THE DELTA-4 EXTRA PERFORMANCE ARCHITECTURE (XPA)

P.A. Barrett  
A.M. Hilborne  
MARI Applied Technologies Limited  
The Old Town Hall  
Gateshead NE8 1HE, UK.

P. Verissimo  
L. Rodrigues  
INESC  
Rua Alves Redol 9-2  
1000 Lisboa, Portugal.

Abstract

As part of the European Strategic Programme for Research in Information Technology (ESPRIT), the Delta-4 project is seeking to define an open, fault-tolerant, distributed computing architecture. The Delta-4 approach to fault-tolerance is based upon the replication of software components on distinct host computers using a range of different replication strategies.

This paper presents the design of an Extra Performance Architecture for Delta-4 which explicitly supports the requirements of real-time systems with respect to throughput and response. The problems of replicate divergence are discussed, and a solution based on message selection and pre-emption synchronisation messages is proposed. The paper includes a description of the ongoing implementation of such a system within the overall Delta-4 framework.

1. Introduction

Delta-4 is a collaborative project carried out within the framework of the European Strategic Programme for Research in Information Technology (ESPRIT). Its aim is the Definition and Design of an Open, Dependable, Distributed computer system architecture (hence the project's name) [1,2,3].

Dependability is defined as being a quality of delivered service such that reliance can justifiably be placed on this service [4]. It embraces the attributes of, for example, reliability, availability, maintainability, safety, integrity and security, each of which is seen as a different perception of the same attribute, which can be addressed by the same underlying support mechanisms. The Delta-4 architecture allows the user to obtain specifiable levels of dependability on a service-by-service basis without assuming any special attributes of participating hosts.

The main focus of the project from the point of view of dependability is the tolerance of (accidental) hardware faults, however the tolerance of intentional interaction faults (intrusions) and software design faults is also being addressed. Extensive validation activities are being carried out to justify the reliance being placed on the architecture.

Openness, in the context of the project, has a number of implications. Firstly, the Delta-4 architecture must be capable of co-existing with, and inter-working under, standards conforming to the Open Systems Interconnection (OSI) reference model. Delta-4 will accommodate existing proprietary computer systems, and the connection of heterogeneous equipment via one or more inter-connected local area networks. Secondly, the Delta-4 Application Support Environment (Delase) will conform to the emerging standard for a Support Environment for Open Distributed Processing (SE-ODP) proposed by the European Computer Manufacturers Association (ECMA) working group TC32/TG2. Finally, the results of the project are published.

For certain critical, real-time applications, the openness and generality which is otherwise one of the major attributes of the project, becomes a constraint. The Delta-4 Multi-point Communications System (MCS) implements a wide variety of services which superset those of OSI, and successfully offers the functionality needed for a broad spectrum of Delta-4 applications. Precisely because of its generality and openness, however, MCS cannot offer the assurances of timeliness required or the performance demanded in the real-time sector. Thus, the Delta-4 project is developing, alongside its Open Systems Architecture (OSA), an Extra Performance Architecture (XPA).

The Delta-4 XPA architecture introduces mechanisms which support explicitly the requirements of real-time systems with respect to both throughput and response. The architecture will support the real-time concepts of priorities and deadlines, and will reflect these concepts within its communications protocols. Throughout, the XPA architecture will inherit as much of the OSA architecture (both concepts and implementation) as is possible within the constraints imposed by its performance requirements.
In order to meet its performance targets, XPA will inevitably lose some of its openness and generality. The functionality of the communication system will be limited to that required to support XPA applications running under XPA-Deltase. Standards which cannot address real time needs will not be used. Collapsed layering will be applied as appropriate to the communications protocols, and fail-silent hardware will be used to avoid the need for the validation of results via voting. This loss of generality will not be global, however; a Delta-4 system will be able to contain both OSA and XPA sub-systems, with the two able to co-exist and inter-work. Similarly, although XPA will not be a fully open architecture, the rationale and results of the work will be publically available.

The project has not yet reached the stage of measuring performance (although implementation of the first full prototype will be nearing completion at the time of publication). However, a round-trip delay time (including task activation latency) for highest-precedence null-RPC (null Remote Procedure Call, i.e. without competing activity, without parameters and to an empty procedure) of the order of one to two milliseconds is the requirement of a useful range of application areas, and is the state of the art in existing non-replicated systems. Such figures provide a challenging target for the project; the cost of using RPCs must be small for it to be attractive to the designers of distributed applications [5]. This applies even where, as in XPA, support for replication is included.

In section 2 of this paper we present an overview of the existing Delta-4 Open Systems Architecture. We then describe the new Extra Performance Architecture in Section 3, and discuss implementation considerations in Section 4. Finally, Section 5 describes the conclusions drawn from the work.

2. An Overview of the Delta-4 Open Systems Architecture

A Delta-4 system consists of a number of computers (possibly heterogeneous), interconnected by a Dependable Communication System. Application programs are structured as software components distributed among the hosts of the system. Each software component may be replicated, however, copies of software components must be located on homogeneous (functionally identical) machines. Each node consists of a host computer together with a Network Attachment Controller (NAC). The NACs are specialised communications processors which together implement a dependable communication system allowing multi-point communication between replicated computational entities.

The Local Executives (LEXs) of the hosts may be heterogeneous. Each NAC has its own local executive in the form of a Real-Time Monitor (RTM), and these, too, may be heterogeneous. The distributed Delta-4 software running on each node may be classified as follows:

- The communications software which executes on the NACs. The communication system of the Delta-4 open systems architecture is called the Multipoint Communication System (MCS). MCS provides reliable multi-endpoint communication based on multi-endpoint connections [1].
- Administration software which provides management of both computational and communications elements of the system, and which executes partly on the host computers and partly on the NACs.
- The Delta-4 Application Support Environment (Deltase) provides a framework for the construction of object-oriented, dependable, distributed applications.
- The user applications software itself. This will consist of a number of software components (objects), potentially written in different languages, and communicating via Remote Service Requests (RSRs), an RPC-like mechanism for inter-object communication.

In the Delta-4 system, a number of techniques are available for providing fault tolerance [1,2]. Each involves the use of replicated software components, and they may be split into two basic groups according to whether or not they require the use of fail-silent host processors.

One technique involves the parallel operation of a number of identical copies (at least 3) of a software component, with the outputs from all components being compared, and the majority decision being used (voting, see for example [6,7]). This is the active replicates model. Comparison of "signatures" (a form of checksum) is used to validate messages. If two components produce an identical signature, MCS selects one copy of the message to forward to its (possibly replicated) destination. The active replicates model detects that a node is faulty when a message originating on that node fails to agree with the majority of its replicates, is not produced at all, or is produced in error; the latter two cases being determined through timeouts. This allows the system to tolerate fail-uncontrolled hosts, i.e. hosts which can fail in an arbitrary way.

A second technique used within Delta 4 is passive replication [3]. In this model, each software component is replicated but only a single copy is active. This component periodically copies (checkpoints) its state to the others (its backups). If the computer hosting an active software component fails, a backup is awakened and begins to execute from its most recent checkpoint. In order to detect failure, the assumption is made that
all processors are fail-silent [2,8,9] i.e. they will fail in such a manner that they will become silent and will never send out erroneous messages to processors functioning correctly.

If software components run on fail-silent nodes, active replication may be used without voting, since any message generated by a process may be assumed to be correct. As a result, the minimum replication degree is reduced to 2, the communications mechanisms required are simplified, and better performance is achieved since results may be propagated immediately they are generated, rather than being held pending the voting process.

A variant of active replication provides a further technique for error processing. In this case, all software replicates are active and execute application programs but only one designated copy (the leader) propagates output messages [10]. The fail-silent assumption ensures that these messages cannot be erroneous. This model is similar to the passive replicates model but uses additional processing to update the states of backups (or followers) rather than checkpoints. As we shall see in the following section, the leader/follower model is well suited to the requirements of the extra performance Delta-4 architecture.

3. The Delta-4 XPA Architecture

3.1 XPA Dependability Models

XPA shares the OSA computational models, and there will therefore be an XPA implementation of Deltase. DeltaSe applications are object-oriented, and the unit of replication is the object itself.

Each of the Delta-4 OSA dependability models described in Section 2.1 has advantages and disadvantages for use in an XPA environment. A benefit of active replication [1,2] is that there is no interruption in service when a failure occurs. Further, if fail-silent hosts are used, the requirement to validate outputs by voting may be avoided, the outputs of the fastest replicate being propagated to the network. On the debit side, it is essential to support active replication with some form of atomic multicasting [11]; the protocols to achieve this are necessarily complex (see Section 3.4). Further, active replicate objects must behave identically with respect to messages consumed and produced [12] (the necessary properties being conferred by the use of an appropriate applications model, by implementing deterministic objects using the State Machines model [13], or implementing replicate-deterministic objects using some form of agreement on execution path). If active replicate objects are required to respond quickly to external events through some form of pre-emption, the difficulty is compounded. Pre-emption is complex and costly to synchronise between active replicates, since each replicate must be pre-empted at exactly the same point in its processing. In practice this is likely to lead to unacceptably large maximum pre-emption times.

The main OSA Passive replication model [3] suffers from no such problem with pre-emption; since only one replicate is active at any time and all external consequences of its activity are accompanied by capturing a checkpoint, pre-emption may occur at any time. Further, this passive replication requires relatively simple communications support and, a major advantage for many applications, does not require objects to be deterministic. Since only one replicate is active, processing requirements are minimised; checkpoint capture will generally use less resource than replicated execution. Passive replication does, however, require the use of fail-silent hosts, and when an active replicate fails there is a delay in the provision of service whilst recovery and re-execution is carried out. Such a delay may not be compatible with the achievement of demanding real-time deadlines which must be met in spite of failure.

In order to counter these problems and attempt to provide a dependability model appropriate for all applications objects, a further model has been introduced for use in the XPA architecture (where it will complement, rather than supersede, the existing models). It incorporates attributes from both existing models, and has been dubbed the leader/follower model. This model will also be accommodated into the OSA architecture.

3.2 The Leader/Follower Model of Replication

In the leader/follower model of replication, all copies of an object are active, in that they all execute the same code. One copy is designated the leader, however, and is responsible for taking all decisions which affect replicate determinism; such decisions are propagated from leader to followers via synchronisation messages. System nodes are assumed to be fail-silent, thus output message validation is not required; messages may be sent by the leading copy of an object immediately they are generated and when the followers generate the same messages, they will be discarded automatically by the communications system.

Two forms of synchronisation message are used; input synchronisation messages and pre-emption synchronisation messages:

i) Input Synchronisation Messages

It is necessary that notions of the precedence of a global computation over another may be propagated from a client, via the communications system, to its servers. Thus, objects must be able to consume input
messages in an order which respects their instantaneous precedence. However, all copies of an object must consume the same messages in exactly the same order, otherwise their paths may diverge and replicate determinism will be lost (see, for example [14,15]). Therefore, when the leader selects a message (according to a precedence rule applied at some instant to its local set of outstanding messages), it also constructs a synchronisation message containing the identity of that input message and sends this to its followers. The followers consume messages in the order dictated by their leader, and replicate determinism is preserved. (Note that in practice this mechanism is embedded in the communications protocols and made totally transparent to the applications programmer.)

ii) Pre-emption Synchronisation Messages

Certain objects must be constructed to be pre-emptable very quickly should certain events (such as alarm conditions) occur. As with active replication, pre-emption can result in replicate non-determinism unless every copy of an object is pre-empted at exactly the same point in its processing. The leader-follower model incorporates a pre-emption synchronisation mechanism which imposes a very small overhead to ensure that replicate determinism is preserved.

This mechanism makes use of the concept of a pre-emption point; this is a predefined point in its processing at which an object may be pre-empted. Since pre-emption point code must be executed more often than the maximum allowable pre-emption delay, it is essential that the normal, non-pre-empting path through this code be efficient. Pre-emption synchronisation is achieved as follows:

Each time the leader reaches a pre-emption point, a counter is incremented. (Note that there is one counter per object replicate, not one per pre-emption point). When a message arrives at the leader, a check is made to determine whether this message requires the leader to be pre-empted. If so, the pre-emption point at which this will take place is selected (the current counter value plus 1 represents the next pre-emption point) and assigned, and a synchronisation message containing this value and identifying the message is constructed and despatched to the followers. On arriving at the assigned pre-emption point (i.e. when their counters match the assigned value), each replicate begins to process the pre-emption. Since pre-emption point code must be executed more often than the maximum allowable pre-emption delay, it is essential that the normal, non-pre-empting path through this code be efficient.

In order for this mechanism to work, the followers must always be executing at least one step behind the leader, where a step constitutes the receipt of a synchronisation message due either to a pre-emption, or to the consumption of an input message by the leader. To avoid followers falling too far behind their leader (a figure which is determined by the time permitted for recovery following leader failure), dummy synchronisation messages (which also double as "I'm alive" messages) may be sent periodically by the leader.

3.3 The XPA Approach to Real-Time

Within XPA we wish to allow both hard and soft real-time behaviour to co-exist. That is, an XPA system should:

- Provide support mechanisms which allow a system designer to assert that a defined subset of required activity is assured to meet deadlines.

- Offer the designer the possibility of defining another subset exhibiting best effort behaviour, but without compromising the deadline assurance of the first subset.

- Offer the opportunity for any spare resource to be occupied with background activity, as long as this does not compromise any other behaviour.

Since it is required to support both hard and soft deadline computations, XPA necessarily takes a dynamic approach to local scheduling, using the instantaneous precedence of computation. The term precedence is generic, meaning any variety of ordering method. Earliest deadline establishes one sort of precedence order, highest criticality another. XPA precedence is a combination of the two: the coarse precedence of groups of components is established through their levels of criticality, and within a criticality level, the precedence of individual components is established by the component deadlines. This combination does not suffer from the disadvantages of either pure criticality or pure deadline methods.

Precedence does not apply just to individual components, but is made to apply to the whole of a distributed computation. When a deadline computation arises, all concerned components must be executed with a particular precedence. Precedence is propagated with a request for service from component to component; all resulting subordinate computation inherits the precedence of its parent. Without such a mechanism, the ability to predict behaviour (to be able to assert that certain computation can be rendered hard) would be lost, and a high-precedence component fulfilling an urgent requirement would effectively be barred from raising a request for service on a server of lower precedence.

Thus, in an input queue of requests to a component, higher precedence requests must be able to overtake requests of lower precedence, otherwise the "hard"
characteristics required for the highest precedence activity will be compromised. The issues surrounding precedence inheritance are discussed in [16,17].

To support Delta-4 dependability models such overtaking and pre-emption must be performed deterministically, since replicates must be certain to receive requests in the same order. One effect of this is to render lower-level methods of achieving message ordering irrelevant; some form of negotiation technique must be used. In XPA, the replication models supported for high-precedence computation are limited to those where such negotiation is inexpensive.

3.4 Communications Mechanisms for XPA

The Delta-4 communications system aims to support distributed fault-tolerance through the replication of objects on host computers inter-connected via local area networks. The system features logical communication, and provides support for the concept of groups. (In principle, a group is simply an arbitrary set of objects grouped together for communications purposes. In general, a group will comprise all replicates of a given object, however this does not necessarily hold for all models of replication.)

The current OSA offers a multicasting service at the data link layer (dubbed AMp [18]), complemented by an ISO-based protocol stack. Mechanisms at the session layer provide for communications multiplexing and error processing through the management of multi-endpoint associations.

Since AMp is used in the OSA primarily to support an actively replicated state machine type of computation [13], the multicasting service provides a single atomic quality of service (hence the name AMp - Atomic Multicast Protocol) with unanimous agreement and consistent ordering of messages. High-level addressing and communications concepts (such as the differentiation of clients from servers, or the mechanisms required to support passive replication) are provided by the session layer protocols.

The aim of XPA is to take advantage of certain features of the dependability models used within it to minimise the functionality demanded of the communications system, and thus to maximise its efficiency. The result is the introduction of new services within an extended AMp (xAMp), which now provides a number of different qualities of service ranging from atomic multicast through reliable multicast to unreliable datagram. To see how advantage may be taken of this extended functionality, consider the leader/follower model of replication. One of the advantages of this model is that it may be supported by relatively straightforward communications mechanisms; since the order of message consumption is dictated by the leader, consistent order of transmission is no longer necessary. Hence, an atomic service is not required, and may be replaced by a simpler and more efficient alternative such as the reliable service of xAMp.

We define reliable multicast to be such that, if a message arrives at one correct destination, then it will arrive at all other correct destinations within a bounded time. The normal mechanism is one of acknowledge and retry (a duplicate message detection and deletion mechanism can be used to ensure that no destination receives the message more than once). In addition, advantage is taken of the fact that message receipt is notified between replicates (as a side-effect of the synchronisation mechanism) to enable replicates to identify the existence of, and a source for, any unreceived messages. Reliable multicasting cannot provide for consistent message ordering at replicate destinations, but since objects must be able to consume messages in precedence rather than temporal order this consistency must be provided by the higher-level synchronisation mechanism described above, so this does not matter. The use of the reliable multicast service of xAMp within XPA will have a number of benefits for the performance of the system:

- The total number of message frames being sent should be reduced in comparison with a system based purely on the full atomic service. Although synchronisation messages are required to ensure that all replicates consume messages in the same order, the commit phase frames of an atomic service are unnecessary.

- The effective message propagation delay is reduced; since messages may be made available to leader objects immediately they arrive on the host. (Figure 1 compares the effective message propagation delay of the reliable service of xAMp with that of its atomic service; although the figure shows propagation delays in the absence of transmission failure and retry, it may easily be seen that the principle of reduced propagation delay holds even where re-tries are required.)

- The reliable multicast service of xAMp is simpler than its atomic service, thus under normal circumstances (and for many failure conditions) the communications system should be inherently more performant.

It should be noted that the functionality provided by xAMp (being, in effect, a superset of that provided by AMp) is capable of supporting both active and passive replicates in addition to leader/follower. Thus, such objects may easily be incorporated into XPA applications should their characteristics be considered appropriate. (For example, active replication may be used
Figure 1. Comparison of the Effective Propagation Delays of Atomic and Reliable mechanisms of xAMp (Assuming no re-tries necessary in either case).

- Atomic - Messages are received by all non-faulty receivers, or by none. Consistent causal ordering of messages is ensured.
- Reliable - Messages are received by all non-faulty receivers, or by none. No guarantee of causal ordering is made. Message delivery is loosely synchronised; messages are delivered to recipients immediately they arrive on the host, and error and failure detection are performed in the background. Inaccessible recipients must be considered to have failed to maintain the properties of unanimity and real-time termination.
- AtLeastN - Messages are transmitted and acknowledged. Message 'commit' occurs when at least N recipients have acknowledged receipt, thus messages are received by at least N recipients or by none. When N is zero, a logical datagram is obtained. If there is only a single addressee, the datagram is unicast.
- AtLeastTo - Messages are transmitted and acknowledged. Message 'commit' occurs when a specified set of recipients have acknowledged receipt, thus guaranteeing that at least those recipients receive the message correctly.

In general, the communications protocols will run on a special-purpose Network Attachment Controller (NAC) board, however it is also possible to use standard network controllers and to execute the protocols on the hosts themselves.
The Group Manager

The Group Manager is actually a distributed object, represented by local pseudo-objects on every node of an XPA system. These pseudo-objects each behave as if they are the Group Manager in their interactions with other objects on their node, but interwork in a non-object-like manner to preserve the consistency of this illusion. The Group Manager is concerned with the management of groups of objects, and with the support of the distribution of such groups.

The Group Manager maintains its own local view of the state of the system. This view includes information on which nodes currently form part of the system, which objects currently exist within the system and which groups are associated with those objects, the nature of that association, and the identity and status of each replicate of an object (e.g., leader, first follower, passive backup etc.).

The Group Manager also incorporates knowledge of the different modes of replication (active, passive, leader/follower etc.), and is able to provide, transparently to the objects themselves, an appropriate level of support based upon the services of the communications layer.

The Interface Manager

The Interface Manager takes the form of a library which is linked into each applications object. Its function is to interface objects with the message queues which form their access point to the communications system. During a typical RPC call, the Interface Manager will firstly pack the RPC details and parameters into an appropriate format, then enter this as a message in the object's transmission queue. When the response is eventually placed (by the Group Manager) into the receive queue, the Interface Manager will extract the response message, unpack it, associate it with the appropriate request (thus identifying the thread to which the result must be returned), then finally return the result via a procedure return. This process is, of course, completely transparent to the applications programmer.

- Message Queues.

The interface between the Group Manager and the Interface Managers on the local node takes the form of a number of message queues. Message buffers will be drawn from a pool shared between the Group manager and the Interface Managers, thus eliminating the need to copy messages during their transition from one to the other.

5. Conclusion

The Delta-4 Open Systems Architecture was designed to meet the requirements involved in the implementation of dependable open systems. A system which aims to support the throughput and response time characteristics of real-time systems requires a more specialised architecture, and Delta-4 XPA is an attempt to provide such an architecture. The architecture is structured similarly to the Delta-4 OSA, but sacrifices openness in order to simplify communications protocols and decrease both communications and computational overheads.

The major OSA dependability models (active and passive replicates) were found not to be ideally suited to all the requirements of real-time systems. The leader/follower model described in Section 3 was shown to address many of the shortcomings of the existing models, while introducing few drawbacks of its own.

The problems inherent in maintaining determinacy between object replicates was shown to be central to the design of the XPA architecture. The leader/follower model allows message selection to be ordered solely according to the precedence of the pending messages. The model also allows for pre-emption to occur deterministically between replicates. Further, maximum pre-emption latency may be defined by the system designer according to the requirements of particular applications. Such features are essential for an architecture intended for the implementation of dependable, real-time applications.

In conclusion, the Extra Performance Architecture of Delta-4 has been designed for the production of real-time applications which demand a highly-responsive,
highly-dependable system in which issues of distribution and fault-tolerance are largely transparent to the applications designer.

Acknowledgements

The work of the Delta-4 project is carried out by a consortium of thirteen partners: Bull SA (F), Credit Agricole (F), Ferranti Computer Systems Ltd. (UK), IEL-CNR (I), IITB-Fraunhofer (D), INESC (P), LAAS-CNRS (F), LGI-IMAG (F), MARI Applied Technologies Ltd. (UK), NCSR (UK), Renault (F), Sema Group (F), and the University of Newcastle upon Tyne (UK).

The authors would particularly like to thank D.Drackley, D.Powell, Prof. S.K.Shrivastava and all other members of the Delta-4 project. We would also like to thank the Commission of the European Communities, Information Technologies and Telecommunications Task Force for providing funding in support of this work under the ESPRIT programme.

References