A Framework for Structuring Group Support in LSDCs

Paulo Veríssimo
paulov@inesc.pt
Tech. Univ. of Lisboa
IST - INESC *

Werner Vogels
werner@inesc.pt
INESC

Luís Rodrigues
ler@inesc.pt
Tech. Univ. of Lisboa
IST - INESC

Abstract

The development of distributed systems in the next few years will most probably be centered around: improvement of facilities for application development and execution control; development of new distributed applications; integration of new technologies to support the first two. User requirements, which are the strongest push for such development, stimulate: the improvement of application development and administration/diagnostic tools; and the incorporation of innovative technologies to overcome present shortcomings, that are known to stall or even prevent the satisfaction of some of those user requirements.

This paper discusses a framework for group-oriented system structuring and programming in large-scale systems. By looking into a few distribution support aspects, it motivates the utility of groups at all system levels. It advances structuring principles and points to the emerging technologies that may support them. The paper focuses, among those, on large-scale internetworking and multiparticipant communication. It discusses how, in the authors' opinion, multipoint (group) communication, logical addressing, membership management, and topology protocols may be used to implement efficient and robust infrastructures for large-scale distribution.

1 Introduction

The development of distributed systems in the next few years will most probably be centered around: improvement of facilities for application development and execution control; introduction of new distributed services; integration of new technologies to support the first two. One of the reasons why is the strong push for distribution represented by *user requirements*. These stimulate research and industry since they imply a market, and in consequence, one may expect: firstly, the drastic improvement of application development and administration/diagnostic tools, shortening the development time and allowing a better run-time control and robustness; secondly, the incorporation of innovative technologies to overcome present shortcomings, that are known to stall or even prevent the satisfaction of some of those user requirements.

This paper looks into the distribution support aspects: which are the structuring principles and the technologies that will change or that are already changing the face of distributed systems.

Classical systems are characterised, or at least have been regarded by researchers and engineers, as: being asynchronous and thus having poor timeliness; having low communication bandwidth and high error rates; suffering from unreliability and partitions; having mainly stream-oriented, instead of transaction-oriented, communication protocols; lacking multipoint support, like multicast networks and protocols; having security loopholes. Under such a framework, it is hard to visualise how advanced applications — such as real-time on-line transaction-processing, cooperative group work, teleconferencing, distributed coarse-grained parallelism, etc. — may proliferate.

However, a few technology advances of the present may have an important impact on the architecture and functionality of distributed systems: 100 Mb/s cheap copper FDDI (CDDI); low-cost telephone-cabling Ethernet; high-speed local computer links (eg. HIPPI); ATM networks and MANs (Metropolitan Area Networks); international very high speed/throughput data highways; digital cellular networks; global positioning (satellite) systems (GPS); multimedia components.

---

*Instituto de Engenharia de Sistemas e Computadores, R. Alves Redol, 9 - 6° - 1000 Lisboa - Portugal, Tel.+351-1-3100281. This work has been supported in part by the CEC, through Esprit Projects BROADCAST and DINAS, and JNICT, through Programs Ciência and STRIDE. This paper is an expanded and revised version (with different title) of a paper to appear in Procs. of 4th IEEE Workshop on Future Trends in DCS, Lisboa-Portugal, September 1993.

We focus on internetworking and interprocess communication aspects, since they are of utmost importance from a distributed systems viewpoint. Interprocess communication in the large is the basis for the formation of large scale distributed systems.

With the help of technology, multipoint communication, logical and global addressing, group management, and topology protocols, may be used to implement efficient and robust infrastructures.

The paper continues by discussing the factors of change in section 2. This is followed by the main topic, in section 3: the presentation of several issues of structure in future distributed systems, some made possible by emerging technologies, as discussed in section 4.

2 Factors of change

Distributed systems today, looked from afar, are best represented by an enormous number of organisation sites, having access to the several interconnected public networks worldwide, and thus being able to maintain some interchange and/or cooperation. A closer look into the inside of organisations, shows machines normally plugged in local area networks. The technological potential of LANs is high: high bandwidth, low and known error rates, multicast, time boundedness, reliability vis-a-vis partitions. However, algorithms and protocols often disregard them, in favour of homogeneous connectivity via the internetworks, which display weaker attributes: low bandwidth, higher and unpredictable error rates, point-to-point, asynchrony, partition-prone.

Users have been presented with ever-increasing power and functionality in their local machines. RISC MIPs in "normal" workstations are now rating beyond the hundred. They have already experienced successful high-performance distributed applications on a LAN scope. They expect to maintain and improve this status-quo, over the "global network", that is, to cooperate with geographically separated participants and to access remotely placed services, as if they were inside their intra-organisation network.

In consequence, they demand large scale, interactivity, real-time and dependability, multimedia, parallelism, cooperation, and so forth. Presently, the simultaneous solution of most of these requirements is either impossible or requires a number of restrictive assumptions about the system structure and/or specialised technologies.

To name a few, multimedia and motion video require high bandwidth networks, not to mention CPU power. The dependability levels achievable by applications distributed over the internet are very moderate. Interactive and cooperative applications require highly responsive real-time inter-process communication, difficult to achieve with non real-time kernels and most importantly, over non real-time networks (even if LANs). Strongly consistent replicated data is a simple and obvious way of controlling data structures pertaining to distributed databases and distributed transactional systems. However, it does not perform acceptably over slow, partition-prone networks, such as the public or leased-line networks of today.

One might think that we are striving for a complete and unrealistic revolution of technology. However, the problem is one of **system structure** for support of distribution, and it is surprising how slight changes in technology may allow drastic improvements in system support for distributed applications. To name just one example, which we will develop further, the advent of GPSs is suddenly providing a synchronised global clock with high accuracy, to systems which have, so far, been considered asynchronous. The full implications of this kind of small technology steps are still to be understood.

Should technology change, and should there be the corresponding system support evolution, the way we build distributed applications may change correspondingly, to yield better, more performant, and simpler ones.

3 Issues of Structure

In this section, we discuss a few issues about structuring the operating system support to accommodate these changes.

3.1 Site - Participant duality

Let us call *participant* to the end-user of the system support. On the other hand, let us call *site* to the machine as seen from the perspective of the network: no matter how many participants it hosts, messages directed to them are funneled through this single attachment, the machine's network adapter.

There is a duality between site and participant. The idea of distributed processing is putting participants to exchange and share information. This is what is relevant at the participant or *process* level. However, when a protocol is initiated, it runs between sites of the network, i.e. at the *processor* level. Moreover, if there are more than one recipient participant in one site, only one message should arrive at the site, and then be copied to all recipients.

This fact is in favour of — and can only be achieved by — a *communication subsystem* approach for structuring the machine's networking. A *protocol server* should take care of multiplexing send requests from participants, execute protocols on their behalf, *between sites* where recipient participants exist, demultiplex and copy incoming messages to all recipients at the site. This approach has proved to be very effective in special-purpose systems [12,27]. Provided that the protocol server is placed at the correct location in the systems architecture (as a dynamic loadable module for monolithic operating systems or as a dedicated protocol server in micro-kernel based OS technology) [42], similar results are achievable in standard systems and do not have to suffer from a lack of performance as experienced by user level located services [4].

The communication subsystem approach, if materialised efficiently, has other advantages: scalability; improved failure detection; service multiplexing.

In large-scale systems, there will be a large number of participants. The trend is for this number to increase, but the number of participants per site will scale significantly faster than the number of sites involved in each application execution. The reasons for this forecast are concerned with the increasing power of sites and with the growth of distributed applications. In consequence, all that is site-level, being embedded in the communication subsystem, need not scale as much as if there were not this hierarchy.

Failure detection will be discussed in section 3.4 below. Informally, one major problem influencing the way some distributed algorithms are implemented, is that one cannot tell genuine performance failures from communication problems (such as network partitions). This has been formalised in an important paper, known as the FLP impossibility result [13], which explains the impossibility of guaranteeing reliable consensus between a group of participants, in all situations, even under very benign failure assumptions, such as crash. Site-participant separation improves the situation w.r.t. to the FLP result, in the measure where stronger assumptions can be made about the behaviour of participants, by testing them from the site-level. Additionally, communication servers (sites) may run protocols among themselves, to check whether connectivity exists, independently from the speed or state of participants.

Imagine, with the proliferation of distributed applications, that there will always be a layer between the participant and the network, containing functionality that helps it run distributed applications with other participants. These tools are becoming very popular [1,4,16,24,26,27]. In this case, the multiplexing of these services provided on a site basis, by the participants of the site, is an effective solution, performance notwithstanding. As an example, imagine several participants in a site wishing to enroll in a cooperative editing session, together with some other participants in other sites. With a communication subsystem approach, they will carry all their interactions through the same communication port or *gate*: the gate is open, allowing communication with the group of collaborative editors, when the first participant in the site joins the group. After that, subsequent participants simply attach to that gate. Incoming messages are copied to every participant attached. In consequence, there is only one protocol running per site, taking care of all the messages from or to participants in that site.

3.2 Network-independence

Since communication speed is the dominant factor in the successful application of distributed techniques, we need to give special attention to the communication framework to ensure that its overhead is kept to a minimum, for a given quality of service.

When looking at the network infrastructure of modern distributed systems we see that Local Area Networks are ubiquitous and it seems reasonable to state that the organisation-wide distributed systems of the near-future will certainly be using LAN/MAN technology. The properties of these networks are very attractive for distributed computing, as pointed out earlier in the text. In consequence, exploiting these properties in multiparticipant communication protocols will result in an important increase in performance when compared with the network-independent protocols. Indeed, users are less and less willing to pay the price for technology independence when the technology is there.

Currently, network independence means slow packet switched networks. Their protocols, albeit robust, hide all technology features offered by today's networks. The recent addition of multicast to IP is but a small sample of what can and should be done. A good example of our argument is the order obtained free in transmissions and receptions in LANs [37], which, if exploited whenever possible, obviates the need for resorting to complicated higher level ordering protocols.

To find a solution with the merits of both the low-level approach, which exploits the available technology but does not scale well, and the high-level one, which is generally applicable but inefficient, we propose a model whereby communication protocols see an abstract interface to given network classes (eg. LAN): (i) making visible relevant properties of that class; (ii) independent of particular networks of each class; (iii) providing a basic set of low-level multipeer services like address management and varying transmission reliability.

Such an **abstract network** interface can exist at different layers in the traditional communication system. One can have an abstract LAN, whose implementation on any LAN, normally around an existing raw driver, must satisfy the abstract definition. In the case of internetwork communication an abstract Internet applies, whose objective is to transport a message between an origin LAN and one or more destination LANs, making use of underlying abstract LANs. Once the message is transported to the network of destination, the system will again take advantage of the properties of the LAN to enforce efficient local delivery.

3.3 Low-level site-oriented protocols

As presented in section 3.1, the first layers in the communication support systems are concerned with site- or processor-oriented protocols. In conformance with the *end-to-end* argument we only introduce basic services at the lowest level, so that the upper layers may use these services as fundamental building blocks for constructing protocols that match their exact requirements.

The basic service available at every participating site is that of efficient *multi-destination dissemination*, with *varying reliability*. The communication subsystem provides no ordering guarantees, but has a number of useful agreement properties. Higher layers can select whether they want the data to be acknowledged, and if so, if they would be satisfied if only a quorum or a specified subset of the addressed sites would respond, allowing for early termination of the transmission.

Besides the notion of low-level delivery service, two other structuring guidelines are relevant. Firstly, the service is implemented in a single layer *on top of* the abstract network, and in *close cooperation* with it. Acknowledgements are processed at network controller interrupt time, providing the highest performance possible. Secondly, in order to fully use the features of modern computer architectures this layer is implemented as a large number of small *emitter machines* that can run in parallel.

At this low level two more services are provided: time-stamping of messages and delegated operations.

Time-stamping should be performed the nearest to the network possible, for accuracy. In consequence, if time is to be taken seriously in distributed systems, as discussed in section 3.5, this is an important structuring principle.

Delegated operations are introduced to allow higher level protocols to execute a small part of their functionality at a lower level, without bringing them down the layers. The main reason to use delegation is to reduce the *critical path* of protocol execution. The best example of this is the sending of an acknowledgement to an arriving message. The response is produced at low-level, by delegation from the higher level protocol, in parallel with the dispatching of the arrived message to the latter, an operation which is then outside of the critical execution path.

Examples of protocols using this basic service are Site Membership protocols (section 3.4) and the protocols implementing the various qualities of service from the protocols toolboxes (section 3.6).

3.4 Site and participant membership

We start by introducing the problem of *membership*. The problem of membership is the problem of knowing who or what belongs to a system or group, or is present in an activity. It has been addressed in different contexts [10,28,22,31]. In the measure that distributed applications are decentralised, dynamic, unreliable, reconfigurable, and desirably *large scale*, membership assumes an essential rôle in distributed systems. However, more important than the protocol choice are the issues of structure: what do we need to know the membership of, in a system?

Large-scale systems are composed of hundreds of thousands of nodes. On this immense cushion, the average dimension of a set of sites thought to interact through the network during a moderately long period of time is much smaller. There are many of those (still large) sets, though. Examples: the machines of an international research network; the machines of an organisation spread over different regions or countries.

Inside each of those sets, the average dimension of the sites hosting participants actually engaged in some interaction during a short period of time is much smaller than the set size, but it may be of the same order of magnitude, under some circumstances. Examples: a cooperative edition; a concurrent engineering worktable; the set of machines hosting database replicas.

Finally, for each set of participants engaged in a distributed activity, there is another important set: the set of sites hosting those participants, smaller than or equal to the participant set.

We believe it is of extreme importance to recognise these membership levels, so to speak. For example, separating site failure detection from participant failure detection is done, for the first, by communication system based connectivity testers and for the second, by user-based participant failure detection/suspicion protocols. Not doing this separation may lead into decisions concerning failure detection which are not the most adequate: if communication is the problem, the system may try to reconfigure, searching for other routes, or may try to assess how long will inaccessibility last, to check whether it is bearable by the application; if the participant is slow (rather than dead), the administration may try to do load balancing.

Existing solutions having addressed parts of the problem, the evolution of large scale distributed systems demands a comprehensive approach. Membership should thus be structured in the following way:

domain management: a domain is an organisation level entity. The domain is defined by quasi-static administration procedures, but should be available to user applications and protocols. It defines the "network" to look for in the following levels.

site failure detectors or connectivity testers: several entities in the system may perform this rôle. The basic one would consist in scanning all the sites in the domain to check which pass a given test, whose semantics may be as poor as *"looking for an ack"*, or as rich as *"looking for a reply to at most k attempts, within s milliseconds"*. One may expect failure *suspicion* [37], or unreliable failure detection [6] to be produced by network drivers, communication protocols, or higher level membership protocols.

active sites management: a global site membership protocol maintains a table of the active sites, as up-to-date as possible and/or required. The table has the size of the domain, and sites are marked *up* or *down* accordingly to their detected state [31].

participant membership management: when a distributed application is initiated and while it lasts, it is most of the times useful if not mandatory, to keep a record of the participants [28]. Examples: to trace whether all skills are still present in a cooperative activity; to check whether the majority still exists in a replicated data manager group, upon a partition.

site membership management: each site may have one or more participants in a given application. Site group membership management provides a list, or *view*, of the sites that have participants enrolled in that application. Notice that, if site membership is based on the active sites table, the update of the site group state is automatic when the table is updated.

3.5 Time and timing

The issue of time has long been overlooked in operating system technology. With the progress of distributed systems and interactive processing, time in its various facets becomes more and more necessary. The literature has plenty of algorithms using time, provided time services are shipped with systems.

The basic notion of time in a system is interval measurement, achieved with *timers*. Here, operating systems are normally lacking in two aspects: (i) guaranteed rate; (ii) efficient start-stop and suspend-resume operations. Normally, timers do not have guaranteed rate, necessary to make long real-time measurements precise. It is also infrequent to find efficient (fast) implementations of start and stop primitives, let alone suspend and resume. The latter are useful in a number of distributed algorithms.

Second to interval measurement, is the positioning of points in time and in consequence, the ordering of events. For example, determining at what time an event happened; when an action should take place; or which of two events occurred first. This requires a *clock*. In distributed systems, all sites should share that notion of absolute time, i.e., a *global* clock. The latter should not be centralised for reliability and precision reasons. This is achieved by having a set of local clocks displaying the same time and by keeping them from drifting apart, through clock synchronisation [19].

Some recent technologies already depend from and require global clock synchronisation, for example the Andrew File System [15]. The Internet provides a global clock service, through the Network Time Protocol [21]. Operating system releases such as Berkeley BSD [14] and OSF1-DCE provide a global time notion.

There are a number of problems in distributed computing that are easily solved if events and messages can be ordered [32]. Global time, though not the most efficient way of doing it, is the easiest and most faithful one. With technology advances, some steps may be taken to improve such services in large-scale systems, as will be discussed in section 4.5.

3.6 High Quality multipoint communication

In most cases protocol and application requirements can not be simply satisfied using the basic primitives offered by the multi-destination dissemination protocols described in section 3.3. To ease the construction of the often complicated interaction structures, a multiprimitive protocol toolbox should be offered. From it, the developer can freely choose to build his own communication modules and distributed algorithms [23,29], or distributed computing support [30].

The protocol toolbox consists of a number of primitives that give access to protocols implementing different qualities of communication services. These qualities of multidestination communication services are in the properties like agreement, order and synchronism. The agreement property denotes to who (Quorum or subset) and when (all or none, even if the sender fails) multidestination messages should be delivered. The order property assures in what relative order (total, atomic, causal, FIFO) the messages are indicated at the destination sites. Synchronism may be loose or tight.

The set of qualities of service offered by the toolbox protocols are chosen in such a way that they range from simple best-effort datagrams to totally ordered message exchange. The user can weigh the costs of the service offered against the quality it offers, in context with his requirements. Offering a full range of qualities of service ensures that the user is never forced to use methods that produce more functionality and thus more costs than needed to fulfill the application requirements.

There is a close interaction between the toolbox protocols and the site membership protocols to ensure that there is a consistent view of which sites are participating in the communication. Both the dissemination layer and the toolbox protocols can generate failure suspicions, triggering investigations of the possible failure of participating sites. Closely connected to this are the failure recovery protocols run by the toolbox protocols to recover from failures in often complex situations.

3.7 Distributed programming support

The goal of providing effective programming support for distributed applications has been a research issue for more than two decades. There has been a large number of publications relating work in this area, including (among many others) distributed operating systems [36], transactional systems [3], distributed languages [18], object oriented distributed platforms [34,33], and distributed shared memory systems [25]. Often limited in early systems, the support for fault-tolerance and replication has been subject to a growing interest in the last few years. Several solutions have been proposed to tackle this problem including: support for replicated objects through remote invocations (and remote function calls) (such as *troupes* [9], ARGUS [18], or DELTA-4 [27]); language approach (such as ORCA [2]); etc. However, most of these systems provide support for only a limited set of pre-defined replication strategies. The use of a problem-oriented shared memory approach [7], by allowing different replication methods to be applied to different objects, will be a key factor for efficiency in future distributed platforms. This last issue is under consideration by several projects like MUNIN [5], ROMANCE [30], or the Fragmented Object model [20].

3.8 What are groups for?

In the previous sections, we have purposely avoided as much as possible the use of the word *group*. The intention was to make it clear to the reader that groups are an effect of the necessary evolution of system structure, rather than a cause or a fashion. As a matter of fact, the concept of group constantly popped-up in our discussion of every system level, in a natural way. No wonder, since with the increasing use of distributed systems and decentralisation of activities, the concept appears intuitively in all flavours of distributed actions [39]: 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).

One often heard remark to group orientation (group communication, membership, etc) is that the notion does not scale. On the contrary, the greater the system scale, the more necessary groups are. In fact, one must look at the issue the other way round. There are several reasons for structuring computations around groups. One of them, certainly valid for large-scale systems, is to help us "find" a group of participants among hundreds of thousands, and keep them in touch during the computation. Another utility of groups resides in the paradigms supplied by group communication (eg. causal ordering, atomic delivery, etc.) [4,27,23]. The paradigms themselves may be of help in designing distributed applications, and there is no scale issue here, apart from the fact that these services help build complex and potentially large applications. However, the same or different paradigms can be enforced for different groups in parallel, and in essence the same group oriented service in a large-scale system can serve hundreds or thousands of user groups.

As said before, with very few exceptions, group dimension will not scale linearly with system dimension. We will not see groups with hundred thousand elements. We may see applications with hundred thousand participants, but this is not where we want to apply groups. Suppose we are talking of an internationally accessible multimedia biology database: we want to apply groups, most probably, not to the end-clients, but to a group of replicated representatives of the database managers, residing in strategic points in the world with high mutual connectivity, and simultaneously giving a good access coverage for the end-clients. The group will serve for: handling updates and maintaining replica consistency; managing local caches; transparently diverting access to other group elements in case of failure; handling failure detection, recovery and reinsertion, and so forth. The database users may even not see groups at all.

Using group tools, one can build distributed applications swiftly. Once in use, whether they work on small- or large-scale systems is an issue of implementation of those tools and here, in fact, technology advances may give a hand.

4 With the Support of Technology...

The scenario of **group-oriented** system support drawn in the previous section may be reached progressively. The concepts advanced here need not be taken as whole, or in rupture with traditional paradigms. The rôle of technology is paramount, and the most important impact will be in networking and interprocess communication. To finalise, we will focus on a few areas where technology may soonest be deployed: addressing; internetworking architectures; site diagnosis and membership; high speed networking; clock synchronisation; real-time and synchronism.

4.1 Multipoint communication in Internetworks

The model of multipoint channel creation and maintenance changes drastically as soon as the participants in the communication are no longer located at the same network link. The advantages of the local network technology that made it possible to create tight execution protocols are often no longer available in this setting and information from the subsystems dealing with the internetworked network operating needs to flow to the protocols module so they can dynamically adapt to changing network situations. We consider two models of interconnection, based on knowledge of the network topology and involved technology by the routing and forwarding stations.

The first model is that a collection of local area networks with a known topology, connected by routers that can run dedicated multi-destination routing algorithms. This model has the advantage that dedicated low-level, LAN-oriented channel protocols can still be used even if not all participants reside on the same subnet. Given the existing knowledge of the network topology, the low-level communication subsystem has properties that match those of the single LAN system very closely. The information about participant and site membership can easily propagate towards all relevant subnets making both the join operation, as well as the case where a non-member site wants to communicated with a group of sites, much more simpler and thus more predictable. Given the high performance properties of networks involved, it is possible to create group management sub systems that can deal with the problems of scale in this set-up.

The second model is that of a generalised internetwork environment where there is no prior knowledge about topologies and technologies involved in the communication. This model introduces more uncertainties and thus is inherently more complicated. The set-up and maintenance of the channels connecting the several sites now involve parts of a general internetworks about which no assumption can be made except that unicast communication is possible. Tunnelling and encapsulation is used to transport the multipoint messages through to their destination organisations where the are delivered using the efficient LAN oriented protocols. Basic to the multi-destination internetwork structure is the availability of multicast routing algorithms that scale well. Preference is given to algorithms like that used in Core Based Trees (centre based shorted path spanning tree per group) over those used in DVMRP (distance vector) [43]. CBT like algorithms build a dynamic infrastructure that involves less routers in the message forwarding than in the traditional set-up, thus limiting the number of stations that need to keep state about particular channel set-up for a certain group.

A hierarchical approach to the internetwork problems helps to construct the required services in a more controlled fashion. We view the internetwork world as a collection of organisations connected by undefined communication mediums. Each of these organisations has a single access point with regard to queries about group and site membership, and provides directory assistance in channel set-up to both local and remote sites.

The initial construction of channels and the connection or subscription of new participants to a certain channel is made simple by providing certain scope information at the time of initial set-up of the communication. Although we do see possibilities for the application of pure "to whom it may concern" group communication it is much more likely that groups are created with a certain predefined context. This context includes information about the possible partners in the communication, making large scale naming, addressing and routing set-up much more simpler.

It is obvious that the predictability of multi-participant internetwork communication is a very complicate issue, and needs dedicated channel management and resource reservation protocols to assist the higher level protocols in creating responsive and real-time guarantees. Current ongoing work shows that is possible to provide these guarantees even in a general internetwork environment.

4.2 Site diagnosis and membership

There are in essence three phases in the membership issue: topology exploration; diagnostic; actual membership management.

Topology exploration is a system administration issue, normally semi-automated. Each *communication scope* should perform its own exploration, with or without the help of the standard services (eg. Internet). The objective is to arrive at a graph whose nodes are the sites belonging to the scope, and whose edges are the paths interconnecting them. In a LAN, it may concern *all* sites, or alternatively, administrative procedures may explicitly `insert` or `delete` sites from a Active Sites Table (AST). At the WAN or MAN level, exploration should be limited to the paths interconnecting the clusters where *concerned sites* of a scope reside. In a similar way, one can talk about an Active Routes Table.

Diagnostic concerns the assessment of the state of the sites. Assuming sites are always plugged to a LAN, we propose diagnostic to be performed in each LAN [31]. Upon the result, the state of the Active Sites Table entries is modified to `up` or `down`. There is no problem of scale with this approach: a copy of this table is to be kept at every site in the LAN, so that collection of system state from remote points is reliable and available.

Actual site membership management is then "reduced" to forming groups of the sites which, in each LAN, have members of a high-level participant group. This can be based on the AST, through `join` and `leave` operations on one group `id`. The advantage is that, when the failure detectors operate, and mark a site `down`, that site is automatically withdrawn from all site groups it belongs to.

For small scopes, exploration, diagnostic, and membership have been aggregated in existing works [10,28, 22,31]. In large scale systems, these operations must be separated for good results, as proposed here.

4.3 Naming and addressing

A complicated issue in the field of multi-participant communication is that of naming and addressing of groups of participants, the binding of those names into addresses, and the addresses into routes to the participants. With the help of membership management, one can resort to global and logical naming/addressing, using short-addresses, and performing selective addressing. These guidelines simplify and improve addressing in multi-participant communication, typical of modern distributed systems.

The major requirement for communication that involves multiple sites and participants is that a such a group of entities can be *logically* named through a single name and address. Both should uniquely identify the group as a whole, without explicit knowledge about the number and location of the endpoints. The binding of this single identifier to the multiple destinations is done at the point that the address needs to be bound to the several routes to the destinations. This is the point in the abstract network where channels are selected to transmit the message on. The final step is to select the individual sites on the terminal LAN(s).

This binding of address into routes is done as late as possible to allow the low-level communication subsystem to maintain the routes to the destinations on a very dynamic basis. This binding is local to a single link/subnet. Because of the locality of this binding it is possible to use *short address* forms to speed the the address processing algorithms. Special protocols based on site membership management, are used to assign and maintain the addresses per subnet.

To be able to name and address arbitrary subsets of the participants in a group, methods are available to create masks for including participants in a subset or subgroup. The *selective addressing* tools play an important role in creating more dedicated forms of the communication protocols.

4.4 Impact of High-Speed networking

New classes of application requirements where short information access times and high bandwidth are key issues have driven the development of new network technologies that make very high speeds available to everybody. These new technologies will force a revolution in the area of distributed systems communication, where after the conversational and the request/response generation, a new generation of computer communication dubbed dissemination-oriented communication will emerge[8]

The dissemination communication model is similar to that of traditional information dissemination like television and radio; streams of information are generated by several sources which are disseminated to a set of subscribers. Multi-media and teleconferencing are very well served by this model as well as other time sensitive communication like the dissemination of sensor or market data.

In the dissemination model information is transported to their destinations in anticipation of the need for it. In large scale computing this approach is even more effective, since it eliminates the large numbers of request/response sequences when clients want to access new data. Using this model a client cache of the data is created, minimising the access time of the data when it is actually needed. This reduction in access delay solves one of the major bottlenecks in current distributed systems.

The actual implementation of this model closely matches that of the multi-participant communication already described in the previous parts of this paper. In cases where a channel has only a single data source and a number of listening subscribers, optimisations can be made in the area of channel and group management.

4.5 Global clock synchronisation

The current clock synchronisation technology in the Internet is the NTP, a multi-master synchronisation service: each local site clock reads from the nearest NTP master in the Internet, which supposedly has external synchronisation from a radio time source [21]. This methodology, although simple, has disadvantages, both from the point of view of accuracy and precision, and availability of the external source.

On the other hand, there are a number of internal clock synchronisation algorithms for different network types, and with or without hardware support [11,35,17]. For generic, large-scale distributed systems, we believe the time service should be built around the following: (i) external synchronisation with moderate accuracy (second); (ii) internal synchronisation with high precision (millisecond); (iii) no specialised hardware for internal synchronisation; (iv) either synchronisation type deterministic.

Satellite-based GPS *(Global Positioning System)* devices are largely available (except in war time...), and are becoming very cheap, though not enough to have one in each site. The best combination seems to be to have a GPS device per LAN or per cluster, and run efficient LAN-based clock synchronisation protocols to perform internal synchronisation, disseminating the external time with high precision. Though generic synchronisation protocols exhibit moderate precision, protocols optimised for LANs can display very good (sub-millisecond) precisions [38]. Protocols such as the one cited are deterministic, and making external synchronisation deterministic as well is a matter of redundancy in the GPS devices.

4.6 Real-time and Synchronism

One of the long-lasting assumptions about traditional large-scale distributed systems is that they are asynchronous, in the sense that there is no bound for message delivery delay. Indeed, communication is sometimes subject to delays whose order of magnitude is much larger than the normal end-to-end transmission delay. These periods are called *network partitions* and in normal conditions they cannot be differentiated from slow participants in the other end of the link.

However, one can take a constructive attitude towards these impairments, by stipulating that, after a certain interval, the service is no longer relevant (which requires appropriate measures). One of the advantages of the introduction of membership management is that automated decisions can be made about failure of those participants appearing not to respond. One can reasonably establish *criteria* for failure suspicion (also called unreliable failure detection), so that the necessary actions can be undertaken and progress can be made [28]. Furthermore, the separation of membership functions proposed in section 4.2 above allows separating a participant performance failure from a network partition, improving the reliability of failure detection. This is not enough for real-time operation, but it is a mandatory hook: *failure detection in bounded time.*

Technology has to improve in basic quality of service before real-time can currently be run on large-scale networks. However, one should not expect the quality obtained in LAN operation. Two structuring techniques should be considered in order to run synchronous protocols on large-scale networks without requiring much from technology.

The first concerns partitions and is a very simple idea: if the system behaves in such a way that, partitions still existing, one can determine an upper bound on their duration, then real-time operation is possible. These periods would be considered, not partitions, but periods when the network is *inaccessible*. A discussion about this technique, in the context of LANs, is given in [40].

The second concerns communication protocols: real-time group communication is normally clock-driven [12], meaning it is based on a global clock. *Clock-less group communication* is more adequate for real-time communication in large-scale, since it is normally more performant and more versatile, for non-hard real-time uses [41].

5 Conclusion

We have discussed a few system structuring ideas and techniques, which we believe may influence the way distributed systems operate. We hope to have made clear to the reader that groups are a common denominator of that evolution, since with the increasing use of distributed systems and decentralisation of activities, the concept appears intuitively in all flavours of distributed actions.

We have looked at the *enabling* technologies that will make evolution possible or easier. Instead of a complete and unrealistic revolution of technology, the evolution of system structure and support for distributed applications may stand on surprisingly small changes in technology. There are a few areas where technology may soon be deployed, and we have addressed them in our paper: addressing; internetworking architectures; site diagnosis and membership; high speed networking; clock synchronisation; real-time and synchronism. The interprocess communication and internetworking issues are paramount: multipoint communication, logical and global addressing, group membership management, and topology protocols, may be used to implement efficient and robust infrastructures.

Given the pressure of user requirements for multiparticipant interactive computing, these techniques have a potential for being included in operating system technology, in the measure that users perceive their utility, and technology becomes available and cheap enough.

References

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

[2] H.E. Bal, A.S. Tanenbaum, and M.F. Kaashoek. Orca: A Language for Distributed Programming. Technical Report Report IR-140, Dept. of Mathematics and Computer Science, Vrije Universiteit, December 1987.

[3] Philip A. Bernstein, Vassos Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

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

[5] John B. Carter, John K. Bennettt, and Willy Zwanepoel. Implementation and performance of Munin. In *Proceedings of the 13th ACM Symposium on Operating System Principles*, pages 152–164, October 1991.

[6] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for asynchronous systems (preliminary version). Technical report, Department of Computer Science, Cornell University, Ithaca, USA, July 1991.

[7] David R. Cheriton. Problem oriented shared memory revisited. In *Proceedings of the 5th ACM SIGOPS European workshop*, Mont Saint-Michel, France, September 1992. ACM.

[8] D.R. Cheriton. Dissemination-Oriented Communication Systems. Technical report, Computer Science Department, Stanford University, 1992.

[9] Eric C. Cooper. Replicated procedure call. In *Proceedings of the 3rd ACM symposyum on Principles of Distributed Computing*, Berkeley, CA 94720, USA, August 1984. ACM.

[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. Probabilistic Clock Synchronization. *Distributed Computing, Springer Verlag*, 1989(3), 1989.

[12] 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.

[13] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the Association for Computing Machinery*, 32(2):374–382, April 1985.

[14] Riccardo Gusella and Stefano Zatti. The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd. *IEEE Transactions on Software Engineering*, 15(7):847–853, July 1989.

[15] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[16] M.F. Kaashoek. *Group Communication in Distributed Computer Systems.* PhD thesis, Vrije Universiteit, June 1992.

[17] Hermann Kopetz and Wilhelm Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, C-36(8):933–940, August 1987.

[18] Rivka Ladin, Barbara Liskov, Liuba Shrira, and Sanjay Ghemawat. Lazy Replication: Exploiting the Semantics of Distributed Services. Technical Report MIT/LCS/TR-84, MIT Laboratory for Computer Science, 1990.

[19] L. Lamport and P. Melliar-Smith. Synchronizing Clocks in the Presence of Faults. *Journal of the ACM*, 32(1):52–78, January 1985.

[20] Mesaac Makpangou, Yvon Gourhant, Jean-Pierre Narzul, and Marc Shapiro. Fragmented objects for distributed abstractions. In *Advances in Distributed Systems*. IEEE, 1992.

[21] David Mills. Network time protocol (version 2): Specification and implementation. Technical Report RFC 1119, DARPA Network Working Group, September 1989.

[22] 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.

[23] Shrivakanth Mishra and Richard D. Schlichting. Abstractions for constructing dependable distributed systems. Technical Report TR 92-19, The University of Arizona, Departement of Computer Science, Tucson, Arizona, USA, 1992.

[24] 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.

[25] Bill Nitzberg and Virginia Lo. Distributed Shared Memory: A Survey of Issues and Algorithms. *IEEE Computer*, 24(8):52–60, August 1991.

[26] 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.

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

[28] Aleta M. Ricciardi and Kenneth P. Birman. Using process groups to implement failure detection in asynchronous environemnts. Technical Report TR 91-1188, Cornell University, Department of Computer Science, 1991.

[29] L. Rodrigues and P. Veríssimo. xAMp: a Multi-primitive Group Communications Service. In *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pages 112–121, Houston, Texas, October 1992. INESC AR/66-92.

[30] L. Rodrigues and P. Veríssimo. Replicated object management using group technology. In *Proceedings of the 4th Workshop on Future Trends of Distributed Computing Systems*, pages 54–61, Lisboa, Portugal, September 1993. Also as INESC AR/28-93.

[31] L. Rodrigues, P. Veríssimo, and J. Rufino. A low-level processor group membership protocol for LANs. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 541–550, Pittsburgh, Pennsylvania, USA, May 1993. Also as INESC AR/30-93.

[32] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. Technical report, Univ. Kaiserslautern, Kaiserslautern, Germany, February 1992.

[33] S.K. Shrivastava, G.D. Dixon, and G.D. Parrington. An Overview of the Arjuna Distributed Programming System. *IEEE Software*, January 1991.

[34] Pedro Sousa, Manuel Sequeira, André Zúquete, Paulo Ferreira, Cristina Lopes, Paulo Guedes, and José Alves Marques. The IK Distributed and Persistent Platform: Overview and Evaluation. *Usenix Computing Systems*, 6(4), Fall 1993.

[35] T. K. Srikanth and Sam Toueg. Optimal Clock Synchronization. *Journal of the Association for Computing Machinery*, 34(3):627–645, July 1987.

[36] Andrew S. Tanenbaum and Robert van Renesse. Distributed operating systems. *ACM Computing Surveys*, 17(4), December 1985.

[37] 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.

[38] P. Veríssimo and L. Rodrigues. A posteriori Agreement for Fault-tolerant Clock Synchronization on Broadcast Networks. In *Digest of Papers, The 22th International Symposium on Fault-Tolerant Computing*, Boston - USA, July 1992. INESC AR/65-92.

[39] P. Veríssimo and L. Rodrigues. Group orientation: a paradigm for distributed systems of the nineties. In *Proceedings of the 3rd IEEE Workshop on Future Trends of Distributed Computing Systems*, Taipe, Taiwan, April 1992. also as INESC AR/67-92.

[40] P. Veríssimo, J. Rufino, and L. Rodrigues. Enforcing real-time behaviour of LAN-based protocols. In *Proceedings of the 10th IFAC Workshop on Distributed Computer Control Systems*, Semmering, Austria, September 1991. IFAC.

[41] Paulo Veríssimo. Real-time Communication. In S.J. Mullender, editor, *Distributed Systems, 2nd Edition*, ACM-Press, pages 447–490. Addison-Wesley, 1993.

[42] Werner Vogels, Paulo Verissimo, and Luis Rodrigues. Requirements for high performance group support in distributed systems. In *Proceedings of the 5th ACM SIGOPS European workshop*, Mont Saint-Michel, France, September 1992. ACM. also INESC AR/29-92.

[43] D. Waitzman, C. Partridge, and S.E. Deering. Distance vector multicast routing protocol. Technical Report RFC 1075, USc Inf. S. Inst., November 1988.