Chapter 3

Comparison with other Approaches

The purpose of this chapter is to position the Delta-4 architecture with respect to commercially recognised and available techniques. The comparison is made with respect to the Delta-4 application domains, characterized as large-scale, distributed information processing systems, and summarised in 1. It would not be appropriate here to make comparisons or draw conclusions about the respective merits of these architectures in other domains, e.g., transaction processing. Rather, the objective is to compare Delta-4 with alternative approaches to the design of distributed systems that are also required to be dependable, or exhibit real-time characteristics, or both. Before making this comparison, a summary of currently used commercial fault-tolerance techniques is included for reference. No similar analysis of real-time systems is offered since conventional stand-alone real-time operating systems are well-known and there are no known actual examples or de facto standards for distributed real-time systems.

3.1. Commercial Fault-Tolerance Techniques

The concepts behind the achievement of fault-tolerance are described in section §4.5.2, together with different techniques that may be applied. These techniques provide a structure for the general discussion below of commercial solutions based on their combination.

The general issue of non-disruptive on-line repair is referred to in all cases examined.

The assumption of *fail-silent* components, i.e., components that fail only by stopping delivery of outputs, is necessary in some approaches. The *assumption* of fail-silence is part of the model; vendors must provide mechanisms to *support* this assumption to the degree demanded by the market for their product. Thus, these mechanisms vary widely, from highly justifiable techniques such as sophisticated lock-step synchronized hardware with self-checking voting logic to address the most critical markets, to use of self-analysis or watchdog software running in largely fail-uncontrolled hardware.

3.1.1. Backward Error Recovery

In this approach, faults are tolerated by use of regular state-capture to mirrored recovery caches commonly accessible to two fail-silent subsystems, one of which is active until failure. The combination of active fail-silent subsystem and mirrored recovery cache permits the other fail-silent subsystem to discover a correct state from which to continue execution in the event of failure of the first, even during state capture.

One issue of importance is the degree of coupling which permits the cache accessibility with minimal overhead, whilst avoiding execution disruption during on-line repair activity.

3.1.2. Error Compensation Techniques

3.1.2.1. Error Compensation by Majority Voting. Practical commercial systems using majority voting techniques are based on *triple modular redundancy*; faults are tolerated by locally triplicating subsystems. Thus, one node contains tripled CPUs, memories, and so on. To permit redundancy to be restored on-line, arrangements are made for physical replacement of faulty components and to cause the state of the restored components to converge with the state of the remaining system without disrupting service provision. The system MTBF is improved by reducing the granularity of containment of failure; to this end, a dependable power-supply can be constructed as a separate subsystem. Several proprietary means of doing this and allowing on-line repair exist.

3.1.2.2. Error Compensation by Redundant Fail-silent components. If a component is arranged to fail only by stopping delivery of outputs, then the error can be compensated by providing more than one such component. A mechanism is required which transforms the several (necessarily equivalent) outputs from all non-failed components so that they are seen as a single output. Sometimes this is done by arranging one component to be a "Master" until it fails, when another takes over.

A typical implementation of this approach is to tolerate faults by locally duplicating subsystems that are themselves constructed, through duplicating cross-checked components, to exhibit fail-silence. Thus, one node contains quadrupled CPUs, memories, and so on. In the event of failure, mechanisms are provided to restore redundancy on-line without disrupting execution of the remaining system. Again, a dependable power-supply is provided.

3.1.2.3. Replica Group Determinism. With either of the above approaches, replica group determinism must be assured, a property which products based on these approaches confer by use of one of the following techniques:

3.1.2.3.1. Microsynchronous Execution. In this approach, replica hardware may be sufficiently coupled to assure replica-determinism without any special software provision, by executing the entire ensemble in a lock-step manner by synchronising all replicated components to a common dependable clock subsystem. Proprietary solutions are used to resolve such close-coupling issues with the need to be sufficiently decoupled to allow redundancy to be restored on-line.

3.1.2.3.2. Macrosynchronous Execution. An alternative to the lock-step synchronization solution to replica group determinism without special software provision is the use of deterministic resynchronization. Each CPU may have an independent and therefore asynchronous clock and hence avoid the need for a dependable clock subsystem. The equivalent of lock-step synchronism can be achieved by resynchronising whenever necessary, for voting and other events. Necessary resynchronization events include interrupts, and voting events such as reads or writes to an external address space. Other resynchronization events may also form part of a proprietary solution. For example, use of a maximum time elapsing since the previous resynchronization event offers a simple resynchronization mechanism where the asynchronous clocks are sufficiently close in frequency to only drift slowly past each other.

3.2. Comparison of Approaches by Concept/Technique

In comparing these architectures with Delta-4, the lowest level of interest is the basis on which dependability is achieved.

3.2.1. Stand-Alone vs. Distributed

A first observation is that, in all the cases examined above, the intention has been to construct "stand-alone" equipment whose external appearance is of a dependable single node. Delta-4, however, uses a distributed software solution, with assistance from a minimum amount of specialized hardware, the fail-silent NACs. At this level, the above solutions may be viewed as competing with that offered by Delta-4. Ignoring any benefits *per se* arising from inherent distribution, Delta-4 provides a more economic solution when based on standard mass-produced nodes, and a more flexible solution arising from both the incremental nature of its dependability and its ability to accommodate nodes with special properties, such as array processors or other mathematical engines, workstations or other nodes supporting specialized I/O equipment. There is even an advantage in accommodating nodes with their own proprietary mechanisms for achieving node-level dependability; see section §3.3.1 below for a discussion of such configurations.

These benefits carry a cost, that of the performance overheads of inter-replica communication protocols. There are, however, some valuable natural advantages to the geographical separation of redundancy underlying distributed fault-tolerance:

a) Replicas execute in physically distinct nodes that may be separately located within limits dictated only by the LAN technology used and any node input-output requirements; the nodes concerned may be located in different rooms, floors or even buildings. For this reason alone replicas can be rendered less susceptible to the common-mode effects of power surges, electromagnetic interference, floods, fires and so on.

b) Such physical separation requires loose coupling between replicas. The mechanisms that permit this also offer natural tolerance of a notorious and quite common class of software design errors that have become known as "Heisenbugs". This term was initially coined to refer to software (or hardware) design errors that lead to faults that can be tolerated by simple re-execution, the definition of the class being that a slight difference in execution circumstances is enough for the fault not to re-manifest itself. Such slight differences in execution circumstances occur between the support environments of loosely-coupled replicas. Heisenbugs are discussed further in chapter 6.

3.2.2. Fundamental Concept

Unlike some products that have briefly competed in this market, all *successful* products have been based on recognisably viable techniques for achieving dependability. This fact is, we believe, of great significance and lends support to the thesis that success for a product offering dependability requires that it be very evident to customers *how* that dependability has been achieved. In Delta-4, as with all the above approaches, a structured approach is taken in which a comprehensive set of identified problems is encapsulated and addressed by clearly defined techniques. A positive consequence of this approach is that customer doubts are localized; such doubts generally concern engineering details ("how do you make your common clock dependable?"), rather than fundamental principles ("on what basis can you be sure to detect

faults of *this* variety?"). In consequence, these doubts are more effectively reassured under such a structured approach.

The achievement and justification of node-level dependability do require extensive attention to be paid to the low-level mechanisms that discover faults and isolate sub-components both to avoid common-mode failure and to permit on-line repair. In the Delta-4 *distributed* approach to fault-tolerance, equally extensive low-level analysis is important with respect to particular sub-components such as the fail-silence of the NAC and, where this is required, of the host. Assuring fail-silence by low-level mechanisms is, however, a lesser goal than providing a complete solution for fault-tolerance within a node as well as providing for local on-line repair. In Delta-4, the distributed basis for fault-independence, fault-tolerance and the isolation needed to permit repair is handled rigorously. However, whereas in any dependable-node approach, details of node hardware raise important design issues (how power is supplied, what bus structures exist inside the node, how to carry out physical component replacement on-line, and so on), in the Delta-4 approach, the mechanisms are implemented at a level of abstraction in which such details are unimportant.

3.2.3. Real-Time Mechanisms

Whereas there are many commercial products that provide mechanisms to support some form of real-time behaviour, such products do not offer dependability support mechanisms. Conversely, products offering node-level dependability do not offer any mechanisms to support real-time behaviour, even in the relaxed sense of handling events and priorities (rather than the strong sense discussed in chapters 5 and 9).

The explanation for this is partly cultural, and partly a question of historical timing of the order in which problems have been tackled. Researchers and manufacturers active in the field of dependability achieved success addressing the issues of a large system market entirely devoid of real-time requirements, e.g., transaction processing. Meanwhile, a quite distinct "real-time culture" addressed the issues of a more specialized market in which dependability, when required, was traditionally addressed at the application level, not always with great success. Although several research projects have shown that the two fields largely overlap in issues and solutions [Dertouzos and Mok 1989, Jensen et al. 1985, Kopetz et al. 1988], this has yet to be reflected in the marketplace.

Section 3.3.3 below discusses Delta-4 as a product in relation to specialized real-time systems.

3.3. Comparison of Approaches by Target System Characteristics

The products discussed above provide or contribute to the platforms on which target systems are constructed. We now consider various system characteristics and their relationship to our chosen commercial examples and Delta-4.

3.3.1. Inherently Distributed Systems

Chapter 1 identified a broad range of application areas (CIME, Industrial, Commercial, IIP, DDBM) which are addressed by systems that are *necessarily* distributed. Most of these have been subject to rapid evolution within the last decade. This evolution has been stepwise, subject to the availability of appropriate technology to solve the most important perceived problem with sufficient economy.

3.3.1.1. Dependability in Distributed Systems. Inevitably, the need for services to exhibit varying degrees of dependability has been discovered by this market. This is currently addressed by arranging to provide node-level dependability and various mechanisms whereby nodes that do not possess enhanced dependability properties may access services provided by these.

To examine a single example, the danger of file loss is, unless special arrangements are made, a problem on specialized workstations linked to centralized file services. The consequences of file loss are more severe when critical information is on a single node rather than distributed. The probability of loss of a particular file is not changed significantly and when loss occurs, the result is more likely to be catastrophic (affecting all files) than unfortunate (affecting some files).

Although other directions have been taken in the marketplace, one identifiable evolutionary step in response to this has been to provide systems where such a central file service is based upon fault-tolerant nodes. This approach is taken in several systems that address the Automated Office market; it is perceived to be a cost-effective means of significantly reducing the probability of loss of files.

3.3.1.2. Distributed Consistency — the Need. The history of such evolution is one of "the next important problem" becoming recognised, as solutions to previous problems are accepted as the "State of the Art", based always on the changing economics of technology and cost. As noted in section §3.2.1 above, a file server based on a single fault-tolerant node remains susceptible to the common-mode effects of power surges, electromagnetic interference, floods, fires, etc. One approach might be to include several such nodes, separating them physically to the necessary degree (e.g., different buildings and electrical supplies). It is then necessary to perform data manipulations across several separately located files. Such manipulations must be made according to a model of distributed consistency acceptable within the context of the application. Indeed, distributed consistency seems to be a valuable mechanism; once it has been constructed, it may be applied, beyond its original purpose, to many applications (see chapter 6). However, such mechanisms are not native to conventional communications systems, nor efficient when built above them.

3.3.1.3. Distributed Consistency — the Delta-4 Solution. Unlike other approaches, distribution is part of the fabric of the Delta-4 solution to dependability. The problem of distributed consistency has been confronted, and the particular solution developed is based upon low-level mechanisms that support multicasting with atomicity (see chapter 10). This is not the case for the "dependable node" approaches described above, where vendors typically provide conventional OSI point-to-point communications support. When the need for distributed consistency arises, for instance when committing a distributed transaction, additional protocols must be implemented on top of the point-to-point protocols. In Delta-4, since distributed consistency of communications is a pre-requisite for ensuring the distributed consistency of replicated computation, extensive use is made of the explicit atomic multicasting protocols built into the Delta-4 communication system. This basic atomic multicasting protocol has allowed the design and implementation of a range of multi-partner protocols that greatly reduce the amount of application-specific code that must be written whenever the need for distributed consistency arises.

Moreover, Delta-4 is in the position of permitting the next step to be taken by present approaches. The Delta-4 solution to distributed consistency may be inherited by existing commercial fault-tolerant nodes by adding Delta-4 communication mechanisms. Distributed-system-level dependability could then properly be claimed for node-based approaches, and in addition, "mix-and-match" strategies become possible; these dependability technologies may

interwork transparently with components that use Delta-4 distributed replication models, to enable a tradeoff of cost against performance requirements. The facility is often provided on commercial fault-tolerant nodes to include externally supplied VME-bus based equipment; this would allow direct use of fail-silent NACs based on the prototypes developed by Delta-4, executing either MCS or XPA communications software. Otherwise, a fail-silent NAC interfaced to a proprietary bus could be developed. In either case, a port of the relevant host driver software is needed, or alternatively, MCS or XPA communications software could be ported in place of the OSI stack running on the host. Since the latter is fault-tolerant, the need for a specialized fail-silent NAC is removed.

3.3.2. "Bespoke-Tailored" Systems

Where the specification of a particular site requirement cannot be fulfilled by a standard product, a system must be specially engineered to fulfil the purpose. Such engineering is called "bespoke-tailoring" by analogy with the business of crafting a well-fitting suit of clothes from a chosen cloth, as opposed to the alternative of offering a choice from a range of "standard" off-the-peg suits.

Vendors of such systems remain competitive by optimising the design along many dimensions; time-scales, costs, quality, maintainability, etc. Advantage is taken of the fact that, even if the overall system is unique, it can be built using methods and from components that are not unique. Vendors therefore invest in *enabling technology*; that is, construction facilities, tools and philosophies that simplify the design process and allow systems and system components to be constructed in a modular and flexible way; to be maintained, reused and ported to new technology as it arises. An enabling technology should offer *transparency* with respect to a number of issues; in particular, transparency to distribution, to the underlying technology, and to dependability.

The ideal is to minimize any new development needed, by allowing system construction to be as much as possible the project engineering task of selecting and combining preexisting components, and as little as possible the programming task of constructing new components or changing the facilities of old components. To protect what can easily become a massive investment in applications software, a vendor must find means to maximize software portability across a range of underlying technologies, so that such software can be reused.

Usually, but by no means always, such systems are large. Methods developed to be applicable *in the small* do not usually scale well. In contrast, methods applicable *in the large* can, if due attention is paid to means of excluding any unnecessary "baggage" of support environment functionality from the system, also be applied in the small. It is highly desirable to a vendor that a spectrum of systems can take advantage of a single enabling technology, and that components can be developed and reused across this spectrum.

The ability to accommodate off-the-shelf components into systems developed under such a regime is also desirable, since it serves both to minimize new-development costs and to address issues of "fashion", such as where a particular relational database is specified by the customer.

Where the complexity of the requirement demands that the system itself is complex, distribution is generally an inherent physical requirement, and dependability becomes essential. If different services can be configured with degrees of dependability related to their role within the system, significant cost optimizations can be achieved. Incremental fault-tolerance is, of course, one of the objectives of Delta-4 (see chapter 1).

For the suppliers of bespoke-tailored systems, Delta-4 is in the position of supporting (rather than competing with) the above approaches. The requirements identified here are addressed by the Delta-4 Application Support Environment (Deltase, see chapter 7). Deltase is intended to be simple to port to any host and local execution environment (LEX), and this will

be trivial for any approach based on 680X0 and UNIX, as are the Delta-4 prototypes. This is the case with several commercial offerings falling into the categories discussed above.

The result of such a port would enhance these approaches through giving access to an ODP world. This would remove the dependency of application programs on what nodes they run on or across. Programmers of dependable nodes typically have dependability transparency, but "see" their own flavour of LEX. They explicitly take account of distribution, with all the well-known difficulties of assuring and maintaining the mutual correctness of separate units of remote interfacing code (typically constructed manually and by different programming teams), compounding the issue discussed above of assuring distributed consistency. Several manufacturers have addressed this by supporting a proprietary RPC across their own flavour of LEX and host, usually built above native OSI. As standards continue to evolve, those intended to support distributed computation will no doubt be accommodated by many manufacturers. Deltase programmers already live in an ODP world; even with the further compounding effect of heterogeneous machines, operating systems and languages, the difficulties outlined above disappear and the investment in code is protected against technology change.

With that transparency, many of the approaches described can be viewed as providing a "high-end" solution to dependability. Engineers could use different builds of system to meet different customer requirements, whilst preserving the software base.

3.3.3. Real-Time Systems

The meaning of the term "real-time" in Delta-4 is discussed in chapter 5. We repeat here our view that *reasoning about run-time timeliness behaviour is crucial to real-time system design.*

To allow such reasoning with respect to applications, run-time support-environment properties must be assumed and therefore must be supported. However, the features offered by some vendors of such environments sometimes do not allow whole classes of timeliness requirements to be assured, or lead to inordinate complexity in the design in order to achieve that assurance.

An environment claiming to be real-time must, at the least, provide some sort of support for handling events, timed activation and timed notification, with bounded latencies and predictable scheduling properties. Support environments that only offer mechanisms to handle events, or to promote such attributes as "a short average preemption latency", do not provide a good basis to address real-time support. (See [Le Lann 1989] for an analysis of the value of such mechanisms.)

A run-time support environment must explicitly support the design process with run-time mechanisms whose properties are based on sound timeliness modelling. To our knowledge, only Delta-4 has considered dependability requirements in the context of large-scale real-time system design. The XPA instance of the Delta-4 architecture is unique in offering both building blocks that allow software to be constructed according to the above perspective, and transparent support for the replicated execution of such software under a high-performance and criticality-respecting execution regime. This is discussed at length in chapter 9, and many of the relevant dependable input/output issues (also commonly neglected) are discussed in chapter 12.

Such systems are really a specialized subset of the "bespoke-tailored" category. Few offerings based on node-level dependability provide any support for this application area; if they were to be included as hosts into a Delta-4 real-time system, they would not be able to support real-time activity. However, they would not necessarily compromise other nodes offering this support. The XPA version of the Delta-4 architecture provides the necessary properties of a communication system that could accommodate such hosts; an appropriate port would only offer low-precedence communications access.

3.3.4. "Shrink-Wrap" Software

Manufacturers are concerned to maximize the return on effort; small-scale application porting exercises are seen as dissipative (in that such effort can generally be deployed more productively elsewhere) and lead to licensing complications, lack of clarity of maintenance issues, unwanted visibility of the inside of the application and so on.

What has become colloquially known as "shrink-wrap" software is often represented as part of the growing recognition that well-defined standard interfaces and clear structural separation of issues and responsibilities are valuable. In fact, shrink-wrap software really represents recognition of the financial rewards of providing the solution accepted by the market to problems common to large numbers of systems. Evidence for this view is the proprietary nature of many offerings (utility interfaces, language dialects, etc.), which runs exactly counter to the above ideal.

Shrink-wrap software is nevertheless emerging as the form of supply which customers, as well as manufacturers, prefer. Customers want to buy software off-the-shelf and run it immediately, whilst being as little constrained as possible over their choice of platform. There is little evidence that they are at present concerned by the consequences of becoming locked-in to a particular manufacturer.

This particular market is moving away from any need for software to be supplied in source or even unlinked-binary form. To achieve this end, major suppliers in the UNIX community are defining a series of binary standards. These must, of course, be specific to the underlying microprocessor; a number of these are recognised as worth supporting. There is strong pressure to avoid diversification of microprocessor types; the list can be expected to remain small. Most "stand-alone" fault-tolerant products based on the techniques described above claim to offer a solution to the problem of conferring dependability on "black-box" software. To guarantee that dependability will be conferred on the "shrink-wrap" market-driven variety, the technology concerned must also be aligned with this movement.

For this particular class of software, distributed fault-tolerance techniques encounter a somewhat different set of difficulties. Can we construct or make use of a host type that is able to accept such software and render it dependable using Delta-4 models?

3.3.4.1. Passive Replication. Passive replication relies on mechanisms that can efficiently take, transmit and install checkpoints. With black box software, checkpoints are difficult to collect and equally difficult to install. There is an additional complexity associated with correctly activating a passive replica after the active replica fails, unless the checkpoint is of a suspended state and the LEX data structures themselves represent this correctly. This would seem to rule out use of Delta-4 passive replica techniques.

3.3.4.2. Active and Semi-Active Replication. To arrange for "shrink-wrap" software to be executed in a distributed replica-group deterministic manner, some assistance must be provided by the LEX. Some recent work aimed at identifying how such a LEX might be constructed is given in appendix B. As with other prototypes developed within the Delta-4 project, this work is based upon UNIX as a representative, popular and open LEX, but the techniques developed are not dependent on any of its characteristics and therefore can be applied elsewhere. Engineering an accepted open version of UNIX in order to achieve the properties described is by no means a trivial exercise. Nevertheless, a possible mechanism does appear to exist that "shrink-wrap" software could be rendered dependable under Delta-4 support.

3.4. Case Studies

This section presents as two case studies the two applications that were used as pilot applications during the present phase of the Delta-4 project. One is a banking application for electronic payment authentication, the other is a production application for car manufacturing.

3.4.1. Pilot Site One

One partner of the Delta-4 project provides a good example of requirements for "bespoke-tailored" dependable distributed systems.

The partner runs an electronic payment business, using two separately located sites, each housing a large fault-tolerant host (dual configuration to provide continuous system availability as well as data integrity).

Because falling profit margins accompanied market growth, the partner faced new constraints and found it necessary to redefine its position with respect to operating costs and performance. The issues included: the overall cost of hardware and software, the cost of upgrading, operating difficulties linked to changes of scale, the inconsistency of batch processing with services intended to support on-line transaction processing,...

The two main sets of constraints that were identified match well the characteristics of Delta-4:

- To decrease the high cost of the fault-tolerant equipment, whilst increasing system capacity to keep pace with market growth, without impairing with system dependability (protection against file corruption, fire hazards and the like).

  One way of doing this is to replace large-scale expensive fault-tolerant hosts by a dependable distributed system that uses sets of smaller stations, linked through a LAN, but physically separated to the necessary degree. This has additional advantages in operational terms. It allows the necessary growth whilst enabling optimal system use, by supporting modular dependability and load-balancing. For example, the batch programs could run on stations other than those dedicated to the on-line services.

- To decrease the high cost of software maintenance, without reducing the span and frequency of the changes in application software that must take place because of the need to maintain position in an increasingly competitive and rapidly-evolving market.

  This can be done by:

  - modifying software production methods, in particular by shifting to the object-oriented modes advocated in the ODP model and implemented in the Delta-4 Application Support Environment;
  - switching from proprietary to standard operating systems and languages, to decrease training costs at a time of high turnover among information systems staff.

If the partner were to adopt generally the Delta-4 pilot site solution, additional advantages would accrue, such as the possibility of achieving very high levels of hardware fault tolerance and of using the fragmentation and scattering technique to ensure the extensive protection of sensitive files (see chapter 13).

3.4.2. Pilot Site Two

This pilot is based on a real case that has been implemented and is now operational in a car assembly factory. The CIME (Computer Integrated Manufacturing and Engineering) subset that has been selected for the Delta-4 project is one of the most critical of the factory since a failure of this subset would lead to a halt in production. This subset comprises several application components distributed on different levels of the factory (shop, cell, operator's level) thus representing a meaningful part of a CIME system:

1) At the shop floor level:
   - The *production management application* identifies the necessary components and the operations (manual, mechanical) to be carried out to build mechanical parts. It works on data produced by the production planning activities and stored in the shop database.
   - The *shop database* stores information about the sequence of cars to be produced with the associated references of mechanical parts, about the current state of production and about the list of components to use and operations to carry out per mechanical part reference.
   - The *message scheduling application*, under the control of production management, schedules messages to mechanical or human operators at the cell level. This scheduling is driven by the state of the work in progress (product tracking), and by the nature of data stored in the database.
2) At the cell level
   - The *message dispatching application* sends messages received from the shop floor level to the right destination within the cell and at the right time.
   - The *automation application* controls the real production process and ensures that this process executes the right operations; this component communicates with PLCs (e.g., for controlling movements of automatic guided vehicles), with identification posts and with human operators through asynchronous terminals.

The complexity of such a system is very high, and is increased by strong distribution and dependability requirements. The risk is therefore real of loosing the mastership of the overall system if the basic technology does not allow simplification and integration. In its initial implementation, the system was implemented over an infrastructure of heterogeneous standard computers communicating over LANs with standard OSI protocols, thus offering a message-passing paradigm with no built-in mechanisms for fault-tolerance. To decrease complexity and to achieve dependability, the application designers and programmers therefore had to fill the gap that leads to a higher level of abstraction by including some additional mechanisms within the application itself. This leads inevitably to mechanisms specific to the application that generally solve the problems of dependability and distributed consistency in an imperfect way.

The use of a Delta-4 platform for such an application places the application designer and programmer at the level of abstraction that enables him to concentrate on the resolution of the application-specific problems, and not of those which are implied by its distributed nature or by the dependability requirements. The use of RPC paradigms associated with multi-threading facilities offered by Deltase has for example much simplified the design of the message scheduling application, as opposed to the use of message-passing support. The distributed consistency of production data and events is transparently offered by the Delta-4 infrastructure, and does not have to be reconsidered in the application (as opposed to what was necessary in the initial implementation).

This application was therefore a good candidate for Delta-4 as an enabling technology for building large-scale complex distributed applications.

3.5. Conclusion

As is proper in the context of an open project such as Delta-4, with both academic and commercial aspects to the development, the above comparison is unusually frank and unbiased by comparison with what would be found in, let us say, the sales literature of a new commercial product. It should be evident that, as is common in all engineering situations, ideal solutions do not exist. We can approach, but not achieve "the best of all possible worlds", and the approach taken is inevitably coloured by the intended market.

For this market and, we believe, for many others, Delta-4 does offer a "shopping list" of significant advantages and advances. Taken in isolation, the list given below, whilst impressive, would be open to a criticism of bias from vendors operating in any market that did not recognise the value of the attributes listed; the above discussion is intended to redress the balance. We nevertheless contend that the contents of the present book amply justify our claiming the following list of benefits to our architecture:

- extension of the concept of *openness* to dependable systems:
  - removal of the traditional restrictions to single vendor;
  - support for hardware and software from many sources;
  - contributions to the evolution of standards;
  - public availability of Implementation Guides;
- selectable levels of dependability at the granularity of individual services;
- accommodation of variants:
  - off-the-shelf hardware which does *not* possess native mechanisms for fault-tolerance;
  - specialized non-Delta-4 hardware which *does* possess native mechanisms for fault-tolerance;
  - new technology as it arises;
- mix-and-match capability allowing cost/performance tradeoffs between any of the above variants:
  - maximization of software portability;
  - uniform set of technology-independent support mechanisms;
  - mapping to existing languages and development tools;
  - minimal restrictions on choice of language;
- accommodation of software not corresponding to Delta-4 principles:
  - preexisting popular applications such as SQL databases;
  - MMS applications software;
- support for specialized application domains such as *distributed real-time* in a much more coherent manner than exhibited by current commercial products.