

DIVISION OF COMPUTER SCIENCE

**File Server Architecture for an Open Distributed Document
System**

**Bruce Christianson
Ping Hu**

Technical Report No.204

August 1994

File Server Architecture For an Open Distributed Document System

Bruce Christianson Ping Hu

School of Information Sciences, Hatfield Campus

University of Hertfordshire, U.K.

{B.Christianson, P.Hu}@herts.ac.uk

Bean Snook

School of Computing Sciences, Milton Keynes Division

DeMontford University, U.K.

jsnook@dmu.ac.uk

Abstract

In this paper we will investigate design and implementation strategies for a file server in an open distributed document system. The aim of the open distributed document system is to provide an environment where a group of geographically distributed users can collaborate to develop documents efficiently and be assured that their integrity requirements will be enforced. We view the integrity policy as part of social contract between users. The services provided by a file server can be divided into two categories according to whether a service is globally or locally trusted. In this paper we call the entity that provides the globally trusted services *visibility server*, the remaining services are provided by *validation servers*. The functions of the visibility server will be kept to a minimum, and can be running in an off-line manner. The responsibility of each validation server is to check whether the document integrity will still be maintained if an update transaction is committed. The validation servers are independent of each other and “stateless”, i.e. each server can always reboot itself before it validates a transaction. An optimistic transaction concurrency control approach is employed for document processing, so that the open distributed document system can achieve very high document availability.

Keywords: data integrity, distributed system, file server, security, transaction concurrency control, trust.

1 Introduction

The collaborative development of documents by a group of geographically distributed users could be accomplished in an open distributed system. Transparent access across the distributed system greatly simplifies the resource sharing. Replication makes services highly available to users. But in a truly open distributed system, resource sharing among a group of users raises another challenge that mechanisms are required to ensure such sharing in a secure, reliable, efficient and usable manner that are independent of the size and complexity of the distributed system.

An object-based document architecture for open distributed systems, which is called *DODA*, is developed in [3, 11]. In this paper, we will investigate design and implementation strategies for a file server in an open distributed document system based on (but not strictly conforming to) the *DODA*. In such a system, we assume that users are more concerned with the integrity and authenticity of documents than with other security aspects, such as confidentiality. Moreover, we will see that instead of imposing a universal notion of integrity over the distributed system, we would rather view the document integrity as part of a social contract between users, and probably with the consent of the system. We shall argue that a group of users across the distributed system who intend to collaboratively develop documents must be able to specify and agree their own notions of integrity independent of any policy provided by shared infra-structure or services and other user groups.

2 Distributed document system

We assume that the open distributed document system (i.e. *DODA*) adopts immutable object schemes [10], i.e. documents are represented by a history of immutable versions. When a change to a document is committed, a new document version is created against the existing document state (old version) which is left unchanged. The open distributed document system does not care much about where and how documents are stored/archived in the distributed system. The document processing system merely assumes that a facility exists which ensures a reliable and permanent medium storage. That means a series of versions associated with a document are maintained somewhere in the distributed system, and users have means to easily access a particular document version (probably transparently) if they know enough information of the version¹.

The scenario is that in an open distributed system, a group of users mutually agree an integrity policy and want to make sure that the policy is imposed on documents that they are collaboratively developing. The users might be situated in different security domains, and it must be very hard, if not impossible, to set up some infrastructure or services which are globally trusted by every remote participant in the distributed system. Each individual user in the group only trusts the infrastructure and services that are chosen for use by himself and that are local to his domain.

¹We will discuss the version information in Section 3.1

Users make changes to their documents through transactions, i.e. two adjacent versions of a document are linked by a transaction. Transactions usually last very long term, compared with document transfer, hash value calculation, or similar activities. Transactions are relatively conflict free, but resolution of the conflicts in failed transactions frequently requires off-line interaction between related users. These imply that an optimistic approach to processing transaction is better than an pessimistic one, and a user usually does not mind that the formal announcement to other users that his transaction is committed has some delay as long as such a delay is short compared with the transaction execution time.

As we have stated, the main goal of designing an open distributed document system is to provide an environment where a group of geographically distributed users can collaborate to develop documents efficiently and be assured that the integrity requirements will be enforced. Here we list some strategies that are used.

- The global trust is kept to a minimum, which means both the number and complexity of trusted entities. We would prefer that such globally trusted entities are running off-line. On-line service makes it more vulnerable to malicious attacks.
- The establishment of document integrity policy is mutually agreed between the user group and the system. The system does not prevent any other users from reading documents, but it only allows the authorised users to make changes to the documents through a proper procedure.
- A user or a user group requires a trusted local environment in which complicated methods, such as document integrity check, can be executed. By saying "local", we mean the user trusts the entities, such as infrastructure and services, that he chooses to use.
- To make documents highly available to users, an optimistic approach for concurrent transaction control would be a better choice. Because of long transactions, correctness criterion other than one-copy serialisability might serve users best.

We should make it clear that the term *file or document server* which we will use throughout the following discussions slightly differs from what we usually mean for distributed file systems. For example, the document archive will not be discussed in this paper.

3 File server partitioning

From the previous discussions, we would argue that to meet the designing requirements, it desirable to partition the traditional distributed file server into two parts, i.e. a visibility server and a validation server. The two servers provide services to manage the distributed documents and also achieve high security and efficiency.

3.1 Visibility server

The role of a visibility server in the open distributed document system is like that of a moderator. It officially announces to other users that a transaction is committed. That is a new document version will be accepted by the distributed document system only if it is confirmed by its visibility server. On receiving a request, the server will return a certificate which identifies the “current” document version to the originator. A certificate (as suggested in [3]) contains at least

- the document name,
- the protection number of the current document version and
- a timestamp,

all signed under the private key of the visibility server. A document version is *current* if it is the latest accepted version to that document. From the information in this certificate, a user could then access the version in the distributed system, and more importantly verify the authenticity and integrity of the document version.

The visibility server and its services are trusted globally. But the visibility server itself could be either centralised or distributed in the distributed system. In the case of distribution, a protocol is required to coordinate those distributed visibility servers². All users in the distributed system believe that the visibility server is capable of providing following services

1. To respond to requests to commit update transactions to the current document version. If a transaction satisfies the requirements of document integrity policy, the new document version will be accepted and announced publicly.
2. To safely maintain the critical information about document versions.
3. To issue document version certificates. Users have to believe what the visibility server says.

Since it is bearable for the visibility server to delay the announcement of committed transaction, the visibility server could periodically publish newly created document versions and each domain in the distributed system could cache those information for local use. It can be seen that functionality of the visibility server is minimised. It is essentially an off-line name server. Because of its minimal functions and off-line services, the visibility server should be easily managed, monitored and protected, although it is not stateless.

²An individual visibility server could then not be globally trusted. However, the distributed document processing system requires that all distributed visibility servers collectively provide services that are globally trusted. For the simplicity of discussion, we consider the situation that only one visibility server is devised in the system.

3.2 Validation server

A validation server, if asked, provides a service with a proof that an update transaction to a document version is valid and leads to a new version. Very generally, a user gets a copy of a current document version with the help of the visibility server. The user could update the document to a new version as long as he could get a validation server's proof that the update transaction is valid according to the integrity policy. If the visibility server finally accepts the update transaction and the corresponding proof, the new document version is created and will be seen by all others shortly.

A validation server is virtually stateless, and can be replicated in the distributed system. These distributed validation servers are independent of each other, and the distributed document system does not necessarily require them to coordinate. The responsibility of a validation server is to make sure that a submitted update transaction to a current document version will not cause any integrity breach if the transaction is committed. As we discussed before, a document integrity policy is part of social contract between users in a group. The policies might be different from user group to user group. Although the distributed document system could develop some system-wide validation methods for valid user transaction, it is likely that an individual user group would prefer to specify its own validation methods that, together with other system-wide validation methods, assure any update transactions will conform to its own integrity policy³. Clearly, the operations carried out by validation servers to validate update transactions could be diverse and very complicated, and even in one validation server, the validation methods executed this time usually are different from the methods to validate last transaction. However, because of the characteristics of independency and statelessness of the validation servers, a user could always ask a validation server to reboot itself before its update transaction is validated. A rebooted validation server will provide a secure environment [7, 8] for transaction validation. Furthermore, because of the independency and locality of the validation servers, one spoilt validation server (either accidentally damaged or maliciously broken) will never affect any services provided by other validation servers.

As the result of partitioning the file server into visibility server and validation server and distribution/replication of the validation server, those validation servers are not globally trusted any more. Actually, as the document integrity policy for a user group could be defined by the group at their own will (probably with the consent of system), why could the group not define (choose) their own validation server(s)? This diversity has the result that the services provided by a validation server are trusted only by its potential users. Of course, to make it function, each validation server must be trusted by the visibility server that it is competent to validate update transactions. But we should bear in mind that the visibility server believing the competence of a validation server only means that its users have chosen and trusted its services. The responsibility is still on the side of the validation server users. So the trust relationship between user and the visibility server is slightly different from that between the validation server and the visibility server. However,

³These self-defined validation methods could be included in the document itself under the guidance of the integrity policy.

we should bear in mind that the distributed document system still could have its own fundamental criteria of what is requested to become a validation server. This is what we mean the choice of integrity policy *with the consent* of the system. After all, the system must take its responsibility for the system-wide validation methods it defines.

Further to the above discussion, we could see that there is nothing that can stop some users in a group from trusting a validation server which is different from other user's in the same group, although efficiency could be affected. For example, user A and user B are in the same group, and there exist two validation server V_A and V_B for them to collaboratively develop a document. Even if A does not trust V_B 's service to validate B 's update transaction to document, he could always, in the last resort, re-check B 's update transaction on validation server V_A . In an extreme case, user A could trust none of the update transactions to the document but only those checked/re-checked by the validation server V_A . That also implies that some entity in the open distributed document system becomes a validation server mainly because some potential users trust its services.

3.3 Document processing

In this section, we will examine the relations between the visibility server and the validation server and between the servers and users.

Figure 1 shows, in a much simplified way, the operations with which a number of users collaboratively develop a document over the open distributed document system. Users turn to the visibility server to ask for a certificate whenever they intend to access a document (step 1 and 2). Users trust the services provided by the visibility server in a way that the certificate includes information that indicates the *current* document version and verifies the integrity of the document version obtained from *document version archive* (step 3). So a certificate should at least include the document name, the current version's protection number (e.g. a collision-free hash value of the version) and a timestamp which assures the freshness of the certificate.

We have not so far mentioned the document-version archive. This entity is not necessarily a part of the open distributed document system although in Section 2 we require the open distributed document system to adopt immutable object schemes. We would rather say it is an independent service provided by the distributed system on which the document system is built, i.e. we assume in the distributed system there exists a reliable function unit for information storage. However, if there is no such a reliable document version archive in the distributed system or the service is not trusted by the entire user community, the user group is free to devise their own document version archive. The basic point is that the document version archive is independent of both the visibility server and the validation server that we have presented. In fact the archive could be placed anywhere in the distributed system, and it could even be cached or replicated for efficiency and performance. But as we have discussed, this service takes no responsibility of maintaining the document integrity, partly because it is not a part of the designed distributed document system. The dotted line in Figure 1 represents the boundary of the open distributed system.

To commit an update transaction to a document, i.e. to create a new document version

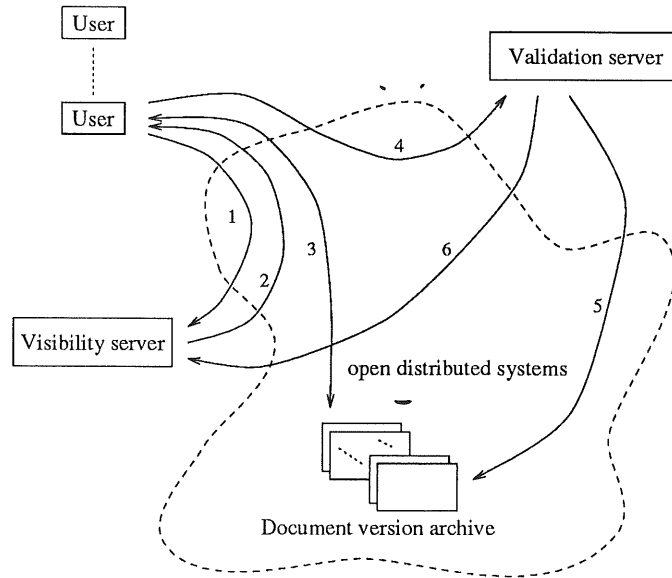


Figure 1: Illustration of document processing

against the current one, the user has to submit his transaction to a validation server for approval (step 4). Among all validation servers in the distributed system, each user must trust at least one of them to faithfully validate the transaction according to the integrity specification, to create a new document version in the archive and to pass relevant approval information to the visibility server (step 5 and 6). On the other hand, the visibility server must believe that the validation server is capable of doing transaction validation check and creating corresponding document version.

4 Integrity policy

In *ISO 7498-2*, the (data) integrity is defined as “the property that data has not been altered or destroyed in an unauthorised manner” [1]. As we assumed, the information storage unit in the open distributed system is reliable, therefore to keep the data integrity we need only to devise mechanisms that could prevent data from unauthorised modification in the open distributed document system.

As it is mentioned before, we view the integrity as part of social contract between users to judge whether a modification is authorised or not. A group of users who wish to develop document collaboratively should specify and agree their own notions of integrity in explicit form. Clearly, each user group could have their own integrity policy, and even one group could have several mutually independent integrity policies for each document they are developing. Also there is no reason why a user group could not change their integrity policy during the process of document development as long as such a change does not violate the social contract. These show the integrity policy has the properties of individuality, independency, and judgement at owner/user’s discretion.

On the other hand, an integrity policy, at least part of it, has to be implemented through the services provided by the distributed system. This dependency means that an integrity policy probably includes those fundamental integrity criteria that are enforced system-widely by system infrastructure. It also means that any services for maintaining integrity that are beyond the system capability have to be constructed by the user group themselves.

For system feasibility and efficiency, it is desirable for the designed distributed document system to devise some integrity check methods. These services could be either enforced to all document development groups or provided for individual group's selection to fit their integrity requirements.

5 Global trust or local trust

Let us look at a conventional distributed file system. A file server coordinates all transactions to the files it manages in the distributed system. The server itself could be either centralised or distributed, but it is trusted by all users. If it distributed, the system should employ a protocol for the distributed components to work harmoniously. Then some degree of trust relations should be established among those components.

The globally trusted file server makes it more vulnerable to attackers since the whole system relies upon services provided by the file server. However, it can be easily seen that some of those services are not necessarily trusted globally. Furthermore, certain user groups might ask for some special services as part of their document integrity check. It is likely that such services are only required to be available to a particular user community, or even specially designed. It is almost impossible to ask the file server to provide all possible services to satisfy various integrity requirements as integrity policy could be "arbitrarily" specified by individual user group. It would certainly complicate the management of the file server and make the server harder to protect from intruders if integrity policy could be revised during the processing from document development in such cases.

In the proposed open distributed document system, services provided by the file server are divided into two groups according to whether they need be globally trusted or not. We expect that there should exist only a few services that are globally trusted and have to be left in the file server, which is now called *visibility server*. Although it requires global trust and is not stateless, the visibility server should be easy to manage and protect because of its minimal functions and off-line service provision as we discussed in Section 3.1. Those services that are moved out of the visibility server form a new server, called *validation server*, which is only trusted locally by its prospective user community. Because of the local trust and the property of its stateless, the validation server can be replicated over the distributed system and more importantly each replica can operate independently of the others. As we discussed in Section 3.2, the validation server is responsible for the document integrity, so its services largely reflect the integrity policy of the user community. Furthermore, any entity in the distributed system could become a validation server if some users would trust its services to enforce their integrity policy and could convince the

visibility server the entity was competent to do the job.

From this analysis, we can see that in the open distributed document system the visibility server is globally trusted whereas the validation server is only trusted locally by its prospective user groups. The visibility server, together with the document version archive, provides a reliable service for document storage. The validation server checks or validates document integrity. There could exist many validation servers in the distributed system. A particular validation server is trusted by those users who use it, and a validation server could even be created by a user community provided that the visibility server is convinced of its competence. We view the protocol of how to create a validation server as part of the integrity policy for a user group⁴. The provision of globally trusted visibility server and locally trusted validation server should give user groups over the distributed system great flexibility for collaborative document development.

6 Concurrent transaction control

One of another major problems for distributed document processing is concurrent transaction control because document versions could be replicated or cached in the open distributed system at user's please. Problems arise when two or more transactions attempt to update the same (current) document version simultaneously, i.e. conflict. Even if there is no document replication the problems still exist as long as the validation server is distributed and some services in the distributed system are suspicious, e.g. locking mechanisms. Many protocols have been proposed for maintaining consistency of distributed file systems [4, 6]. Generally speaking, the protocols fall into one of two categories, i.e. *optimistic* and *pessimistic*. Pessimistic protocols make worst-case assumptions about transaction conflict, and operate under the pessimistic assumption that if a transaction can conflict with others, it will. Whereas optimistic protocols operate under the optimistic assumption that transaction conflicts, even if possible, rarely occur. Mechanisms must employed in these protocols to first *detect* conflicts and then *resolve* them. Pessimistic and optimistic approaches are in the two extremes of conflict assumption. Each of them has its own advantages and disadvantages. It is up to individual application to choose one most suitable for the environment.

The proposed open distributed document system adopts a thoroughly optimistic approach for document processing. Documents in the system are freely replicated, migrated or cached at user's will, and users are free to operate upon documents as they please. By using such an optimistic approach, users enjoy very high availability of documents, but they have no guarantee that their update transactions will not conflict with other transactions issued by others concurrently, which leads waste of resources. An optimistic approach is a better choice mainly because of rare conflict transactions, conflict transaction rescue and locking mechanism implementation.

We view a document as structured text with a defined operational semantics [3]. In an open distributed document environment, documents are manipulated by transactions which

⁴A user group should also take responsibility for those validation services defined by themselves.

are initiated by users. Distributed document processing, e.g. cooperative development of a suite of software by a group of users to meet some defined requirements, is typically evolved by very long term transactions. The probability that one transaction conflicts with another one is very low. In the case of document processing, if the work done by one transaction is incompatible with what others have done, part of the work could usually be rescued. For example, two conflict transactions could be merged without integrity violation by simply text cutting and pasting, but resolution sometimes requires off-line interaction between related users. From user's point of view, an optimistic approach for document processing is a better choice than a pessimistic one, because the user will hardly encounter the situation that his transaction will conflict with other's. If a pessimistic approach was used, time and resources could be wasted to prevent the rare situations, i.e. conflicts, from occurring. Even if a user later hears a transaction conflict, he would not be disappointed as part of his work could be rescued⁵.

Now let us discuss what is the criterion to determine transaction conflicts and who is capable of detecting conflicts. A generally accepted notion of correctness for a distributed file system is that the system has the same input/output behaviour as a centralised, one-copy file system that executes transactions one at a time (*one-copy serialisability*) [2, 12]. The criterion has two characteristics, i.e. the multiple copies of file behaves like a single copy (insofar as users can tell) and the effect of a concurrent transaction execution is equivalent to a serial one. The former is guaranteed by the visibility server, as only the visibility server has the authority to "officially" announce the current document version. The latter (serialisability or *atomic transaction commitment*⁶) needs more detailed discussion.

The serialisability is a very strong correctness requirement. It is popular because it is simple and intuitive, and can be enforced by very general mechanisms that are independent of both semantics of the file being stored and the transactions manipulating it. However, as the proposed system is aimed at document processing and employs an optimistic approach, we would prefer to ease the serialisability requirement for correctness to reduce the probability that transactions conflict or conflicting transactions have to be re-done. For example, some correctness criteria in the form of *integrity constraints* could be used for concurrency control so that two or more concurrent transactions are compatible even though the execution is not serialisable. Clearly, the criteria are related to semantic constraints and yet need further investigation.

It is obvious that the visibility server could be used for concurrency control as it will be notified of all changes to document versions. Actually the visibility server is responsible for the final integrity check before a new document version is visible to users. Broadly to say, the correctness criteria for concurrency control could be viewed as part of integrity policy as the conflicts lead integrity violation. But the check should be rather primitive because of the simplicity of the visibility server. What we are more interested in is to

⁵This is acceptable provided that the cost of rescue part and re-doing the other part of the transaction is likely less than that of re-doing whole transaction.

⁶Atomic commitment means the execution of each transaction is "all or nothing", i.e. either all of the transaction's operations are performed or none are performed [5], which is, in fact, equivalent to the serialisability.

exert concurrency control at the validation server level in order to enforce complicated integrity policies. The dilemma is that there is no reliable and trusted relations between the distributed validation servers for concurrency control. However, nothing could prohibit the open distributed document system from setting up informal connections between the validation servers for such a purpose. While a validation server validates a transaction, it can make an informal contact with other distributed validation servers to detect possible conflict transactions. Whenever a possible conflict is detected, the two involved validation servers should try to resolve it themselves, otherwise the users who initiate the transactions have to be warned. A warned user could either instruct his validation server go ahead unanimately or make an off-line contact with the user at the other end to cooperatively resolve the problem. We would expect that a large portion of conflicts could be resolved or avoided at this stage. Again, how the strategy works will largely depend on the details of enforced integrity policy.

Yet another reason why a pessimistic approach is not used for document processing is that to realise the approach a locking mechanism or similar protocol is a basic requirement. For a specific system like the open distributed document processing, we could argue that there are some difficulties to properly implement a locking mechanism. Firstly, another service, i.e. locking mechanism, has to be trusted globally besides the visibility server. The situation that a group of users over several domains collaboratively develop documents will further complicate the problem of locking mechanism implementation because in any truly open system, autonomous management domains will never unconditionally relinquish control over their resources and domain administrators will always retain a last-ditch means of reclaiming control over “their” resources [3]. Secondly, the efficiency of document processing could be affected by the size of objects that the locking mechanism applies if a pessimistic approach is used in the document processing system. Some properties of object protection are discussed in [9]. Surely, a very fine grained object locking mechanism for the pessimistic approach could certainly avoid most of the pseudo-conflict cases, i.e. two or more transactions that appear to conflict but actually their operations are compatible. However, such a locking mechanism must be very complicated and difficult to manage. But for the system efficiency, some kind of “soft”, simple and untrusted locking mechanisms could be devised to give warning to relevant validation servers and users of possible transaction conflict. Precautions can be taken by the warned users, e.g. off-line contact. We would expect by using an untrusted locking mechanism some conflict transactions could be warned and thus avoided at their early stage, while the trust relationship in the system and the optimistic approach for concurrency control could still remain unchanged.

7 Conclusion and future work

Architecture of file server for distributed document processing is investigated in this paper, which enables a group of geographically distributed users to collaboratively develop documents in a secure, reliable and efficient environment and to be assured that the integrity policy is enforced based on an open distributed system. The proposed architecture splits

a file server into two parts. One is called visibility server, which includes all globally trusted services, but for security reasons it should keep its functional entities to a minimum. Preferably, the visibility server could run in an off-line manner. Another one is called validation server, which is only trusted by its “local” users and is responsible for transaction validation. Also because of its stateless, it could be easily replicated in the distributed system and each replica could operate independently to others. Instead of a universal notion of integrity, the document integrity is viewed as part of social contract between users and probably the system as well. So each user group who collaboratively develop a document can specify their integrity policy for the document. An integrity policy in the distributed document processing system has the properties of individuality, independency, and judgement at owner/user’s discretion. An optimistic approach for document processing is employed to control concurrent transactions. The open distributed document system that adopts the file server architecture should be able to achieve very high document availability and provide each user group great flexibility for collaborative document development.

There are still several topics that need further investigation for this distributed document processing system. Integrity policy specification is one of the major research areas we would like to carry on. Integrity policy also influence the trust relations between users and validation servers and between validation servers and visibility server, and has its impact on the way that a validation server is constructed. Concurrency control is another research area, which includes correctness criteria of integrity constraints, and implementation strategies for transaction conflict detection and resolution. Probably a formal specification is desirable.

References

- [1] ISO 7498-2. *Information Processing Systems – Open Systems Interconnection – Basic Reference Model, Part 2 Security Architecture*. International Standards Organization, 1988.
- [2] P. A. Bernstein and N. Goodman. An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Transactions on Database Systems*, 9(4):596–615, December 1984.
- [3] B. Christianson and B. Snook. Shrink-wrapped optimism: The DODA approach to distributed document processing. Technical Report TR-187, School of Information Science, University of Hertfordshire, March 1994.
- [4] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, September 1985.
- [5] J. N. Gray. Notes on data base operating systems. In R. Bayer, R. M. Graham, and G. Seegmuller, editors, *Operating Systems: An Advanced Course*, pages 393–481. Springer-Verlag, Berlin and New York, 1978.

- [6] P. Hu. *Dynamic Supporting: An Efficient Method For Replicated File Systems*. PhD thesis, University College London, London, U.K., April 1993.
- [7] P. Hu and B. Christianson. Problems while remotely booting a workstation. Technical report, School of Information Science, University of Hertfordshire, 1994. In preparation.
- [8] M. Lomas and B. Christianson. To whom am I speaking? Remote booting in a hostile world. Technical Report TR-178, School of Information Sciences, University of Hertfordshire, January 1994.
- [9] M. R. Low. Fine grained object protection in UNIX. Technical Report 130, University of Hertfordshire, March 1992.
- [10] S. J. Mullender. *Principles of Distributed Operating System Design*. PhD thesis, Vrije Universiteit, Amsterdam, October 1985.
- [11] J. F. Snook. *Towards Secure, Optimistic, Distributed Open Systems*. PhD thesis, University of Hertfordshire, Hatfield, U.K., September 1992. Computer Science Technical Report 151.
- [12] I. L. Traiger, C. A. Galthier, and B. G. Lindsay. Transactions and consistency in distributed database systems. *ACM Transactions on Database Systems*, 7(3):323–342, September 1982.

