a server are commutative. On the other hand, if a participant is actively replicated, messages to the replicates should be sent in the same order. This is called a total order.

Synchronism of a group protocol can be defined by its *tightness*, or the maximum difference between instants of reception of a message at any two recipients, and its *steadiness*, or the the maximum difference between the duration of any two protocol executions [25]. According to this, protocols are loosely- or tightly-synchronous, depending on whether those differences are large or small, compared to the execution time, or even asynchronous, if they are not bounded at all. Theoretical lock-step protocols represent one end of the spectrum, being completely tight and steady [18]. Practical instantiations of clock-driven $\Delta$ protocols [9] will be tight and steady in the measure of the clock pre-

cision, which is normally a very low figure compared with the delivery time. The other end of the spectrum of synchronous protocols is represented by clock-less protocols which though not using clocks, display a bounded and known message delivery time [24]. Practically all known clock-less protocols are asynchronous (eg. [21,5]).

A group communications subsystem should have a number of services, each formed by a combination of some of the properties enumerated. For example, the combination of total order with unanimity yields what is called an atomic multicast protocol. The choice of properties by the system architect must rely on a good use of the end-to-end argument as discussed earlier in this paper. Rationale for this exercise can be found in [17,5,22,21].

### Time and Timing

The importance of time in distributed systems has been largely underestimated. Real-time systems require the ability to control duration of activities, response time, etc. This requires measuring durations and the position of an event relative to the environment. In distributed systems, the duration to measure often concerns events which have been observed by two different nodes. Scheduling of actions to occur at a given absolute time may concern several places in the system. A global timebase accessible by all nodes is thus mandatory for distributed real-time systems. It is normally achieved by each node having a local clock, and having local clocks synchronized periodically, because they naturally deviate from each other. Since systems in general are becoming more interactive (multimedia, CSCW, etc.), real-time is becoming a necessary attribute.

On the other hand, a number of distributed algorithms are based on the existence of a global time notion. We claim that even non-real-time or soft real-time applications could benefit from these algorithms and thus from the existence of a reliable timebase in the system. So, global time is a very useful building block in any distributed system.

### Group Management

Whilst group communication is concerned with allowing participants of a group to interact and establish rules for that interaction, *group management* is concerned with defining and controlling the group objectives.

Distributed activities can be reduced to combinations of three fundamental operations: replication, competition, cooperation[4]. *Competition* concerns activity directed to a group of recipients, so its rules, namely concerning ordering (eg. causal), may be implemented by the group communications services alone, whereas *replication* and *cooperation* concern activities performed **by** groups, so requiring management. Also requiring to be managed is the membership of the group, i.e. who are the participants, and if needed, what they do.

In consequence, all necessary group management protocols to control a distributed group activity lie in one of three classes: membership, cooperation and replication management protocols, and most application support protocols will use combinations of these.

An example of cooperation management is a protocol to control a task to be performed in parallel by a group of processors, or a protocol to control the simultaneous editing of a document in CSCW [3]. An example of replication management are protocols to manage replicated components, in order that they perform fault-tolerant computations, ensuring whatever actions needed, like voting, collating, etc. [22].

Membership protocols [10,15,20] know who is in and who is out, and control joins and leaves according to predefined rules. For example, detect failure and re-establish the level of replication of a group, or ensure the necessary "skills" for a partitioned cooperative activity are present in a group.

In the working field of clock-less protocols, the group membership problem is sometimes aggregated to the group monitoring problem. In short, group monitoring is concerned with assisting the correct execution of the protocol (eg. if the protocol uses acknowledges, it is necessary to have a view of the group from whom replies are expected). Group membership is on the other hand independent from group communication, and concerns the group users. While in group monitoring a totally ordered and explicit view of the protocol entities executing the communication protocol is normally necessary, in group membership the order of propagation of view changes depends on the very requirements of the users and may not be total [20]. Similarly, the users may even not need to know that there is a group (replication transparency) [22].

## Group-Oriented Programming

The concepts of group-oriented cooperation and information sharing are extremely relevant, from a number of user viewpoints. Two very diverse application fields illustrate that relevance: computer supported collabo-

[4] Although with different designations, this was established by LeLann [19].

rative group working (CSCGW); distributed computer control.

### Computer Supported Collaborative Group Work

Rules and means for collaborative group working are a necessary requirement of practically any collective activity, more so when assisted by the efficiency and cost-effectiveness of computer support. Computer supported collaborative group working (CSCGW) has been a discipline of growing interest, in the measure where widespread, ever-increasing use of communications and distributed systems make possible its application in a large number of activities, both in local and geographically broad areas. Applications for collaborative work introduce particular requirements in the underlying systems.

A major requirement for the progress of CSCGW is the existence of fundamental supporting concepts, like replicated data types or group methods, which must be merged both with user interface concepts like multiple views and dialogue encapsulation, and with the adequate architectural support (e.g. reliable and real-time group communications), to provide a unifying approach to the task of application construction in a distributed environment.

### Distributed Computer Control

Distributed computer control systems are a very challenging field which is evolving fast. The target systems encountered in the process control area are an ideal field to explore the notions of direct distribution, concurrency and groups. However, performance requirements and the importance attached to these problems, mostly money-critical, if not life-critical, deter the fast introduction of new concepts.

Distribution in computerized control has so far been almost exclusively limited to networking facilities, to down-load and up-load information (e.g. shop-floor data or CNC control programs), or to replace point-to-point cabling (e.g. centralized polling field-buses). Decentralized approaches where some node autonomy is conferred are normally specialized application-level solutions.

For an evolution to take place here, such new generation distributed computer control systems must be able to provide assurances about: correctness; dependability and real-time behavior; performance; testability. Recent architectural work (MARS [16], DELTA-4 [26]) has shown some paths to the combination of distribution, fault-tolerance and real-time, in what one could call *responsive* systems. The notion of groups is of paramount importance, to break with the need for reasoning in terms of the global system. The global approach compromises scalability and performance, and renders assertions about correctness, dependability and timeliness in a dynamic context more difficult.

## Conclusions

Issues of scale, algorithmics, development support and sheer technology barriers (bandwidth, speed, reliability) have prevented distributed computing from advancing as fast as would be desirable. The situation is changing, and it is believed that efficient abstractions for the design of distributed applications can be created if: (i) structured around concepts like object groups, group communications and management, and fundamental synchronization paradigms; (ii) taking advantage of technology, e.g. hardware multicasting and logical addressing, multimedia, high-speed, high-bandwidth, high-reliability networks.


### Acknowledgements

There a number of people whose ideas significantly influenced our work. We have discussed or worked with them during these years. The work of some of them is represented here, in that we tried to validate what we propose against their own systems: the DELTA-4 team, K. Birman (Cornell), F. Cristian (U.S. Diego), H. Kopetz (U. Vienna), R. Schlichting (U. Arizona).


## References


[1] D. Agrawal and A. El Abbadi. Efficient techniques for replicated data management. In *Proceedings of the Workshop on the Management of Replicated Data*, pages 48–52, Houston - USA, November 1990. IEEE.

[2] Mustaque Ahamad and Arthur J. Bernstein. Multicast communication in Unix 4.2bsd. In *Proceedings of the The 5th Intern. Confer. on Distributed Comp. Systems*, Denver, USA, May 1985. IEEE.

[3] P. Antunes, N. Guimarães, and R. Nunes. Extending the User Interface to the Multiuser Environment. In *Proceedings of the European Conf. on CSCW, CSCW Developers Workshop*, Amsterdam, September 1991.

[4] Architecture Projects Management, Ltd, Cambridge, UK. *The ANSA Reference Manual*, release 1.1 edition, July 1989.

[5] Kenneth P. Birman. The process group approach to reliable distributed computing. Technical report, Cornell University, Ithaca, USA, July 1991.

[6] K.P. Birman and T.A. Joseph. Reliable broadcast protocols. In Sape Mullender, editor, *Distributed Systems*. ACM Press Frontier Series, 1989.

[7] D. Cheriton and W. Zwaenepoel. Distributed process groups in the V-kernel. *ACM Tran. on Computer Systems*, 3(2), May 1985.

[8] Eric C. Cooper. Replicated distributed programs. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, Berkeley, California 94720, USA, November 1985. ACM.

[9] F. Cristian, Aghili. H., R. Strong, and D. Dolev. Atomic Broadcast: From simple message diffusion to Byzantine Agreement. In *Digest of Papers, The 15th International Symposium on Fault-Tolerant Computing*, Ann Arbor-USA, June 1985. IEEE.

[10] Flaviu Cristian. Agreeing on who is present and who is absent in a synchronous distributed system. In *Digest of Papers, The 18th International Symposium on Fault-Tolerant Computing*, Tokyo - Japan, June 1988. IEEE.

[11] Flaviu Cristian, Robert D. Dancey, and Jon Dehn. Fault-tolerance in the advanced automation system. In *Digest of Papers, The 20th International Symposium on Fault-Tolerant Computing*, Newcastle-UK, June 1990. IEEE.

[12] H. Garcia-Molina and Annemarie Spauster. Message ordering in a multicast environment. In *Proceedings of the 9th Internacional Conference on Distributed Computing Systems*, pages 354–361. IEEE, June 1989.

[13] Paulo Guedes and José Alves Marques. Extending the operating system to support an object-oriented environment. In *OOPSLA '89*, New Orleans, USA, October 1989. ACM.

[14] Larry Hughes. A multicast interface for Unix 4.3. *Software Practice and Experience*, 18(1):15–27, January 1988.

[15] H. Kopetz, G. Grunsteidl, and J. Reisinger. Fault-tolerant membership service in a synchronous distributed real-time system. In *Proceedings of the IFIP WG10.4 Int'l Working Conference on Dependable Computing for Critical Applications*, Sta Barbara - USA, August 1989.

[16] Hermann Kopetz, Andreas Damm, Christian Koza, Marco Mulazzani, Wolfgang Schwabl, Christoph Senft, and Ralph Zainlinger. Distributed Fault-Tolerant Real-Time Systems: The Mars Approach. *IEEE Micro*, pages 25–41, February 1989.

[17] R. Ladin, B. Liskov, and L. Shrira. Lazy replication: Exploiting the semantics of distributed services. In *Proceedings of the Workshop on the Management of Replicated Data*, pages 31–34, Houston - USA, November 1990. IEEE.

[18] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Prog. Lang. and Systems*, 4(3), July 1982.

[19] G. Le Lann. An analysis of different approaches to distributed computing. In *Proceedings of the 1st International Conference on Distributed Computing Systems*, Alabama-USA, 1979. IEEE.

[20] Shivakant Mishra, Larry L. Peterson, and Richard D. Schlichting. A membership protocol based on partial order. In *Proceedings of the 2nd Intl. Working Conf. on Dependable Computing for Critical Applications*, pages 1–18, Tucson, AZ 85721, USA, February 1991. IFIP WG10.4.

[21] Larry L. Peterson, Nick C. Buchholdz, and Richard D. Schlichting. Preserving and Using Context Information in Interprocess Communication. *ACM Transactions on Computer Systems*, 7(3), August 1989.

[22] D. Powell, editor. *Delta-4 - A Generic Architecture for Dependable Distributed Computing*. ESPRIT Research Reports. Springer Verlag, November 1991.

[23] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4), November 1984.

[24] P. Veríssimo and José A. Marques. Reliable broadcast for fault-tolerance on local computer networks. In *Proceedings of the Ninth Symposium on Reliable Distributed Systems*, Huntsville, Alabama-USA, October 1990. IEEE. Also as INESC AR/24-90.

[25] Paulo Veríssimo. Real-time data management with clockless reliable broadcast protocols. In *Proceedings of the Workshop on the Management of Replicated Data*, Houston, Texas-USA, November 1990. IEEE. also as INESC AR/25-90.

[26] Paulo Veríssimo, P. Barrett, P. Bond, A. Hilborne, L. Rodrigues, and D. Seaton. The extra performance architecture (xpa). In D. Powell, editor, *Delta-4 - A Generic Architecture for Dependable Distributed Computing*, ESPRIT Research Reports. Springer Verlag, November 1991.

[27] Werner Vogels and Paulo Veríssimo. Supporting process groups in internetworks with lightweight reliable multicast protocols. In *Proceedings of the ERCIM Workshop on Distributed Systems*, Lisboa, Portugal, November 1991. Also AR/51-91.