DIVISION OF COMPUTER SCIENCE

Shrink-Wrapped Optimism: The DODA Approach to Distributed Document Processing

Bruce Christianson
Bean Snook

Technical Report No.187

March 1994

Shrink-Wrapped Optimism: The DODA Approach to Distributed Document Processing.

Bruce Christianson
(B.Christianson@herts.ac.uk)

School of Information Sciences, Hatfield Campus, University of Hertfordshire

Bean Snook
(jsnook@dmu.ac.uk)

School of Computing Sciences, Milton Keynes Division, DeMontford University, England, Europe

March 1994

Abstract. In this paper we introduce a distributed object-based document architecture called DODA in order to illustrate a novel strategy for achieving both high availability and high integrity in the context of open processing distributed between mutually suspicious domains without a common management hierarchy.

Our approach to availability is to structure documents into small components called *folios* in such a way as to allow the maximum opportunity for concurrent processing, and to allow these components to be freely replicated and distributed. Integrity conflicts are resolved using an optimistic form of control called optimistic integrity control (OIC) applied to recoverable work units.

Our approach to security is to shrinkwrap the document components using cryptographic checksums, and to provide a set of building block components called *functionaries* which a group of users can combine in such a way as to provide each user with a means of ensuring that an agreed notion of integrity is enforced while relying upon a minimum of non-local trust.

In particular, we do not rely upon a trusted computing base or a shared system infrastructure. The local availability of document versions and of the resources to process them are completely under local user control. The lack of availability of the functionaries does not have severe consequences, and the presence of mutual suspicion makes it easier to ensure that users can trust the functionaries to provide the intended service.

A major benefit of using OIC is that it allows the integration of untrusted components such as filestores and directory servers into the system. In particular, an untrusted soft locking service can be used in order to reduce the number of concurrency conflicts, and untrusted security components can be used to screen out attempted access control violations.

Keywords. access control, availability, concurrency, CSCW, distributed, document, integrity, object-based, OCC, optimism, security, semantics, trust, version control.

1. Introduction. The Scylla and Charybdis of open distributed computing are lack of availability and loss of security. Availability, in the sense of transparent access being provided to data and services from anywhere in the open system and at any time, and security, in the (wide) sense of the integrity of data and services being protected from the effects of malicious or inept behaviour on the part of users or other system components, are apparently paradoxical requirements. Attempts to increase availability, for example by replicating resources such as files, tend to increase the opportunities for destroying system integrity, for example by someone updating the wrong version of a file. Conversely, conventional attempts to make a distributed system more secure are usually perceived by users as an attempt to make its resources and services less available.

Most attempts to resolve this paradox in the area of what has come to be called computer supported cooperative working (CSCW) either do not explicitly address the issue of trust, or assume global trust on the part of the participants of a large part of the computer based environment itself, including infrastructure (such as hardware, operating systems, networks) and remote services (such as file stores and name servers).

The cooperative sharing of resources (including software artifacts and data) is fundamental to the motivation for building distributed systems. But in any truly open system, autonomous management domains will never unconditionally relinquish control over their resources, since there is no "lowest common boss" in a shared organizational lattice by whom disputes over such resources could subsequently be resolved. Domain administrators will always retain a last-ditch means of reclaiming control over "their" resources.

Consequently, where computer supported cooperative working crosses organizational boundaries, it is frequently inappropriate to assume global trust on the part of participants. In practice, different members of the consortium will have different views about which other parts of the system they trust, and for what purposes. Users may entertain doubts about the honesty or competence of other consortium participants or their agents, or may be unwilling to rely upon trust to ensure the correct action of remote parts of the system belonging to other domains over which they ultimately have no administrative or management control.

On the other hand, and for the same reasons, it is usually reasonable in a genuinely open system to insist that participants should trust the infrastructure and services which they themselves choose to use and which are local to their own domain[1]. This principle of local trust is the exact reverse of that obtaining in the case of a strictly hierarchical (closed) distributed system.

Because of this locality of trust, it is unsafe to rely upon a global trusted computing base to enforce a common security policy in an open distributed system, since this would require users to trust infrastructure or services in every remote participating domain. Divergence of security domains and resource management domains makes such an approach problematic in any case: frequently users (and resources) in a single administration domain will wish to participate in a number of cooperative consortia, each operating a different security

[1]An important (and often neglected) consequence of openness is the counterpoint to the principle of least privilege: the need to allow users themselves to confine the range of processing operations for which they are willing to accept responsibility. This allows users to reflect their degree of trust in the local facilities by indicating what uses of it they are willing to have attributed to them by users in other domains (and by implication, allows them to repudiate any other purported use.)

policy. Even if the policies are compatible, the software to enforce one policy might not be certified to perform the necessary operations on the shared resources by the other policies. The hierarchy of control need not correspond to the hierarchy of trust, even if both exist.

In this paper, we attempt to explore some of these issues in the context of a distributed object-based document architecture called DODA[2]. We understand a document as, broadly, structured text with a defined operational semantics. For example, we would regard the development of a suite of software to meet a tender requiring compliance with some defined quality standard such as ISO 9000-3, by a consortium of subcontractors who are competitors with respect to other tenders, as an example of the cooperative evolution of a document.

Document processing is an interesting example to consider because documents exhibit a number of characteristics which complicate attempts to resolve the distributed system paradox by conventional approaches. Documents are typically evolved by very long term transactions (days or even weeks). The probability of transaction conflict is typically fairly low[3], but resolution of the conflicts in failed transactions frequently requires offline interaction between humans. The state of a document is easily encapsulated, and consequently documents can be freely replicated and migrated. Facilities for performing operations upon documents exist at a wide range of different locations in different domains. Consequently control of access to a document and control of the resources used to access the document cannot simply be conflated.

In this paper we are primarily concerned with mechanisms for ensuring the integrity of documents (rather than, say, their confidentiality.) We do not impose a particular notion of integrity. Central to our approach is the view that document integrity is part of the social contract between users from different domains, under which the cooperative work is taking place. Users must be free to specify and agree their own notions of integrity independently of any policy provided by shared infrastructure or services. Such a shared notion of integrity may include protection of the state of the document from deliberate tampering or accidental corruption, enforcement of access control, version control, concurrency control and/or semantic consistency, and the provision of an unforgeable audit trail. Rather than follow the usual practice of separating these issues into a hierarchy of layers, designing the layers independently and then trading off the consequences, we shall argue that it is both possible and desirable to adopt a unified approach to enforcing integrity.

We do not seek to prevent users from freely replicating and manipulating documents in their own domains using uncertified software in an arbitrary fashion. Rather, we seek to prevent a user or system service from successfully "passing off" an illegitimate or untimely version of a document to another principal as authentic.

Our approach to availability is to structure documents into small components called *folios* in such a way as to allow the maximum opportunity for concurrent processing, and to allow these components to be freely replicated and distributed. Integrity conflicts are resolved using an optimistic form of control called optimistic integrity control (OIC) applied to recoverable work units.

Our approach to security is to shrinkwrap the document components using crypto-

[2]DODA is more fully described in [10], but the terminology used there is slightly different.

[3]At least in a well-organized consortium.

graphic checksums, and to provide a set of building block components called *functionaries* which a group of users can combine in such a way as to provide each user with a means of ensuring that an agreed notion of integrity is enforced while relying upon a minimum of non-local trust.

As we shall see, a major benefit of using OIC is that it allows the integration of untrusted components such as filestores and directory servers into the system. In particular, an untrusted soft locking service can be used in order to reduce the number of concurrency conflicts, and untrusted security components can be used to screen out attempted access control violations.

2. Document Structure. A DODA document consists of a set of components called pages, arranged in a rooted directed acyclic graph (DAG). The leaves of the DAG contain only document text. Pages which are not leaves are called *folios.* Folios are index pages which identify the contents (children) of that folio. Each page is uniquely identified by a protection number, obtained by applying a collision free hash function to the page contents[4]. The protection numbers of the children (but not their text itself) are included in the data hashed to form the protection number of the parent. Since the hash function is collision free, knowledge of the correct protection number for the base folio thus allows a user to verify the integrity of the entire document, or of any connected part of it.

Each page is an immutable object which, once created, can never be modified. A new version of a folio can be created by copying the old version, modifying the relevant fields and re-calculating the checksum. Because the hash function is collision free, the checksum will be different. Preparing a new version of a document by modifying one of its leaves thus requires making a new version of every folio which lies on a path from the modified page to the base folio, in a manner reminiscent of the Amoeba file system [7].

The protection number of the base folio (at the root of the DAG) is used to identify the current version of the document. Each folio contains the protection number of the previous version of that folio, allowing reconstruction of a linear version history[5]. Each document also has a unique name which remains constant throughout its history.

The folio is the unit of encapsulation, and so in order to maximize the opportunities for concurrent processing of the document, the folio structure should reflect the semantics of the document, with parents in the DAG being dependent upon their children. For example, in a document containing a piece of software, the DAG structure will resemble that of a unix "make" file, with folios containing header files being at the edge of the DAG, folios containing source for functions being parents of each included header folio, and the folio containing the object code being at the base.

Documents are object-based, so that access is via specific methods (e.g. replace, delete,

[4]OK it's more complicated than that. A multiplication free hash of the contents is appended to each page and the whole thing is then signed using an RSA private key to produce the protection number used to identify that page. The key ties responsibility for generation of a particular version of a folio to a specific management domain, or even to specific tamper-proof hardware within a domain. Keys are managed as described in [6]. Principals need have no access to secret keys used by their agents in remote domains, and vice versa. Of course, a domain need not include more than one principal.

[5]In this paper we assume the simplest case, that of a single, universally visible, linear history for all committed versions of a particular document.

insert, append) applied to specific folios, with only certain combinations of methods being allowable transactions (eg edit a source file, recompile and re-satisfy regression test.)

The methods associated with a document can also be regarded as text, since they are just code. Hence the methods can be encapsulated into folios and included in the document. This allows document processing to be generic and stateless. In particular, inclusion of variant folios containing method code allows easy accommodation of any heterogeneity within the distributed system, and method folios may be shared between documents of the same type[6]. There is no requirement to use the method code in the document to manipulate the document - users may manipulate the document by any means they choose, e.g. using a favourite editor. But the visible effect on the document must be just as if the document method had been used. This allows other principals (including functionaries associated with the validation process) to check what has been done.

An audit trail is formed by associating with each folio an incremental change record giving the protection number of the method executed to create the folio from the previous version. Each folio is signed by the principal responsible for creating the folio in the course of generating the protection number, and so the audit trail cannot be forged. Nor can principals avoid taking responsibility, since we may make it an integrity requirement that the change record exist.

A user may wish to delegate certain parts of the transaction (eg compilation, regression testing) to a service or another user, possibly in another domain, either because they trust them, or because doing this would absolve the delegating user from liability under the terms of the integrity contract. We therefore distinguish attribution of the principal responsible for carrying out a particular method (ensuring that the thing is done right) and the user responsible for authorizing the transaction (ensuring that the right thing is done.)

We also associate an access control list (ACL) with each folio. The ACL is also just text and so can be encapsulated and included in the document[7]. Of course, this does not by itself prevent users from preparing versions of the document which violate the access control conditions, or even from altering the access control table itself. But, since users' public key certificates are included in the ACL, any other user or functionary can (by looking at the ACL in previous, accepted versions of the document) determine whether the access control policy has been followed in the subsequent (uncommitted) history[8].

Delegation certificates (in the form of self-authenticating proxies [4]) can also be inserted in the audit trail, in order to allow users explicitly to record assumptions of trust. The propagation of delegated authority can be controlled by including propagation rights as well as access rights in the ACL.

The notion of integrity may also include discretionary elements, for example by requiring users to get their work signed off, either by other users, or by specified software checkers which act as principals in their own right. In such a case it is the presence of an appropriate certificate signed by the checker which is mandatory.

We assume in the case of each type of document that the user group (consortium) have

[6]In fact, we could go further and regard document types as documents in their own right, with relationships of inheritance and instantiation.

[7]This contrasts with the server-based ACL proposed in [8].

[8]This will require a traversal of the changed parts of the document, starting from the base, but this is necessary anyway in order to check the protection numbers of altered folios.

agreed an algorithm which (given an acceptable global starting state of the document) will determine whether or not a particular set of proposed changes preserves integrity. This algorithm can itself be thought of as a particular method, called the validate method, and the code for it can be included in a folio and placed in the document with the other methods. We assume that the base folio of any document contains, in a known place, the protection number of the folio containing the validate method.

3. Document Processing. We have placed enough information in the document to allow users to satisfy themselves whether or not a particular evolution history for a document preserves integrity. It remains to describe how a group of users may identify beyond dispute which is the current version of a particular document at a particular time, and how they can ensure (in the presence of concurrent transactions by untrusted means) that the history of the currently "visible" version of the document will always satisfy the integrity criterion.

The DODA architecture does this by defining a number of *functionaries*, which are abstractions [1] trusted by certain principals to carry out particular functions. The basic philosophy in realizing functionaries is that it is infeasible to deceive a number of independent components in different mutually suspicious domains, when each component is small and self-checking.

Let us begin by imagining a document server, which is responsible for making updates to the currently visible version of a document. Given a request containing the unique name of a document, the server will return a certificate containing (i) the document name (ii) the protection number of the current version of the document (iii) a timestamp, all signed under the private key of the server.

Upon being given a new version of a document, the server will check that the proposed new version is based upon the current version, and then will extract the validation method from the base folio of the current version, and run this upon the new version. If the validation method succeeds, then the new version will be made visible as the current version to subsequent requests.

For the moment we imagine the server as a small piece of globally trusted code running on a secure hardware platform in an inaccessible place, rebooted before each service invocation, and communicating with the rest of the world by some off-line mechanism such as electronic mail, via a gateway of a type which allows all users to monitor all traffic in and out. We emphasize that this description is an abstraction of the service provided, and that any realization with the same security properties would do just as well.

A problem which is immediately apparent is that the server must not trust the validate method code to be free from side effects. Although the users for a particular document have agreed upon the integrity method for that document, it is clearly desirable for the same document server to be responsible for many different documents, of different types and with different integrity criteria (and hence different validate methods). A malicious or inept integrity method used by one group of users may in the course of execution attempt to make visible the effects of an invalid transaction upon another document of a different type owned by a different (but possibly intersecting) group of users. To prevent this from happening, the server must run the validate method in a separate (confined) address space, passing it the document folios when requested and accepting a result when the method finishes[9].

Conceptually, we regard the document server as being made up of two functionaries, a visibility server and a validation server. The validation server is given two versions of a document, one based on the other, and runs the validate method contained in the older version upon the newer, and the visibility server checks that the proposed new version is based on the current version and then updates the version table. The validation server is required to run all kinds of code, but contains no mutable state, and so can effectively be wiped clean after each method invocation. Since each method thus effectively runs in a separate address space, even on the same validation server, users of different document types do not need to certify one another's validation code. The visibility server is not stateless, but the code which it is allowed to run is very minimal, unchanging, well understood (the visibility server is effectively a name server) and is the same for all types of document, which enables a high level of certification. The advantages of this type of separation are discussed further in [5].

If the validation server produces a signed certificate recording the result[10], then the validation server need not be co-located with the visibility server. Because the validation server is stateless, a distributed service which provided the same abstraction in parallel, perhaps using tamper-proof boxes with different keys, would be easy enough to design, and could provide higher reliability and more localized trust. Availability *per se* is not an issue : if in doubt a user can always check a transaction by running the validate method in her own domain, although of course other principals would not be obliged to believe or accept the result.

A user attempting to make visible a new version of a document must supply with it a validation certificate. The public keys of acceptable validation servers for a document are included in the base folio of the current version of the document itself. The signature of the validation certificate must be included in the request to update the document version, and the whole request signed by the requesting user.

In fact, the problems of distributing a highly reliable name server in such a way as to require only local trust are relatively tractable [2]. Ensuring that every user trusts at least one combination of validation servers, and requiring one certificate from each such server, for example, would allow further confinement of trust by users[11].

Since transactions are long term and have a low probability of conflict, an old version of the document will probably suffice for validation checking, so the visibility server does not need to be available on-line. In a cooperative environment, there is nothing to prevent users from "unofficially" sharing changes in progress but not yet committed, in order to improve awareness of any impending conflicts[12]. Users may either bear the risk of the base version for their transaction failing to commit, or may exchange cryptographic instruments which effectively commit the version history in advance in spite of the unavailability of the visibility server. In the latter case, the integrity method may even allow roll-back

[9] A way of constructing an appropriate confined environment under unix without kernel changes is described in [5].

[10] In the case of failure, the result may include a list of conflicts.

[11] The extreme case is when there is one validation server in each user's domain.

[12] Protection numbers may be exchanged by any means between parties who trust one another, for example over the telephone or in the pub.

transactions to de-commit renegade transactions.

It is worth noting that all file accesses associated with the validation server can be done by untrusted software, since the integrity of each folio supplied can be checked by the validation server upon supply. Since the public key of the visibility server is assumed universally known, certificates issued by the visibility server can also be cached by untrusted software.

An attempt to commit a document transaction may fail because the transaction was validated against a version of the document which is no longer current. However, in most cases where there is no underlying conflict a simple cut and paste of the base folio index page will allow the intervening transactions to be merged. Even if this is not possible, it is usually unnecessary to reapply all or even most of the individual methods making up the transaction, since many folios will be unchanged from the first submission.

DODA therefore includes a functionary called the submission agent whose job it is to act on behalf of the user in order to recover a failed transaction. Note that transactions may be recoverable even if they are not strictly serializable, provided the submission agent has the user's authority to vary the methods making up the transaction in the light of the new state of the document. Of course, a user may choose to act as her own submission agent (and trust no others), but this makes it more likely that subcontracting work items to principals in other domains will require the use of on-line transactions, with the associated availability and authentication problems.

A user must trust her submission agent(s), even when they are not in her user's own domain. A submission agent has delegated powers from the user, so other users must trust the submission agent at least as much as they trust the delegating user[13], or must exercise control over the user's delegation rights via the ACL.

We think of the submission agent as a tamper-proof box belonging to one domain but inserted into another domain, perhaps connected via a "suspicious bridge" belonging to the second domain. Again, we emphasize that this description is an abstraction. A submission agent may be made stateless by putting the code which the user wishes it to run in the document as a method. If the user desires, the code in the document can be encrypted under the submission agent's public key to prevent other users from examining it.

Other functionaries such as archive servers or notaries [3] can also be defined, and certain aspects of their use constrained by the integrity criterion.

The separation of concerns enforced by the use of these functionaries allows DODA to reduce the effects of the open distributed system paradox to the tradeoff between the availability and security of a small number of simple services. The local availability of document versions and of the resources to process them are completely under local user control. The lack of availability of functionaries in other domains does not have severe consequences, and in the presence of mutual suspicion this makes it easier to ensure that users can trust the functionaries to provide the intended service. This in turn suffices to guarantee the integrity of the cooperative work.

The server is not the authority over the document. Control remains in the hands of the users.

[13]Although it may be easier to monitor the submission agent.

4. What price optimism? DODA adopts a thoroughly optimistic approach to processing. Users are free to replicate, migrate, and operate upon documents as they please. The price which must be paid for this is that users have no guarantee that their work will not be wasted through incompatibility with work concurrently being undertaken by another principal. An objection which is often raised against the use of optimistic protocols in this respect is that they are less efficient than an appropriate fine-grained distributed locking scheme such as [12].

In fact, an optimistic protocol can usually be regarded as a semantic locking protocol in which all the locks are acquired immediately prior to committing, rather than earlier when it would have been more useful to know whether or not they were available. It would therefore seem that a locking scheme must always be more efficient.

This argument (valid though it is) does not take adequate account of the open systems paradox, in particular the consequences of lack of availability and lack of trust. The availability issue arises because we cannot assume that all domains are on-line to each other at all times[14]. Consequently there is inevitably going to be a trade-off between availability of the document page and availability of the corresponding (possibly distributed) lock. The trust issue arises because we cannot enforce the correct implementation of a lock controlling resources in a remote domain - if the application of a lock has implications for resource availability then it is only reasonable to suppose that a system administrator somewhere has the capability to remove the lock without following the protocol agreed by the consortium. We therefore need to have some code for dealing with the inevitable when it happens, and this code amounts to a form of optimistic integrity control.

Having established that we need optimistic control, we can take advantage of its presence to integrate into our system untrusted processing components, such as an uncertified distributed lock manager or protocols such as in those in [11], in order to gain performance benefits. Since we do not ultimately rely upon the lock management protocol to preserve the integrity of our document, we are free to use whatever software we please.

However, there are a number of reasons to suppose that the costs of using an optimistic approach will be low in the case of document applications : good semantic structuring of a document means a low probability of transaction conflict and therefore of validation failure, and since a transaction is represented as an unforgeable audit trail of recoverable work units, we can usually avoid re-performing most of the work units when re-performing a failed transaction. In addition, we can use other users' uncommitted versions of documents (at our own risk) as a basis for our future transactions.

This discussion on locking illustrates nicely a general principle in the design of DODA: we assume that untrusted (non-local) parts of the system will work correctly most times. Our lack of trust in something requires us to assume that it will go wrong sooner or later, and since we are not prepared to bear this risk we must have a means of recovery - but the recovery process doesn't need to be efficient if it will be infrequently invoked.

An objection to shrink-wrapping which we frequently hear is that public key cryptography is too slow. We argue that this difficulty can be solved by careful structuring of documents and of the systems which process them. First, shrink-wrapping a new version of a document is an incremental process. Even though the total amount of text in a docu-

[14]A domain may periodically go offline for reasons of internal security.

ment may be very large, when a document is developed only those folios on a route from a changed page to the root of the DAG need be re-wrapped.

Since our approach is optimistic, we do not require shrinkwrapping to be done or checked immediately or on every occasion, but only when we are ready to commit a version, or (as receiver) when we are unwilling to spend more resource on processing a version with unchecked integrity. There is usually plenty of spare processing capacity and idle time on systems which the local user trusts, since our transactions are long-term[15].

It should also be remarked that a large proportion of mutable information is only relied upon for performance. Such information need not be included in the protection numbers, and hence change in this material requires no cryptographic recalculation. For example, each folio must contain a list of its children (identified by protection number) together with the number of other parents possessed by each child. The identities and locations of these other parents (which will be needed if the child is updated) do not need to be protected, in the case of identity because the hint can be checked as we go and if wrong can be repaired by exhaustive search, in the case of location because we do not need to know where the correct folios were stored in order to check their integrity. We can therefore use untrusted file-management software to increase availability.

5. Conclusion. Building open distributed systems means more than just using standardized public networks to interconnect heterogeneous hardware and software.

It is easy to enter a situation in which A's best interest is served by subverting B's artifact in such a way as to make C appear responsible. A's actions may well be actions which A (and C) are expressly permitted (trusted by B) to perform. If in fact C is responsible, but can persuade B that A has framed her, then B will trust C and not A in future. C could use this argument to blackmail A into cooperating with an even worse act of subversion, without B's awareness, unless A can be confident of proving her innocence.

DODA gives users a way to ensure that conformance of jointly developed documents to a mutually acceptable integrity contract cannot be irrecoverably lost, while allowing individual users to retain control over whom they trust for what purposes, and over what they themselves are willing to be held accountable for. A user can ensure that she and her fellows must each take responsibility for their own work and cannot be forced to take responsibility for anything else. Users thus have an incentive to keep their secret keys secret. A user who fails to observe proper security precautions puts only herself at risk.

DODA's end-to-end [9] integrity enforcement provides a unified approach to preventing accidental corruption, deliberate tampering, access control violations and conflicts arising from concurrency semantics. By using optimistic control, DODA provides a high locality of trust in a way which allows free replication of data and processing functions, and in-

[15]RSA encryption can be done in software on a 10MIP workstation (using a 500 bit modulus) at a rate of about 5Kbps, which is faster than most people can type. Integrity checking involves an RSA decrypt, which can be done in software at the rate of about 300Kbps, which is faster than most people can read. Operations (such as compiling) which require consumption or generation of text by machine can be migrated to systems with hardware cryptographic support. As was remarked in [3], a user can effectively encrypt a document in software at the decrypt rate by checking the work of an untrusted Notary with hardware support, and then signing the base folio ourselves, possibly using a smart card to conceal her private key.

tegration of trusted with uncertified components. The social contract is embodied in the document structure, and the means by which the contract is enforced are also held and controlled within the document. This approach is particularly suited to multi-party software development environments, when developing software conforming to quality standards, using object oriented development and certifying modules for re-use.

DODA functionaries could be realized in the form of stateless security boxes, which would then be capable of acting as property-servers, for example providing proof of whether a transaction instrument conformed to a particular policy. Such boxes (painted in pleasing pastel colours) would be of use even in a closed supposedly secure system with a traditional hierarchical lattice structure.

Acknowledgements. Our thanks to the Distributed Systems Group at Hatfield and to the Computer Security Group at the University of Cambridge Computer Laboratory for many helpful discussions. In particular, to Roger Oliver for suggesting that document processing could be interesting, and to Bill Harbison for insisting that trust is a fundamental issue in open distributed systems.

References.

[1] T.J. Gleeson, 1990, Aspects of Abstraction in Computing, PhD thesis, University of Cambridge.

[2] P. Hu, 1994, Extensions to DODA, University of Hertfordshire Computer Science Technical Note

[3] M.R. Low, 1992, The Notary, University of Hertfordshire Computer Science Technical Report 153

[4] M.R. Low, 1993, Self-defence in Open Systems using Self-authenticating Proxies, University of Hertfordshire Computer Science Technical Report 161

[5] M.R. Low and B. Christianson, 1993, Fine-grained Object Protection in Unix, ACM Operating Systems Review, 27(1)33–50

[6] M.R. Low and B. Christianson, 1994, A Technique for Authentication, Access Control and Resource Management in Open Distributed Systems, Electronics Letters, 30(2)124–125

[7] S.J. Mullender, 1985, Principles of Distributed Operating System Design, PhD thesis, Vrije University, Amsterdam.

[8] R.G. Oliver, 1990, Protection in a Distributed Document Processing System, ACM Operating System Review, 24(2)56–66

[9] J.H. Saltzer, D.P. Reed and D. Clark, 1984, End-to-End Arguments in System Design, ACM Transactions on Computer Systems **2**(4) 277–288

[10] J.F. Snook, 1992, Towards Secure, Optimistic, Distributed Open Systems, University of Hertfordshire Computer Science Technical Report 151

[11] R. Yahalom, 1991, Managing the Order of Transactions in Widely-distributed Data Systems, University of Cambridge Computer Laboratory Technical Report 231

[12] VMS Distributed Lock Manager, VAX Cluster Manual, Digital Equipment Corporation