DIVISION OF COMPUTER SCIENCE

Towards Secure, Optimistic, Distributed, Open Systems

Technical Report No. 151

Jean Fiona Snook

September 1992

TOWARDS SECURE, OPTIMISTIC, DISTRIBUTED, OPEN SYSTEMS

JEAN FIONA SNOOK

A thesis submitted in partial fulfilment of the
requirements of the University of Hertfordshire
for the degree of Doctor of Philosophy

September 1992

This research programme was carried out
in collaboration with Digital Equipment Co.

Contents

Dedicated to the memory of my parents

Betty and Billie,

in fulfilment of a long ago promise

Abstract

This thesis is about DODA, a Distributed Office Document Architecture, designed to facilitate secure, yet cooperative, document development. It is an object-oriented system, based on the abstraction of document objects and functionaries. A document object is a structured entity composed of sub-components called folios, which may be textual or hold document methods. A document's folios may be processed in parallel, through transactions that may produce document versions. DODA combines, in a novel yet coherent manner, well-known techniques from the fields of data protection, access and concurrency control. DODA offers a unified approach to providing mandatory access control, concurrency control, version control, semantic consistency, protection against tampering and an unforgeable audit trail, in a way which facilitates the replication and local processing of document folios by a number of users in parallel.

Ackowledgements

To my family for financial support, amusing distractions, deferment activities and, most importantly, cuddles and Earl Grey; these have all been copiously supplied by Colin, Louise and Ruthie. I'm very grateful to you chaps for putting up with my studenting and domestic incompetence. The help of my outlaws, Brenda and Brian, has also been very welcome. To my parents for providing me with a goal. A conversation with my Mummy at Westminster Hospital in 1970, shortly before her death, motivated this thesis. Thanks too to my Daddy who trained Bean into the belief that giving up is equivalent to failing; it was that fear of letting him down that has kept me going.

To research colleagues (alias the Coffee Club) an excellent support group, whose capacity for winge absorbtion must be unrivaled in the known universe. In particular to Dr. Jean Baille, who lead the way, and to Karry Omer and Marie Rose Low both of whom experienced office cohabitation with my blazened walls and me. Most especially I want to thank Gordon Green (and his Grizwold!). He is able to appreciate all sides of any argument, his conversation is always fascinating and often witty, and his 3-pint joke repertoire is groanable. As if this wasn't enough, he has also provisioned me with an inexhaustible supply of postage stamps, lunch subs and sympathy. I feel extremely privileged to be able to count him amongst my friends. Gordy, your help has been indispensable and much appreciated.

To Dr. Tim Gleeson, acknowledgement reciprication! His energetic enthusiasm about most things, including distributed computing, is infectious. Pub lunches talking shop and his mail from Japan exclusively about food are among the assistance he has provided.

To SERC for sponsorship and for providing me, indirectly, with the confidence and determination I needed to finish this work. Through its involvement with the Sparsholt Graduate School, in July 1990, SERC was responsible for introducing me to a number of people who have subsequently been influential to the project. Support and guidance has been given to me since the summer school by Nick Willmer (wit, wiseman and thoroughly 'good egg'), by Chris Leiby, currently making being clever look difficult at QMC and by Davey Jones, formerly of Reading, now Doctored and employed by the University of Brunei.

To Mr. Charles Fox of Digital Equipment Co. for accepting the nomination of collaborating body and performing his duties, in the guise of Cap'n Spocko, floating down the

Kennet.

To the Division of Computer Science of what is really The Hatfield Polytechnic. In various ways the technical and teaching staff have made life quite interesting for the last few year with gossip etc. Thanks to Supervisors Bob Dickerson, for help practical, and to Dr. Roger Oliver for keeping me from complacency!

To my Director of Studies. Finally, but by no means last, I'd like to express my gratitude to Dr. Bruce Christianson, Supervisor 'extraordinaire', for his sensitive management of this research project (or do I mean me?). For his encouragement and consistent interest in the work, for his patience and persistence in cojolling my 'spaghetti' ideas into a more coherent form, for his approachability and the calmness with which he tempered my extremes of defeat and elation. In my view he is the personification of 'cool'. He has earned my respect and trust. Thanks mate!

Chapter 1

Introduction

Within open distributed computer systems there is a paradox between the provision of data security and data availability. The paradox arises because, although the system's resources are subject to a complex set of regulations and mutual agreements regulating users' accesses, these systems are open in the sense that both network and system software, including security information, are uncontrolled by any central authority. An implication of distributed openness is that system components are subject to independent failures and different management policies and hence resources are subject to differing levels of availability (as stated in [TEH+89]).

This thesis suggests that by following a path towards secure, optimistic, distributed, open systems it is possible to resolve the paradox. The path is pursued with reference to distributed (structured) electronic document processing, a field that has received comparatively little attention in the literature. The term *document* is open to wide interpretation, for example a university entrance form and a software suite may each be regarded as a type of document. However, certain characteristics are common to the apparently diverse document processing environments. These include the need for collaborative working by a number of geographically distributed users each of whom may have a particular role to play in the process, the need for any member of the user group to be able to ascertain who carried out a particular operation and the necessity to carry out operations upon a document in a particular order and to confirm that the operations were performed in the prescribed order. Thus the implications

of the work extend beyond the realm of traditional document processing to systems displaying the above characteristics. Examples of such application areas are police case documents, including suspect interview files and witness statements, and the development of legislation from inception through the reading and Committee stages to final drafting and royal assent.

The thesis describes DODA, a Distributed Office Document Architecture, which has been developed by the author to facilitate secure, yet co-operative, document development. DODA combines, in a novel yet coherent manner, well-known techniques from the fields of data protection, access and concurrency control to offer a unified approach to providing mandatory[1] access control, concurrency control, version control, semantic consistency, protection against tampering and accidental corruption and an unforgeable audit trail. Instead of taking the usual layered approach of separating concerns, the thesis advances the proposal that access control violations, concurrency conflicts, semantic consistency failures, deliberate tampering and accidental corruption should all, for the purpose of detection and prevention, be treated in a uniform fashion; namely as violations of a document-specific notion of *integrity*. From this viewpoint the thesis develops proposals for distributed document structure, and a model for highly distributed processing of such structures.

DODA is an object-based system in which the notion of a 'document' is a broad one which includes, for example, associated suites of software with related documentation. In contrast to previous interpretations (e.g. [QNA90],[PEea90], [NKCM90]), a DODA document object is an entity with predetermined structure[2] for example each DODA document is composed of sub - components called *folios* which may be textual, graphical, hold document methods or access control lists or even public encryption keys. A document's folios may be processed in parallel through single-document transactions involving the application of a method or series of methods to a folio or series of folios. A DODA document is an immutable object that progresses through the production of new versions. As such, each valid transaction produces a new document version.

To facilitate document availability DODA offers to all members of a geographically

[1] Access control is mandatory in the sense that all those wishing to make a visible contribution to a document's development are subject to access control.

[2] i.e. the structure of the document has been established prior to its instantiation cf. a wordprocessing file in which, for example, paragraphs are specified at the user's discretion.

widely dispersed document user group local and unrestricted access to document contents. The system encourages document replication by the user and local processing of document parts (folios). Read access to documents is uncontrolled. However, modification protection is strictly enforced. To facilitate document integrity DODA offers firm and unforgeable guarantees that all updates (even by other users) conform to the "social contract" established beforehand by the user group for the given type of document. DODA's optimistic approach to processing document transactions is at the heart of such provision and enables DODA to address the problem of supporting long term transactions, perhaps lasting days, and potentially conflicting parallel updates associated with documents.

Document transactions are optimistically processed. Each transaction takes place in three phases, the read phase, the validate phase and the write (or abort) phase. A document's user group specifies (in the course of specifying the document's type) specific validation criteria. The notion of serialisability that recognizes conflicts in terms only of read-write orderings is unsuitable in document development environments [HOS90][3], hence conflicts within DODA are recognized in terms of method applications to document folios. DODA's validation criteria express the "social contract" (between users of the document type) relating to transactions upon that document type. The criteria are rules that govern the application of methods to a document's folios[4] and, significantly, the rules also outline which personnel can be involved. Modification protection is provided via access control list structures associated with document folios and access control list checks form part of the validation process. Access control is augmented by an access audit scheme and a data sealing service[5] associated with transactions.

A difference between the DODA approach and that of other optimistic schemes (e.g. [Mul85], [ABGS86], [Her87], [HOS90], [Mar91]) is that the initiator of an unsuccessful transaction is provided with a list of conflicts that identifies the nature of the conflicts and names the users involved. Thus a user wishing to revise a transaction proposal is

[3]The reason being that much of the material examined by developers provides a background and if this material were treated as a read requiring serialisation, the number of apparent conflicts arising would be unacceptable.

[4]The rules cover not only operations i.e. which method can be applied to which folio, but also make clear permissible operation orderings.

[5]Each document user occupying a different security domain is provided with a single trusted functionary, a *notary*, which provides the data sealing or "anti- tampering" service.

offered significant guidance to amendments and scope for the resubmission of 'units' of a transaction without amendment.

In its current form DODA offers document availability and document security. It ensures that

- a user, say A, may only perform operations that A is entitled to perform;
- A cannot pass off or *attribute* A's actions to another user, say B, even if B is also authorised to perform the same actions;
- A cannot perform an operation that violates the document's integrity even if the operation is one that A is entitled to perform.

DODA does not aim to prescribe to a user group a particular view of document integrity but aims to provide guidelines within which a user group can reason about and implement the document type which meets these requirements. DODA offers great flexibility by making possible the provision of any of a range of integrity management schemes, each of which has associated with it a clear set of trusts and vulnerabilities. In addition, should a document object's management requirements change over time, DODA allows the provision of integrity control to be changed accordingly through transactions[6] that are subject to the stringent controls applied to all document updates.

The work demonstrates that the adoption of this unified approach to distributed processing as outlined above can facilitate data security and availability and, in addition, accommodate heterogeneity. The abstraction permits the development of a system protocol which allows data, in this case office documents, to be securely, yet flexibly, shared by the users of an open heterogeneous system. Thus, a group of users, working through a number of geographically distributed host computers, can cooperate in the preparation of documents.

A system conforming to DODA has been implemented by the author as a simulation.

[6]Document processing involves the application of methods, or more accurately method folios, to (data) folios. In this context a validation method would be viewed as a data folio.

1.1 Thesis structure

The rest of the thesis is organized as follows. Chapter 2 describes the wide ranging work that forms the backdrop for the succeeding material. The chapter reviews the topics openness, object-orientation, distributed parallel processing, system security and recovery and transaction processing within a number of computing environments, in particular software engineering and document processing environments. The review is conducted within the context of the historical development of DODA. In the summary of this chapter we initiate the argument that one effect of the traditional approach to distributed system design (typified by general purpose systems, constructed on 'divide and conquer' lines with the separation of functions such as access control and scheduling) is to produce a paradoxical relationship between certain functions within systems, such as data availability and security.

Chapter 3 introduces the DODA environment and provides the reader with an overview of the architecture by outlining the DODA notion of documents and document transactions. A more detailed examination of a document object is made in Chapter 4, in which concepts such as document semantics and type are explained along with a description of document object representation and methods.

Document integrity is the focus of Chapter 5. The various mechanisms by which consistency and correctness of document instances are maintained are described; version and access control, atomic transaction processing and encryption key management within self-protecting documents. The unified approach to the provision of document integrity is highlighted.

Chapter 6 gives a more detailed account of document processing within DODA. A system protocol, use of which ensures the integrity of document instances, is presented with a description of the system functionaries that facilitate the protocol's use. An example of document development within a software development context is present to illustrate and reinforce previously presented concepts.

Background work, first examined in the literary review chapter, is reexamined in Chapter 7 in the light of experience gained from DODA's development. This exercise leads to the suggestion of future avenues of research that may be worthy of exploration.

The final chapter, Chapter 8, contains a thesis review, in which the more novel and interesting aspect of the work are highlighted and a presentation of the conclusions.

Chapter 2

Background

2.1 Introduction

This Chapter examines the work in which this thesis is rooted. The intention is to introduce aspects of DODA, the Distributed Office Document Architecture designed and implemented by the author in the course of this research. The precursor of DODA was a centralized document processing system (also implemented by the author). It was based upon a simple client-server model in which a type specific document server managed document transactions on behalf of a user group. The tenets and limitations of this system are noted in order to illuminate the motivation behind DODA. Implications about the nature of relationships between DODA and reviewed works will be made. However, the issues raised will be re-examined in greater depth in Chapter 7, following a full description of the DODA architecture.

2.2 Centralized Document Processing

The precursor of DODA was a centralized system based on the client-server model. Within this system a document type-specific server mediated all document processing on behalf of a user-group and guaranteed that, provided the users adhered to the prescribed protocol, concurrently executed document transactions would leave the document in a consistent state. Adherence to the protocol was enforced by the

document-type$_o$server.

The motivation for the design and implementation of such a system were fourfold :

- to explore approaches to optimistic processing e.g. gain a greater understanding of the merits of contrasting approaches such as those reported in [Mul85], [Her87], [ABGS86], [ABGS86], [JM86], [WBHN87] and [BLM+86]), for example;
- to investigate access control (considered within e.g. [Mul85], [Don81], [Kar88], [Gon90], [Oli90]);
- to examine the issue of merging versions after a period of concurrent processing (raised within e.g. [Oli90], [Mar91], [HOS90]), with particular reference to document versions and
- to ascertain whether it was possible to adapt the notion of flags used in Amoeba as a form of transaction description [Mul85] to provide richer 'semantic' information (raised within e.g. [Smi92], [WBHN87] and [BLM+86], [LLS90], [GM83]) about document transactions.

The system was based upon the following set of assumptions that were derived from the background literature which will be discussed later in this chapter.

The system protocol was known to legitimate system participants eg. a description might appear within the document; the server was trustworthy, omnipresent and maintained an archive of successive document versions; each document version was protected from tampering by an encrypted protection number calculated by the server, and each user had 'authentic access' to a trusted document tamper-checker; a document-type's methods were 'good' and methods were contained within the protected document server; the server created and archived each new document version and provided the identity of the Current_version document when required; each document version was uniquely identifiable and each user and the server was able to provide a unique proof of identity to accompany communications. Additional assumptions were that a trustworthy authentication service was available to system actors, user access rights information was globally available[1] and that all proposed

[1] Access control lists existed for each field of a document.

document transactions were internally consistent.

The experimental system was an examination document application[2] in which a number of lecturers sharing the teaching of an undergraduate course could cooperate in the development of an examination paper and have it internally and externally moderated. Thus each of the users involved played a particular role in the document's processing and had particular responsibilities. The system was optimistic[3] and object based in its design. The document type was designed as an abstract data type with separate fields for each examination question and each moderator's question comments. The scheme provided access control by access control list. Auditability was achieved by maintaining an archive of committed document versions each of which harboured audit trail information. Each field of an archived version provided audit information.

The validation of optimistic processing was performed against a simple set of rules that specified what constituted an acceptable document update in terms of the actions performed and their ordering, the role player performing them and the rationality of the update in relation to the current state of the document. It worked on the basis that, for example, there was no point in the moderator taking part in processing until there was at least one examination question in situ. A simple rule set closely associated with that just described was developed, not for merging various versions into a single document version (as in [Oli90]), but for applying a single transaction representation to the current version document which may not have been the version upon which the transaction was initially proposed.

A review of the literature follows. The motivations for the design of the centralised and distributed document systems are clearly rooted in previous work. This review will be conducted on a topic basis. It must be realised however that there are not necessarily clear distinctions between areas.

[2]In this experimental system the notion of optimistic access control was not implemented. Before read access to the examination document was permitted an authentication check was performed by the document server to confirm whether or not the user requesting access is a member of the user group. If the user was not a member then read access was denied.

[3]Characterised by no locking and unsynchronised read accesses of data; each update proposal takes the form of an update to a shadow object. Concurrently produced updates undergo synchronization through a validation process which, if successful, results in a write to the persistent object.

2.3 The Literature Reviewed

2.3.1 Immutable Objects

In immutable object schemes (e.g. [Ree83], [WBMN88], [Mul85]) objects are represented by a history of immutable versions. Rather than updating an object, a new version is created and the old version is left unchanged. Since several versions of an object exist and may be available at the same time synchronisation becomes the problem of ensuring that an object is consistent in relation to certain other versions and indicating which versions of related objects are consistent. One approach taken in [WBMN88] is to group related versions into domains that are appropriately named. Such schemes are obviously well suited to support the function of version control because old versions are automatically maintained. DODA makes use of the technique of immutable objects. Its approach to document and folio versioning and the sharing of folios between document versions, can be seen as examples of domain naming.

Long term transactions are problematic because they often remain active for long periods of time e.g. in programming environments there are frequently long edit and system builds. Concurrency and performance may be severely limited by the requirement to keep intermediate results invisible outside the transaction until commit time. It has been suggested that immutable objects provide support for long-term transactions. DODA makes use of an immutable object scheme to support long term document transactions.

Mullender's Amoeba makes use of immutability within file processing. The details of this system are of particular interest and relevance to DODA, for it is the system that stimulated the earliest work of this thesis. The Amoeba file system is implemented as a tree of pages, whose subtrees are files. A file is made up of a series of committed versions, ordered in time. A version too is represented as a tree of pages[4]. The root of a page tree is referred to as the *version page*. The version pages of the committed versions form a version history[5]. A version page contains a data area and the page

[4] A tree structure of 'superfiles' may also be constructed and atomically updated, provided there is a common root. It behaves like an ordinary file: all pages of a superfile may contain data, the root page of a superfile, however, contains references to the root pages of other files and/or superfiles.

[5] Most are associated with a predecessor and a successor forming a doubly linked list. The current version's commit reference pointer and the oldest version's base reference are nil.

reference table with an entry for each child page. Thus pages have path names. Each entry contains a block number, i.e. a child page reference, and five flags that are used for concurrency control.

In the file service, principally two mechanisms are employed to implement transactions, these are *copy-on-write*, a form of shadowing mechanism, and the *flags C, R, M, W, S*[6]. These flags are worthy of attention for they illustrate, we suggest, a form of *semantic representation* of a transaction upon a hierarchical data structure. The C flag, when set, indicates that the page was copied and is no longer shared with the version on which it was based[7]. The R and W flags indicate whether the data of that page has been read and/or written, respectively. The S flag indicates that the references have been searched, and the M flag indicates whether these references were modified. Thus, the S and M flags give hints about transaction activity in lower pages of a tree. The version pages' combined flag settings convey transaction read and write information to the concurrency control mechanism. In order to determine whether or not to commit the transaction the mechanism uses this within an algorithm akin to that of Kung and Robinson [KR81] to examine whether conflict has occurred between the read and write set of this and other concurrent transactions. Further details of the scheme are found in Mullender's thesis [Mul85]. It is this primitive notion of transaction representation that motivated development of a more sophisticated form of transaction representation in DODA by reference to more high level actions than simply read and write, i.e. the examination of method applications to an object's data.

Several points about the Amoeba file service are worth noting. It is underpinned by a communication mechanism (known within Amoeba as the transaction mechanism). This forms the only means of communication between users and the file service. Its use is intimately linked with the capability-based access protection mechanism within Amoeba and with the page and disk block writing scheme and the identification of concurrency conflicts. In this light Amoeba may be described as taking a unified approach to these issues. However, it is not thorough going in the approach. Consider

[6]Additionally soft-locking, i.e. an advisory lock, is available. These are intended for use in transactions that span many files or superfiles, because such transactions have the greatest probability of non-commitment using optimistic control.

[7]When a client starts a transaction, the version created shares its page tree with its base version, until a page is to be changed. Then the page is duplicated and altered.

the use of flagged information for example. A page's flags are stored in that page's parent page. The root page, without any parent, is therefore the only page whose flags are kept in the managing server. With the flags for this page readily available to the server, it would be possible for the mutual exclusion information the server possesses to be passed on to the capability checking mechanism and hence to implement mutual exclusion within the framework of access control. Such an approach is pursued within DODA. For even though a user may possess the rights to initiate a transaction, that right can only be executed under certain conditions[8]. This, we will suggest in Chapter 7, represents a loss of opportunity[9].

The above work was formative in the development of DODA for an examination of the detailed working of the file service suggested a number of ideas in relation to the use of immutability and optimistic processing which required the development of validation criteria. In particular the wish to explore 1) the possibility of representing the semantics of a document in the storage structure used for sub-components, the access paths to sub-components themselves being of semantic significance, 2) 'semantic flags' in transaction description to be adapted for use in document concurrency control and 3) prolonged existence of each transaction's description as part of an audit trail.

2.3.2 Semantics

Smith [Smi92] takes semantics to be "the meaning implicitly or explicitly represented by data. In particular, data semantics involve the static and dynamic properties of the structure and contents of data objects and the relationships between those objects". The DODA notion of document semantics conforms to this definition of semantics. The use of semantic information to assist in application processing is not widespread; an explanation for this is offered by Garcia-Molina [GM83] who says that making use of semantic information within concurrency control, for example, will only be useful in certain applications. We suggest that the processing of structured documents is one such application.

Walpole et al. [WBHN87] suggest that semantic information can highlight the

[8] With regard to the version capabilities, each must be created when the version is created by the copy-on-write mechanism.

[9] Dollimore et al. [DMX91] also recognised this.

situations in which a data type can remain consistent after concurrent access; a simple example is the use of read and write locks; the presence of a read lock need not inhibit concurrent readers.

COSMOS is a software engineering environment that utilises semantics to support what is called *typing* [BLM+86]. Typing assumes that only certain operations are defined for a particular type of object[10] and that not all the defined operations will be suitable in all object states[11]. Typing examines the validity of operations being performed in a particular context and as such can be viewed as a form of semantic checking of transactions. The benefit of type checking is that it represents an "insurance of sensible actions", i.e. it provides a form of protection against actions that make no sense. The COSMOS typing system influenced the notion of validation adopted within DODA. DODA's validation checks incorporate not only concurrency control checks but also provide "insurance of sensible actions" for users.

Semantics are also associated with a *coercion mechanism* in COSMOS [BLM+86]. This mechanism can 'rectify' some forms of mistake made by a user in her expression of a transaction e.g. should the context of a transaction require an object of type X and the user specifies an object of type Y, coercion may search a rule base for details of a relation which will map type Y into type X. If such a mapping exists then the related object (type X) can be derived from type Y and substituted for it in the transaction request. An interest in this mechanism is reflected in validation and commitment of versions within DODA, in particular in the application of a transaction proposal to a version other than the version upon which it was originally based.

Ladin et al. in [LLS90] suggest that the use of information about an application's semantics can lead to increased availability of that service. Though their notion of semantics is not overtly described, it is possible to discern the view that an application's semantics can be expressed, to some degree, in the nature of the invariants and operation ordering restrictions imposed upon an application's processing. For instance, their concurrent system provides three different forms of update operation, each of which has associated with it a different serialisation requirement (e.g. one requires that such update are processed in strict serial order) and when an object is

[10] e.g. read and write are defined for text files but it is illegal to read a printer.
[11] e.g. printing an empty text file.

instantiated within the system the types of updates permissible in the object must be defined on the basis of the object's semantics. This is a view similar to that adopted in this thesis, for example an update to a comment field of a document would have less restrictive rules associated with it than an update to a bank balance field.

Garcia-Molina, in [GM83], similarly investigates the use of semantic knowledge of an application. Unlike Landin et al. who are considering a range of application types, Garcia-Molina's investigation was restricted to transaction processing within distributed databases. The aim is to produce transaction interleavings (schedules) in a concurrent update environment that conform to a notion of *semantic consistency* rather than by the conventionally notion of serialisability which considers schedules only in terms of the ordering of read and write actions [KR81]. Semantic information received from users facilitates the production of such schedules. The information required concerns the type of each transaction e.g. update or delete, how to divide each transactions into steps, compatibility sets (i.e. sets of actions that are compatible when interleaved) and countersteps (i.e. how to un-do actions). Thus Garcia-Molina implied interpretation of semantics relates to those application features. The DODA view of document semantics does relate to similar application features, although 'countersteps' is not a relevant concept to DODA.

The issue of how an application's semantics may be expressed has become a topic of particular interest to DODA. Clearly the principal burden is upon the user. COSMOS takes the view that access rights are integral with object semantics [BNY86], [WBMN88] and Smith, in [Smi92], discusses security-relevant data semantics. He advocates the use of secrecy semantics and its expression through rules called *secrecy constraints*[12]. The details of Smith's work have no direct bearing upon DODA, for security is seen by Smith in the rather restricted sense of secrecy. However, these works do suggest a precedent for the DODA approach of considering the security and access control requirements of an application as aspects of that application's semantics.

[12]Secrecy constraints are used for classifying data and combinations of data into the hierarchical sensitivity levels of data and user clearance i.e. into Unclassified, Classified, Secret and Top Secret.

2.3.3 Security

Smith [Smi92] suggest that the definition of security may be application specific and should therefore be considered as an attribute of a particular application. DODA concurs with this and the thesis takes the view that security is an aspect of application semantics.

Security serves not only to prevent unauthorised users from accessing a system's resources but also to prevent authorised users accessing services in an unpermitted manner [TKS88]. Although system security research has concentrated on the former aspect of security, namely protection in a 'hostile' environment e.g. [Kar88], [Gon89], protection is highly desirable even in a system where all users are honest, to prevent the consequences of mistakes. This aspect of protection is particularly relevant to a distributed system in which benign users flexibly share data, documents for example, and concurrently progress. DODA's optimistic approach to processing that requires transaction validation even in the absence of concurrency has been developed to address this issue.

Clark & Wilson [CW87] state the view that handling integrity rather than confidentiality is the most important security problem facing a commercial organisation. This is a view of great relevance to the work on DODA. Stimulated by the work of Clark & Wilson, the National Computer Security Center has suggested, in report [Cen91], that "integrity denotes the goal of ensuring that data has at all times a proper physical representation, is a proper semantic representation of the information and that authorised users and information processing resources perform correct processing operations on it". This broadly is the view of integrity that DODA represents.

Protection

Secure systems provide some form of protection of data. Cryptography is the popular means of maintaining secrecy and, more importantly from the perspective of future discussions, the integrity, of material [Nee90]. An account of the general criteria for encryption appears in [PS83]. Two categories of crypographic tool exist, namely symmetric or secret-key systems and asymmetric or public-key systems.

With secret-key systems two users wanting to exchange cryptographic information must share a common key. Hence, the users must be able to exchange such keys through some key distribution system over some secure channel prior to communication. Key distribution is problematic. As such the key cannot serve as a digital signature for a message, as it is not a unique identifier, either of the two parties knowing the key could have signed the message.

Public key systems have been devised to remedy the key distribution problem. Each party selects a pair of cryptographically related keys, one of which is kept as the party's private key, while the other is advertised to any potential communicating partner as its public key. This latter key is public information and is shared. Because secrecy, integrity and authentication are all based on pairs of keys of which one is always public, there is no need for secrecy in public key distribution. Only integrity and authentication of public keys are required, which are achievable through digital signature of keys by a common trusted authentication service. Cryptography is used within many access control schemes, for example in the preservation of login passwords by an operating system.

Access Control

Access control should ensure that directly or remotely connected users of a computer system cannot read, copy, modify, destroy or use information resources unless they are authorised to do so [MT84]. DODA does not aim to prevent read access of documents, simply to safeguard instances from any uncontrolled change.

Most access control models are based on Lampson's access matrix, in which the rows of the matrix represent the *active entities* (subjects or domains) and the columns represent *passive information-containing objects.* Its interpretation can be quite complex and such information is generally stored in one of two ways, as *capabilities* or in *access control lists*, ACL. The use of both capability and ACL forms of access control relies upon secure ways of addressing messages and authenticating receivers. The availability of a foolproof authentication service is therefore fundamental to the design of a system that enforces access control. However, the means by which such provision might be made in DODA is beyond the scope of the presented work.

Capability-based control

The traditional view is that a capability is a way of addressing a particular instance of an object. It is analogous to having a key to a particular lock; the lock will been unlocked by the appropriate key regardless of the identity of the key user. In Amoeba, for instance, a capability to a resource is just a large number, and knowledge of this number is taken as prima facie evidence of the right to use that resource [Mul85]. Thus, in this view a capability is addressing knowledge shared by a limited number of users.

There are two predominant problems with traditional capability systems, as stated in for example [Gon90][13]. The first is that the use of capabilities (though not the initial receipt) is user independent, while the second is that the set of access rights represented by each capability include the right to grant access to others by passing on (a copy of) a capability. Thus, capabilities can propagate and be used freely around the system. On the one hand this can provide flexible sharing of objects, but on the other makes it impossible to ascertain who does and who does not possess access rights to a particular object [KH84]. Hence, the use of such capabilities makes auditing impossible to implement, a point also appreciated by Oliver [Oli90]. He suggests controlling capability propagation (in a document processing system) by including within each capability an *authorise rights bit*, set only for those authorised to confer rights to others.

Gong suggests three alternative ways by which greater control of access rights may be obtained, namely

- *prohibit free propagation*
- *prohibit free access but allow free propagation*
- *prohibit both free propagation and free access*[14]

Gong's ICAP facility provides the latter form of capability control which he regards as superior to providing access control tables via an operating system. However, ICAP

[13]These have led to the description of capabilities as "representations of necessary but not sufficient conditions for access" [Nee90].

[14]The Oliver scheme is of this ilk.

is not an optimistic system but is concerned with read protection of data. Thus its aim is rather different from DODA's. Also Gong indicates that the provision of a capability mechanism that provides functionality equivalent to that of ICAP entails costs over and above those required for access control lists. The costs arise from secure storage, forgery protection, propagation and revocation of capabilities.

Oliver [Oli90], in a paper that provided the initial inspiration for this research, speculates upon how certain types of objects, documents, can be protected in order to provide decentralised processing. The suggested solution involves multiple similar servers that each support multiple document types; each server is a composite of multiple document types' servers[15]. Documents are protected from unauthorised access by *capabilities or access control lists* (ACLs)[16] and encrypted checksums and are passed freely around a distributed, heterogeneous computer system.

Oliver suggests that to permit parallel processing a document may have several versions. There is a notion of ownership of a document instance. A document's *possessor* can request that a document copy, with a different set of access rights from those of the original, be made for a named individual by a server. This is the way in which cooperative development can occur. Duplication is achieved by presenting a legal version of the document together with the requestor's user-id and password, rights modification details and the user-id of the named individual to a server. However, as access rights may need to be recoverable he suggests that rights restrictions should be considered as the document's owner temporarily "masking out" access rights for which he is responsible.

Dollimore et al. [DMX91] have designed a platform for building applications that are intended for use by a group of people who share information and communicate with one another. Thus it is a system with similar strategic aims to DODA. The requirements of such a system are identified by them as

- placement of shared objects in any workstation that runs their software;

[15] Although this is not because the server is generic but is a composite server encompassing a collection of document type servers.

[16] It would be possible to store an ACL for an object within the object. The ACL would be included in the calculation of the checksum, making tampering with the ACL detectable, thus removing the need to maintain access information about every type instance on every type server.

- location and access transparency;
- an interactive user interface for presenting information to users, necessitating replicas of shared objects in user's workstations;
- concurrent viewing and editing of objects by users and the ability to observe one another's effects;
- object consistency during concurrent processing;
- privacy and protection;
- long-term reliable storage - transparent persistence.

In some respects the approach that has been adopted resembles DODA. For example the scheme allows concurrent development which makes use of a form of visibility management. Changes are made visible to concurrent object accessors by a scheme that uses defined object *dependencies* linked to a message broadcasts. When a user changes an object, all other users with access to that object are informed by an *object changed message.* Within this scheme (as in DODA) objects are made available to users concurrently and the concurrent update of objects is mediated through the access control mechanism of the system. Protection and privacy are provided by capabilities (c.f. ACLs within DODA) that act as message filters forwarding permitted messages i.e. those referring to objects that this user is currently accessing, and reject others. However, the Dollimore et al. scheme does not provide 'formal' consideration of the process of change; changes are simple made rapidly visible to the user group who, presumably, will realise and rectify 'unwise' updates.

List-based control

ACL's facilitate traceability, confinement and revocation of access rights more readily than capabilities; these are the main advantages cited by Reiter et al. [RBG91] for ACL's and arise from the fact that ACL's offer person-based rather than process-based protection. The ability to trace actions is central to DODA, therefore ACLs would appear to provide a suitable means of control. The essential operational difference from capabilities is that access controlled by ACL requires authentication of the accessing user at the time access is actually required. Thus the efficiency, in terms of

response time, of a system upholding ACL control may be poor in comparison with a capability-based equivalent. In an optimistic system such as DODA this operational factor is not a disadvantage to the use of ACLs.

ACLs make the provision of access control of different granularities. By this we mean that an ACL can offer coarse-grained protection e.g. at the UNIX file level and can also be designed to provide fine-grained access control e.g. to provide different access permissions for a user to each method of an object. Such a distinction between coarse-grained and fine-grained protection is made in [Low92].

2.3.4 Open Distributed Systems

Tschammer et al. [TEH+89] suggest that distributed systems have many potential resources and services, a heterogeneous community of service users and providers and a variety of overlapping organisational structures. Systems of this type can be termed *open*, as they usually have no restrictions on the number, type and behaviour of components and are ready to admit any customers, clients and users[17]. Further to this they state that 'open' is a synonym for unlimited[18], accessible[19], heterogeneous[20], autonomous[21] and decentralised[22]. DODA takes this view of open distributed systems and in addition points to independent failure of nodes as a particular problem because of the consequent unavailability of data from a node.

Blair et al. [BLM+86] highlight the same problem in a critique of the client-server paradigm and identify the cause as the division of control. The conclusion is that the client-server model is an inadequate basis for achieving highly available and extensible systems because the server itself is a potential bottleneck. The remedy for loss of access to data due to server unavailability, namely the replication of data and offering redundant copies at multiple sites, raises the problem of mutual consistency of replicas.

[17] Although local conditions may require that local components stay under local control.

[18] Unlimited means that there are no restrictions on the number and geographical locations of the components.

[19] In that nobody is restrained from occupying system resources, this represents a theoretical approach only.

[20] Each organisation within the open system has the freedom to install and use its own local resources.

[21] Different locations operate under different authorities.

[22] All the capabilities are distributed amongst the components.

This can be an extremely complex problem, the complexity arising in part from the use of *update-in-place*. The use of immutable objects[23] and devolution of resource control to a set of cooperative *comanagers* offers an alternative approach [BLM+86]. This approach is clearly visible in the design of DODA that makes use of multiple 'functionaries' in the integrity management of documents.

When discussing DODA it will become clear that DODA is an application, neither part of nor dependent upon an operating system and that DODA provides conceptual 'building blocks' from which a user group may construct a tailored document processing system. The approach has been taken as a response to the discontent among practitioners caused by the division between general-purpose[24] and special-purpose systems[25].

2.3.5 Abstraction

Gleeson's ideas [Gle90] have influenced the work upon DODA. Thus a holistic approach has been taken to the abstraction of the system; an approach supported by the notion of object-orientation. In principle such an approach is not new. Blair et al. [BLM+86], for example, favour a holistic approach to application design. Object orientation has been suggested as a means by which to formulate such an approach e.g. COSMOS [WBMN88], Timewarp [JM86]. The object oriented paradigm is characterised by objects [Jon78]. Each object is defined by :-

- its type;
- the operations permitted on it;
- the access rights and/or restrictions associated with the object and with the operations;
- semantic information pertaining to that object;

[23] An immutable object can never be updated; the application of one or more of the operations defined for the object can produce new objects.

[24] Characterised as overhead intensive [Lam73], less responsive in any given situation than its application-specific counterpart but allows greater flexibility as a 'bulk' of software exists to meet a range of possible requirements.

[25] An application specific system can be optimised to perform very efficiently within its own domain, but may have little built-in flexibility.

The model has been enhanced in a variety of ways e.g. to provide semantic information about the inter-relationships between objects [WBMN88]. The object oriented approach is that adopted within the document processing simulations produced during this work.

Horn in [Hor89] assumes that to unify distribution, immutability and general programming within object oriented systems is a desirable goal and raises the issue of whether it is possible to build an integrated development environment based on object invocation. DODA can also be seen to address this question and DODA's development suggests that it is also an achievable goal within the area of document processing.

2.3.6 Recovery

Recovery relates to the recovery of data objects after errors and system failures. For systems that process transactions, for instance, the recovery problem is concerned with returning the system to the last consistent state (rollback) or advancing the system to the next consistent state after a failure or error has occurred [Her85]. DODA makes use of immutable object versions and optimistic processing to avoid the difficulties of rollback. Each object version represents a consistent[26] state of a document and only successfully validated document transactions can change this state. Optimistic processing uses a sophisticated form of intentions list, a doculett. The application of a transaction to a document version is regarded as the process of providing visibility to the transaction's results. DODA guarantees the isolation property of transactions (see Section 2.3.7) because the results of a document transaction are made visible by a single version reference change within the trusted server.

Reed [Ree83] and Walpole et al. [WBHN87] make the prescription that synchronisation and recovery mechanisms must work together. This is the approach that DODA takes. Jefferson presents an exciting example of synchronisation and recovery mechanisms working together in Timewarp [JM86]. Like DODA Timewarp employs optimistic concurrent processing[27]. The detailed operation of Timewarp has

[26] Each current document version in DODA actually represents a state of integrity of a document. This is a stronger guarantee of 'fitness for processing' than consistency.

[27] Paradoxically Timewarp uses a very restricted notion of serialisability, namely serial ordered of transactions as designated by the system's (virtual time) clocks.

not directly influenced the design of DODA. However, its recovery mechanism has inspired one feature of the document system; namely the submission to validation of a transaction proposal that is constructed from some 'units'[28] that appeared in a prior (unsuccessful) transaction validation by that user.

Jefferson [JM86] describes objects. In Timewarp objects perform processing in response to received messages. All messages are timestamped with a *virtual time* and arrive asynchronously. There is no guarantee that messages will arrive in the order that they were sent, but messages *must* be received (i.e. read and processed) in increasing timestamp order. Messages do arrive at objects out of order. Such arrivals initiate a recovery process known as *rollback*, during which a *timewarp* occurs. The system is returned to a virtual time just prior to the occurrence of the anomaly and all objects affected by the timewarp[29] are returned to the state each held at that virtual time. Once rollback has completed[30], processing recommences and virtual time runs forward. *Lazy rollback* is the process relevant to DODA. It is an optimisation that minimises the quantity of processing necessary during re-execution by implementing selective cancellation of previous processing. In Timewarp, if the same results are produced during execution and re-execution of an action at an object (i.e. locally) then the results of the original execution are not cancelled globally. Re-execution is efficient because it executes only the 'difference' between the original and re-execution paths. This has an obvious parallel within DODA.

2.3.7 Transaction Processing

The traditional view of a transaction is that it is an ordered sequence of processing actions that are performed as a single unit and therefore displays the ACID properties[31].

[28] A 'unit' in this context means the description of processing in which a named method is applied to named data folio(s), the application being attributed to a named user and the entry finally being protected with an encrypted checksum.

[29] i.e. all objects having 'later' virtual times.

[30] By the cancellation of all processing associated with timestamp in advance of 'timewarp' time.

[31] The ACID properties are defined as those of
Atomicity in which either all or none of the component operations are performed and contribute to a persistent object state;
Consistency whereby the transaction takes the system from one consistent state to another;
Isolation in which an incomplete transaction cannot show its results to other transactions unless all consequences of that transaction's abort (should it occur) are also aborted;
Durability is concerned with time and ensures that transaction results are persistent.

However, work on abstraction by Gleeson [Gle90] has suggested that such a definition represents a widespread misunderstanding of the concept 'atomic transaction' because the definition focuses on implementation issues. Any system is atomic, in his view, if it is an abstraction and it is thus an attribute of a specification not an implementation. Atomicity, being an abstraction, is a relative concept. Hence, for example, a transaction is atomic relative to its actions and a database is atomic relative to its transactions. DODA uses the term transaction to describe document processing and sees no incompatibility between the two views expressed above because transactions can be directly represented within the specification (and implementation) of a document. In particular the folio structuring facilitates this. Folio structuring, known within DODA as foliation, provides for the representation of associations between folios. Folios may hold data or the operation(s), methods, that may operate upon the data. Thus by choosing an appropriate foliation it is possible to associate the data and method folios that comprise a particular transaction.

Pessimistic and optimistic concurrency control techniques (see for example [CD88] for an overview of these techniques) are the two contrasting approaches to the coordination of concurrently executing processes. The most simple pessimistic algorithms e.g. locking mechanisms [EGLT76], perform synchronisation between processes before accessing data objects on the pessimistic assumption that transaction conflicts will be frequent. Locking effectively prevents concurrent access to the data item upon which the lock is placed. Information about the lock must be stored and accessible to the concurrency control mechanism of the system. Such locking prevents inconsistent updates of data, however it also prevents some concurrent actions that could be interleaved e.g. multiple read operations on data. Thus the technique can be said to display the following disadvantages

- *lock maintenance* represents an overhead in all cases except when conflict occurs;
- *unnecessarily restrictive* because locking may be necessary only when concurrent transactions process the same data item;
- *reduce concurrency* as locks must not be released until the end of a transaction to allow transactions to be aborted when there are errors;

Optimistic algorithms e.g. [KR81], Timewarp [JM86], [ABGS87], allow processing (i.e. transactions in which there may be multiple actions involved) to execute freely. Concurrent processing is synchronised after data objects have been accessed. Thus such algorithms operate on the assumption that transaction conflicts will be infrequent and hence rarely will the work involved in the transaction be wasted. Only transactions that conflict are involved in synchronisation. Optimistic schemes are most successful where transaction conflicts are low and the cost of re-doing computations after conflict has been detected is minimal. DODA uses an optimistic approach to processing presuming that conflicts will be rare since users are provided with the means by which to monitor a document's development. We therefore assume that users will propose appropriate transactions. DODA also minimises the costs of reapplying a transaction following the identification of problems.

2.3.8 Software engineering environments

Within software engineering environments there is a requirement for version identification and management because software development frequently necessitates the building and rebuilding of various software system configurations from various versions of component modules.

Within the software engineering environment COSMOS [WBHN87] version control is provided by placing related versions e.g. those from which to built a particular configuration, within a *single named domain.* A particular domain and its associated objects are explicitly named and thus can be requested as input for transactions within the system. Likewise, the output of a transaction constitutes a new domain to which a name is attached. An initiating transaction creates a *transaction object* that holds state information about a given transaction[32]. It can be *opened* and *closed*; closing creates a *save point* by causing the partial results of the transaction concerned to be saved to a new version of the transaction object. Opening restarts the transaction from the specified save point. Creating a new version of a document instance within DODA is similar to versioning in COSMOS in that each document version can be

[32]Such an object remains private to the transaction's initiator until the transaction is committed and stored. Transaction objects can be stored in a database indefinitely to assist long-term transactions.

viewed as a domain that bears a unique name that links the appropriate versions of each folio.

The approach to synchronisation of concurrent transactions within COSMOS depends on the type of objects concerned. This is where the similarity with DODA ends because COSMOS recognises the type of an object, for the purpose of synchronisation, in terms only of whether it is free branching, controlled branching or linear[33]. The three types can be subject to different forms of concurrency control. At one extreme free branching objects are processed completely optimistically with no synchronisation. In contrast the strongest form of concurrency control, that exerted upon linear objects, is implemented in the form of reading and writing locks. Thus what might appear at first sight to be a system that makes use of object type (or semantics) in decisions about concurrency control does so to only a very limited degree and determines transaction conflict at the primitive level of read and write operations. Like Amoeba, COSMOS economises on the storage of multiple versions of objects by allowing different versions of an object to share common disk blocks. Only when a particular disk block is altered, is a new disk block allocated. DODA's approach to folio sharing is similar in principle. However, in practice DODA takes a semantic view of data rather than viewing it at the level of disk block reads or writes.

Linton [Lin87] describes Allegro as a decentralised, distributed, object-oriented approach to managing information during software development. Although it is a distributed, object oriented program development facilities like COSMOS, in other ways it is in contrast. It shows a commonality with DODA because it considers an object's protection and processing requirements within its approach to object storage. To speak of Allegro in DODA terms, this system attempts to express (some of) the

[33] COSMOS object types are

- *free branching* - no synchronisation is necessary. Branches may be created unintentionally by concurrent transforms; each configuration will create their own version branch but is intended for use for configurations that are not shared;
- *controlled branching* - branches are created intentionally and attempts to transform a single configuration object concurrently result in a warning. To discourage concurrent transformations an advisory lock is applied. It does not prevent other transactions from reading the locked configuration but should a transform be attempted a warning is returned. Ignoring the warning will create a branch in the version history of the configuration;
- *linear objects* - for some conceptual objects, it is desirable for all users to see the same and most recent version in the version history. The object is therefore locked for both reading and transformations. This type of lock cannot be ignored by other transactions.

object semantics in the structure of the stored object.

Within Allegro, objects are grouped into *object spaces.* Criteria for partitioning objects into spaces depends on the specific needs for protection and processing efficiency. Objects with many inter-relationships that as a whole represent a more complex object are usually stored in the same object space. Object spaces are organised hierarchically and objects in one space may refer to objects in another space. Thus object spaces can be organised to share information. The organisation of object sharing produces object organisation graphs which show the cross-references made across object spaces.

Harrison et al. [HOS90] uses immutability, provides concurrent access and supports version merging through the concept of *change-serialisation.* This concept does not use only read and write sets but attempts to use more rich application semantics. Like Garcia-Molina's scheme, conflict resolution is provided by the directed involvement of users. Access control is not a consideration within this system.

Harrison's system provides a software engineering environment in which software is divided into *artifacts*[34], each consisting of a set of files kept in a store called the *master store.* This master store is equivalent to the current version document instance of the DODA system. A modification activity is a set of changes, made in isolation in a separate store i.e. modification is conducted upon a shadow object. Multiple modification activities can occur concurrently, each in its own store. This too has parallels within DODA, however DODA does not use shadow versions but expresses changes in terms of a doculett. For the changes to become visible outside its particular store, that store must be *merged* with other stores. Ultimately, all changes that are to become part of the artifact must be merged into the master store and become visible to system users. Thus, in DODA terms, Harrison et al.'s system is optimistic in that concurrent changes synchronise once they have been formulated. However, synchronisation employs the use of locks.

Development consists of modification activities and merges. Concurrent updates are coordination by means of protocols to lock stores during the merge process[35]. Merges

[34] An artifact is the equivalent of an Allegro object space, a COSMOS configuration object and a DODA document instance.

[35] Such locking is of two types, namely

- *strict locking protocol* - guarantees that all merges are safe by preventing concurrent modifica-

provide synchronisation by enforcing what is termed *change-serialisability.* This simply means that no change is overwritten by a parallel change without consultation with the user. In this way one change does not accidentally supersede another, although a change can be deliberately superseded. Overwriting takes place as a modification activity that is recorded. Change-serialisability permits greater concurrency and is weaker than read/write serialisation because it makes no statement about files that are not themselves changed yet are examined in the course of making a change. Hence, it is possible to base a change on read material that has itself been changed. This ability is also present within DODA.

The DODA notion of an audit trail sees its origins in this system's notion of a *modification history.* A file consists of a file name, a modification history and the file contents[36]. A modification is a function from an input to an output file and a modification history is a sequence of modification identifiers that are unique and serve to trace all modifications to a file since its creation. Each file can be described as having an associated (if limited) audit trail, although this does not record the name of the user that conducted each modification. The trail is used during merging of versions, as in DODA.

Marsh's work also addresses the issue of merging versions of an object following concurrent processing. In a paper entitled "The V Project Manager Tools" [Mar91], he describes a series of tools for managing concurrent accesses to a series of files by many developers. The tools may be added to a software development environment. The interesting aspect of the work, from the perspective of this thesis, is the provision of a conflict resolution tool and visibility control. Again updates are proposed to copies of visible files that reside in an *origin world.* This world can be concurrently accessed. When a user wishes her changed files to become generally available they must be integrated into the origin world. *vresolve* is the tool that identifies inconsistencies between worlds by comparing the given development world i.e. the set of changed files, with the origin world. The tool interactively scans the development world checking

tion. Requires that before a file is modified, it be locked successfully in all stores into which it might later be merged;

- *lenient locking protocol* - risk of collision deliberately chosen by developer to avoid holding up work. Warns file changer that the file is locked by someone else.

[36]Thus a newly created file with null contents has null modification history.

that the file versions on which changes were based coincide with those of the origin world; it also checks whether new files exist within origin world which may not in the development world. For each file that requires some resolution, all inconsistencies are expressed to the developer. Each one must be addressed and the developer is given a series of alternatives to resolve each inconsistency.

The similar approaches to file conflict resolution described by Marsh and Harrison et al. are not heavily prescriptive. Users are alerted to apparent conflicts and able to make, within limits, their own assessment of the necessary steps to resolve problems. Thus the resolution of update conflicts is seen as a tractable problem. However, no attempt is made to record any information about the way in which users make decisions regarding conflict resolution. Such information, once formalise in some way might be used as a knowledge base for future automated conflict resolution.

2.3.9 Document processing systems

Horowitz and Williamson [HW86] have produced SODOS for Software Documentation Support. Of particular interest from the perspective of DODA is their generic (within the context of SODOS) definition of a document.

SODOS documents are the means by which the system structures and organises the activities, milestones and deliverables of the software development process. Recognising the fact that different software development teams and projects approach the software development process in different ways, SODOS supports the definition of documents to be used in software development. Each document represents a particular stage in the Software Life Cycle (SLC). The descriptions, provided by the project team, of each phase are managed as structured documents. Documents can be manipulated.

The document is represented as a complex object with internal structure and well defined relationships with other documents used within the SLC model. Each SLC document is represented as an instance of a document type e.g. design specification, implementation notebook and user manual documents are predefined types that are supported within SODOS. A Document Administrator is responsible for providing

further provision within the system by tailoring document definitions by subtyping. Thus the SODOS document administrator plays a similar role to the user group within DODA in relation to document definition. Previously defined document types may be inherited (i.e. used as templates in the definition of new document types) and particular document instances may be shared e.g. SLC stage definition documents. Each type contains a description of the structural representation of relationships among the components; a *document interface relation* defines its relationships with other document instances. A document's structure and relationships are represented in SODOS as a graph structure.

The SODOS notion of a document has strongly influenced the DODA notion of a document. This is particularly apparent from SODOS's use of document semantics for example, for a document's semantics provide the basis for the relationships between the various document components of a SLC document e.g. a system requirement document is related to a functional requirement document by the 'derived from' relationship; the functional requirement is related to a design module by the 'required by' relationship.

Though not the main focus of interest in [Oli90], a form of decentralised document transaction processing is suggested. In the proposed scheme, multiple distributed servers support multiple document types. A document may have several versions permitting parallel processing with optimistic concurrency control.

A document object, once created, can be passed freely from user to user by various means e.g. E-mail, file transfer, though one user plays the primary role when completing a document. Users fill in fields. If fields must be filled in a particular order, by particular personnel. The possessor of the document directs the process and after appropriate access rights 'masking' (see Section 2.3.3 for details of this stage), passes the document to the first required contributor, for the first required entry, etc. When operations are not inherently sequential, a server is requested by the possessor to produce multiple copies of the document, each with appropriate access rights, in order that parallel processing can take place. Completed document copies are finally 'merged'. Objects are presented to an authorised server when confirmation is sought

that an existing document is valid[37] or when a supported operation is required e.g. a merge.

Neuwirth et al. discuss, in the paper [NKCM90], how the co-authoring and commenting of technical documents is handled by the PREP editor. The main interest of the work in relation to this thesis is its notion of document processing. Users developing the document are assigned particular *roles* within the development process. The approach is to identify and provide assistance for "the social, cognitive and practical issues of cooperative document development".

PREP provides support for social interaction among co-authors i.e. document developers. The multi-user environment of PREP uses hypermedia and makes possible a variety of collaborative relationships. In order to co-ordinate collaboration *social roles* can be definition (and redefinition) e.g. the roles of author, co-author and reviewer. Redefinition of roles may be useful when, for instance, for a co-authored document it is not clear at the outset of a project when roles are defined, who is going to make a significant contribution and therefore who should get the authorship role. This simple approach of defining roles in order to reduce the coordination problem is particularly interesting because it can be seen as a form of access control by which to prevent certain developers from performing certain tasks. DODA has adopted this approach also and has developed the notion of access control as a form of concurrency control.

English et al. have developed the Interleaf System [PEea90] as a runtime-extensible, object-oriented system for describing and executing *active documents*; active documents are taken to be "structured documents and their processors in which the objects in the document can be acted upon by, and can themselves act upon, other objects in the document and the outside world". DODA is therefore an active document system by this definition because each DODA document is a structured entity that holds its own methods and these methods are used to manipulate the document's subcomponents, its folios. The Interleaf system shows some striking similarities to DODA in its notion of document structuring and has been developed contemporaneously with and independently from our work. The scheme is quite different from DODA in implementation.

[37] Valid means the contents match the checksum.

An Interleaf document is a structured object, or composite object[38]. A document objects is equipped with default methods for interacting with other document objects, its environment and for computing its own contents. As a document is a heterogeneous collection of document objects, *navigation methods* are provided for manoeuvring about the sub-objects.

Documents are a very general paradigm of communication. The provision of subtyping i.e. providing different functionality to a particular document, permits an application to be tailored or optimised for a particular application area. Carrying methods within the document is regarded by English et al. as a very powerful facility because these can provide type specific operations. Permitting these methods to be defined at document execution time allows the document to adapt to changing circumstances.

The system is not open in the way DODA interprets the term. Interleaf is built upon a LISP interpreter embedded in the distributed software. The LISP environment offers run time binding and thus eliminates the requirement that the object's architecture be determined at compile time. Rebinding of methods to alternative application programs provides a form of resource sharing; the resources do not need to run on the same host as the document processor, as long as they can communicate with it. No version control is provided because there is no document duplication and, with only a single user per object, no access control exists because there is no security requirement on material.

Leland et al. produced Quilt [LFK88]. It is a computer-based tool to aid collaborative document production and is therefore of great relevance to DODA. It claims to support all types of documents and degrees of collaboration necessary for efficient document processing. Quilt explicitly addresses the issue of document access control. It assumes that shared documents must be protected so that only appropriate people have access to them. Quilt enables the specification of *proper function*, i.e. responsibilities and patterns of interaction, based on the social roles users play in a document's production. For example, there are the predefined roles co-author, commenter and reader. As well as user roles there are document types e.g. base document and suggested revisions, and a predefined set of operations that can be performed upon a

[38]Interleaf recognises intradocument structure ranging from characters and graphics to the high level objects that give them structure e.g. paragraphs, chapters, pictures.

document, such as create and modify. User defined roles, document types and operations are permitted.

To transact in Quilt a collaboration must be set up before a document type is initiated. A collaboration consists of an list of the users that constitute a user group, details of the styles of collaboration to be used and a statement of the social roles played by collaborators. Quilt-defined collaborations are

- *exclusive* - only the author of a section can modify it;
- *shared* - any co-author can modify any document section;
- *editor* - the designated editor can modify any section; other co-authors may only make submissions to the editor.
- *customised* - user-defined style specific to particular document types or collaborative group;

Text must be stored and accessed through the Quilt system in order to uphold the collaborative relationships prescribed. The Quilt database is centralised and therefore all collaborators must have access to the same machine. These notions of access control are not expressed through the medium of an access control list and control is of the coarse-grained type.

2.4 Distribution of Document Processing

2.4.1 Motivations for Distribution of Processing

The centralised document processing system designed and implemented by the author indicated the feasibility of enhancing the 'semantic' significance of Amoeba concurrency control flags. Each document flag signified the application of a method to a particular document field. The document server was, as with Amoeba, provided with a rule base of permissible field flag permutations in order to analyse presented transactions at validation to determine whether validation should succeed and lead

to the production of a new version document or whether it should result in a transaction abort. It became apparent that the flagged information, when examined in conjunction with the identity and access rights of the transaction's initiator, provided a sophisticated form of transaction representation suitable for use as an audit trail of a transaction. Audit information was actually contained in each field's ACL for there was a direct mapping between the ACL representation used and flagged information[39]. The rights represented in each field's ACL were restricted[40] to indicate only the access rights that were actually exercised over that field by the user.

The centralised service outlined above, and reported in [Sno90], provided concurrent document processing and auditing facilities. However, the system had a number of the weaknesses associated with systems based on a client-server design [BNY86]. In particular, the security of the system was based upon the trustworthiness of the single, multifunctional server and progress in processing was wholly dependent upon the availability and performance of this centralized server.

Distribution of the centralized document processing system outlined above was seen as a way of overcoming the recognised limitations. A number of additional benefits seemed likely to accrue from distribution such as greater version availability, increased concurrency and also, by using an appropriate document abstraction, the provision of a more general purpose system that eliminated the need to develop a new server for every new document type.

The form of audit trail arrived at in the precursor system suggested the opportunity for a unified approach to consistency enforcement within the distributed document processing system; in particular by the association of a concurrency control and access control policy, as both types of control are concerned with maintaining the consistency of documents.

[39] Each entry in the list represented the right of an identified user to make a method application upon a field.

[40] i.e. the full set of rights held by a user at the outset of the transaction was restricted in the archived version. For a user who proposes a transaction in which she exercises all her rights, no restriction would take place.

2.4.2 Distributed Document Processing

The basic tenets of the document processing system developed are summarized below.

- distributed;
- open, yet secure processing environment;
- processing of document parts and incorporation of updates into new document,
- to allow parallel processing of document parts;
- to provide an audit trail, through version archiving.

The system was named DODA, an acronym for Distributed Office Document Architecture. The design of DODA provides unrestricted read access to documents[41]. However, there is rigorous control of write access. Read access can be gained either through an archive server or from an untrusted local cache. In either case the integrity of the document is assured and can be verified by the user through the application of a "tamper-checking" method. The currency of the document is verified by reference to the single *Visibility_server* in the system[42]. It is assumed that copies of folios of a document will be widely available in "untrusted" local caches.

2.5 Summary

This wide ranging literary review has examined a number of issues that relate to the design of DODA and identified a number of recurrent themes such as object orientation, distribution, concurrency and optimistic processing, the use of application semantics within transaction processing and security. We have suggested that one effect of the traditional approach to distributed system design is to produce a paradoxical relationship between certain functions within systems, such as data availability and security.

[41] Although it may be beneficial to restrict the membership of the user group to which the facility is available. See also Chapter 4 for further information on restricting read access.

[42] i.e. this functionary is not replicated within the system.

In the succeeding chapters we will examine how the recurrent themes have been brought together for the design of DODA to be unified within a single concept that we refer to as "document integrity" in order to resolve the conflict between document security and document availability. The following chapter describes the motivations that gave rise to our work and begins to indicate the nature of relationships between the document architecture and other processing environments.

Chapter 3

Overview of a DODA system

3.1 Introduction

The objective of this chapter is to provide the reader with an outline of DODA by giving a general description of distributed document processing within DODA.

3.2 The Approach

DODA is intended for use in an open environment and does not aim to prevent data from being read. An 'open' processing environment, as stated by Tschammer et al. [TEH+89], is potentially hostile. Document security is not dependent upon the underlying processing environment for an open environment is one in which there may be heterogeneous hardware, operating systems and network software, with no common underlying security structure. Hence the document architecture itself facilitates security for documents. However, we make the legitimate assumption (according to Tschammer et al) of communications between cooperating components. Security must be provided within DODA; the level of security provided is prescribed by a document's user group. DODA assists the user group to prescribe security for the management of document updates. Should a user group demand read protection of document, then the user group may provide this independently for a DODA document but DODA will provide no guarantees or direct assistance with its enforcement.

DODA is based on the assumption that users at geographically distributed locations, collectively responsible for a document's preparation, should be able to work effectively in parallel and working should be sufficiently secure to guarantee the integrity of the developing document.

DODA documents are structured objects because they are sub-divided into smaller units called *folios* (akin to fields), which can be considered as sub-objects. Folios are manipulatable via methods. Chapter 4 contains a fuller account of the DODA conception of a document. Folios are distinguishable from the fields of a relational database system for instance for they have associated with them control information, such as an encrypted folio checksum to deter tampering and the identity of the user that last updated the folio. The methods of a document are literally part of that document object instance because each method is held as a folio of the document. Method folios are managed in a similar way to data folios and thus are protected from tampering and are attributed to a named user.

Control of processing is optimistic. A user reads a document and produces a transaction in the optimistic belief that the document changes it represents are valid and will, therefore, contribute to the document's development. DODA allows multiple users to propose such transactions concurrently, in the optimistic view that there will be no interference between their proposals. Validation of transaction proposals takes place in order to decide whether conflicts between concurrently prepared updates did or did not take place. A successful validation results in a commitment, a write, of the updates through the creation of a new document version. An unsuccessful validation does not lead to the update being committed but results in the user concerned receiving a list outlining conflicts identified during validation.

DODA advocates, but does not force, a *thoroughly optimistic* approach. By this we mean that, in addition to the optimism outlined above, the checks of validation examine the process and procedures used to construct the transaction proposal e.g. was the user entitled to make the method application (i.e. was it part of her role in document development)? were the applications made in a legitimate order? DODA suggests that, because such validation is worthwhile even when there is no concurrency within the system, there is benefit in defining validation such that all but 'integrity violat-

ing' transactions are commitable and incorporated into a document. For example, if it is reasonable for changes proposed to a preceding current version document to be committed to the now Current_version then validation should succeed.

As validation indicates, DODA takes a unified approach to the detection and prevention of access control violations, concurrency conflicts, semantic consistency failures, deliberate tampering and accidental corruption. The approach is dependent upon the specification, by a user group, of a document-specific notion of *integrity.* This 'integrity standard' is the measure against which document update actions are measured during the validation process.

3.3 Document Transactions

DODA does not implement update in place. Document transactions are optimistic and therefore execute in three phases, read, validate and commit (or abort). The result of successful transactions is the development of a sequence of versions (as illustrated in Figure 4.1), forming a chronological, *linear version history* of the document, which is archived. A linear history was considered most appropriate for document application[1].

Document transactions are initiated by a user, who is a member of the document's user group[2], requesting read access to a document. The user must present an ID with the request, but this is ID is not authenticated at this stage. However, ultimately any document changes resulting from the transaction are attributable to this user, hence authentication of the user's identity is required at a later stage. A document reference is issued in response to the request and the user has access to the means by which to confirm the integrity and, less importantly, the currency of the referenced document i.e. checking that the reference refers to the current version. Confirmation of currency and consistency facilitates progress, for it ensures that changes are proposed only to consistent documents that have been recently committed thereby decreasing the likelihood of conflict between a version and the initially proposed transaction.

[1] However, this form of version history is not rigidly prescribed within DODA; it is possible to implement branching version histories if a user group decide that it is a more appropriate in their application.

[2] A document type may be provided with methods that implement procedures for introducing new members to the user group on the fly.

Read access to the archived current version document allows the user to take a local copy of the version. Changes are proposed on the basis of this local copy. The form of transaction proposal is called a *doculett.* It has some similarities to an intentions list but, as stated above, contains additional information pertaining to the user(s) that applied operations to folios. It represents possible future state of the document and only becomes committed and therefore a visible document version after successful validation. The *Current_version* is the most recently committed version; other committed versions represent past states of the document.

A document's user group specifies the document's type specific validation criteria with regard to the semantics of the document. Conflicts are recognized in terms of method applications to document folios because a notion of serialisability that examines only read-write orderings is unsuitable in document development environments [HOS90]. DODA's validation criteria express the "social contract" (between users of the document type) relating to transactions upon that document type. Hence validation examines not only the proposed update itself but also the process by which it was developed. The representation of transaction proposals used within DODA, a structure called a doculett, reflects this.

The last phase may be a write phase which creates a new current, immutable document version when validation is successful and the transaction proposal is successful. Unsuccessful validation results in the return of the transaction proposal to the initiator with an explanation of the commit failure.

3.4 Processing functions

It will be recalled that in the centralized system that motivated DODA a type specific server mediated all transactions on behalf of users. Although a form of optimistic processing was used and that form represented a reduction in the time during which a server was blocked and unavailable to other transactions (compared with the scheme suggested by Oliver[Oli90]), nevertheless doculett presentation was a blocking call. All validation and version creation activities were conducted by the server on a serial basis. An important aim of distributing document processing in DODA has been

to enhance availability yet maintain security by alleviating the bottleneck due to blocking calls, host failure or security violations. To this end the range of processing functions performed by individual type-specific servers in the precursor system, for example transaction validation and authorization for archiving, are distributed so as to provide a more robust service that is able to resist, or else contain, the results of server unavailability. The model that has been formulated divides the functionality formerly the responsibility of a single type specific server between a set of functional components; the point being to provide functional components that can be duplicated for availability and protected from tampering for security. The components take two forms.

Firstly there are components that manage document integrity. These are document methods; for every type of document instantiated in the DODA environment, a pre-defined set of methods must appear as a subset of a document type's methods. The identified subset of methods are those that carry out transaction checks e.g. the validation and archiving methods, and are the means by which 'type specific' integrity, formerly offered by the type specific server, can be provided. These methods form part of every document type and as such may vary in implementation between different document types. Document methods and the way in which methods are protected are topics covered within Sections 4.5 and 5.2 respectively. This method set (and other methods also) appear as part of each document version. Hence availability of methods is provided by local duplication of document versions.

Secondly there are agents, called *functionaries*, which together provide the server authorization role, particulars of which will be discussed in Chapter 5. The important point is that their intervention in document processing is detectable and signifies recognition of some processing action having taken place. Functionaries make use of a uniform interface to *all types* of document object, this interface being provided by the subset of document methods mentioned above. Section 6.3 is the point at which the role of functionaries is fully revealed. To advance availability of services DODA provides a number of these servers locally to users. To maintain security, these functionaries are small self-checking pieces of software, as suggested by [Coh85].

These functional components achieve *document integrity.* Although any given doc-

ument type may be viewed by its user group as having individual integrity requirements, all DODA documents handle the issue of integrity in a similar way, namely the provision of control information, functionaries and certain 'supervisory' methods.

3.5 Documents

The DODA notion of a document is discussed in depth in the next chapter. Here it is sufficient to say that DODA's document abstraction allows documents to provide a high degree of self-protection from both tampering and from 'unwise' processing. Thus, in part, a document type instance takes on some of the functionality of a type-specific server. It does this in a self-referential way by providing a protected environment in which to maintain the document's type methods and facilities such as public encryption keys (used for part of the local "informal" document and doculett integrity checks) that help to manage the document's protection.

3.6 Summary

The provision of concurrent document processing through the intermediary of a centralized server suggested that an improved service could be provided by distributing the server functions. The distributed server design developed is that of documents and multiple functionaries. As we shall see in the following chapters, such a model can provide a generic document service that can, nevertheless, be tailored for the instantiation of a wide range of document types. In the next chapter we discuss the document abstraction that forms the basis of all DODA document types.

Chapter 4

Document Architecture

4.1 Introduction

DODA is an object based system; document types can be specified and instantiated. Specification of a document type involves defining a document structure and representation, the methods by which instances of the document type may be processed and the semantic constraint rules. This chapter contains a detailed explanation of a document object. It begins by describing a document representation which provides the basis for representing many different types of document object. The following section relates the DODA notion of document type to the concept of document semantics. Finally the implications of this document abstraction are explored.

4.2 Document representation

In DODA each instantiated document object of a particular type[1] comprises a historic sequence of immutable document instances, called *versions*. These are archived in persistent storage in the order in which each version was committed, as shown in Figure 4.1. The archive, situated at a single node, maintains the self-protected current version document (i.e. the definitive state of development of the document instance). The caching of copies of whole or partial versions, or even the caching of an archive

[1] Document type will be discussed in Section 4.3.

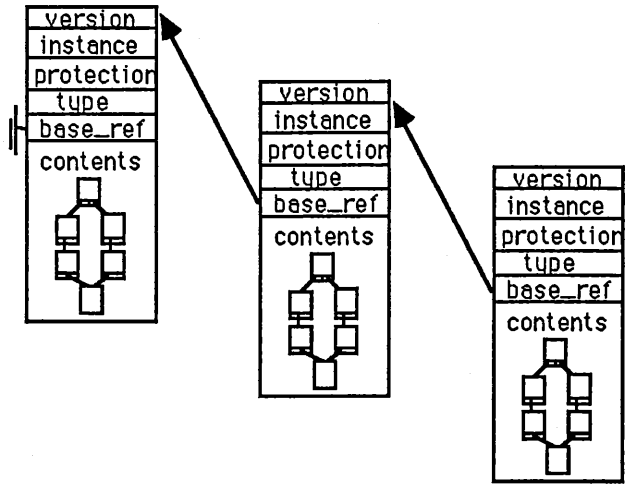

Figure 4.1: A Representation of Versions of a Document Instance

of document versions, locally to a user is encouraged within DODA. Such duplication facilitates version availability, yet cannot undermine version integrity, as we shall see in Chapter 5. The most recently committed document version is known as the *Current_version*; it is linked into the relevant version history within an archive by a *base_reference* within the version's *index_page*. The version's base-reference holds a reference to the immediately preceding current version document instance i.e. the version to which a validated doculett was applied in order to create this current version. Note that the base-reference does not necessarily indicate the version upon which the doculett was originally based.

Each document version comprises not only user's data, stored in fields known as folios[2], but also document control information, including the base_reference, which is held within the index page. Instances of all document types possess, for control purposes, self-description information, for example, each has a unique document version

[2]The term 'folio' is used rather than fields to emphasis the fact that folios hold data and data management information that includes, for example, the ID of the user attributed with the last update to the folio.

identifier, a document type descriptor and folio description information, such as each folio's unique identifier and contents reference. Additionally each document version maintains access control information, the placement of which will be discussed in Section 4.5. Whenever a new document version is created by a transaction, a new index page is created because the control information for any version will be unique. However, a newly created index page may refer to folios which are shared with previous document versions; references to folios unchanged by the committing transaction are simply copied into the new index_page. Only modified folios, the updated contents of which are being archived anew, necessitate the creation of folio content references. As shown in Figure 4.2, each version's index_page comprises a,

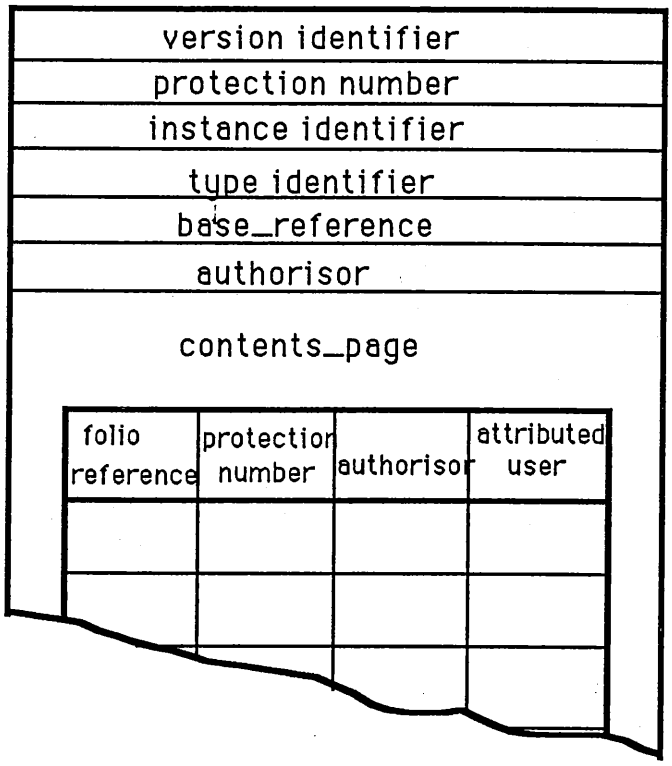

Figure 4.2: A Document Index_page

- document version identifier;
- protection number;
- document instance identifier;
- document type identifier;

- base-reference;
- authorisor's identity;
- contents-page;

  A list of folios each entry of which is a

  - reference to folio contents, i.e. user's data;
  - folio protection number;
  - folio authorisor's identity;
  - folio attributed user's identity;

The existence of a document's protection number is crucial to document integrity (as will be shown in Chapter 5), for changes to an archived version detectably invalidate its protection number i.e. a protection number provides an *anti-tamper device.* The authorisor's identity also plays a role in document or folio integrity protection and facilitates the auditability of processing. The base-reference, as mentioned earlier, is a reference to a previous document version. The reference will refer to the document version identifier of the immediately preceding version and hence forms the link between successive historical instances. The contents_page is, in principle, a list of protection numbers for and references to the document's component folios. The phrase 'in principle' is used because the contents_page is potentially of complex configuration, being the structure through which a user group may choose to express any relationships that exist between the folios of a document. Further details of this appear in Section 4.4.

In processing, users are offered, by default, read access to the Current_version instance of a document. However, users are not confined to accessing only the Current_version instance of a document.

Folios are accessible on an individual basis because it may be helpful to users to be able to trace the development of a single folio instance. Hence, each folio version has a unique identifier in order that users should be able to request accesses. Folios are akin to nested document objects and possess the folio equivalent of a document version's base_reference. This folio base reference is known as the *fbase_reference.* The reference

allows the historical development of a folio to be traced simply, without the need to navigate whole predecessor documents. It facilitates economic storage of folios within the archive because it enables folio versions to be simply shared between document versions.

Although a user group has free read access to document versions, modification access is rigidly controlled by checking the access rights permitted to an accessing user against those granted within an access control list. The form of access control list advocated for a DODA document is elaborated in Chapter 5, suffice to say here that each list entry can indicate not only the possession or denial of a right to a user, but also whether that user is granted the capacity to delegate a particular right to some other user. Access rights are applicable to folios and conceptually an access control list exists for each folio of a document. However, this apparently simple statement masks an interesting access control feature that arises from the sub-division of documents into folios, namely the potential to provide an access control list for a whole transaction that involves multiple folios. To illustrate this, consider the example of an examination paper document in which question sections are developed by a lecturer, with the aid of an internal and an external moderator. Suppose each moderator performs the same functions namely to provide attributed comments on the exam questions and model answers; in DODA this equates to the moderators invoking the same document methods upon folios. Suppose also that the exam 'paper' comprises six question, each of which appears in a folio and each of which must be moderated. Since access rights reflect or constitute roles played in document development, a moderator's role in relation to each question folio is identical. Thus a single ACL could provide the information. In this case there would be a one to many relationship between the ACL folio and the question folios.

4.3 Document type

The document control information and representation described above is general to all document types. Providing customized document types involves a user group clarifying the type's required functionality. Functionality is specified in terms of document structure, operations on folios and invariants, also in terms of permissible

ordering of execution of methods and personnel involved. The structure of a document type is expressed in the configuration of its component folios. However, to declare the set of component folios is insufficient to define a document type, as we will show in Section 4.4.

A DODA document is a structured, form-like object that is differentiated into fields, known as *folios*. A folio may comprise text, numeric data, graphical material (conceivably also voice or video, although issues related to multi-media folio types are beyond the scope of this work); a single folio may even comprise all of these because it is possible for a folio to be a composite object consisting of sub-folios[3] i.e. a folio may closely resemble a document. It may be that a particular document is most appropriately structured as a suite of documents with each 'top level' folio literally being a document, in turn composed of folios. For instance, one can envisage a document type created to manage a company's expenses forms. The company expenses document might appear as a tree of folios, its structure mapping directly the hierarchical structure of the company. Thus, the company expenses document would be composed of a number of folios, each being a company division's expenses document, a division's expenses folio in turn being sub-divided into folios for each department within that division and groups within each department having their own expenses folios[4]. To complete the analogy of a folio to a document, each folio is provided with a reference to its predecessor folio version (the contents of which might not be stored as part of the predecessor document version). Thus the interpretation of 'document' has wide scope. A document might, for example, be interpreted as a software system composed of a suite of programs[5]. In contrast, a document might be interpreted as a single program, with each procedure forming a folio, or both.

The rules governing how a document is completed in order to maintain a version's integrity will vary with the document type. For instance, a simple Bank-statement document may demand that credits and debits must appear in the chronological order in which the bank was notified of the entries. Perhaps another document type may demand that two particular folios must be filled in by the same user. To illustrate,

[3]There is no theoretical reason why a single folio cannot contain all three forms of information. However, the representation of a document instance as folio divided into sub-folios naturally leads to the interpretation of the different information types of a folio appearing in separate sub-folios.

[4]The result is a structure closely akin to Amoeba's superfiles, files and pages hierarchy.

[5]There is a precedent for regarding a program as analogous to document[QNA90].

imagine a form documenting the contracted hours of a part-time lecturer and payments claimed and made for teaching at an educational institution. Such a document would be complete once the contracted hours worked had been remunerated. This document would possess folios giving the lecturer's personal details and also folios detailing the lecturer's teaching commitments and pay rates. In addition, a series of folios for claiming and authorizing payments would be necessary. Several parties may be involved with the document's development. Let us imagine that the demand for part-time staff is determined on financial grounds and that, as a consequence, folios dealing with hours worked, wage rate and salary payment authorization must all be filled in by the same user, the financial department. Another rule applying to the processing of this document type might be that total hours claimed does not exceed those stated in the Contracted-hours folio.

An instance of a document type is created and developed through a series of transactions that are implemented using the document type's methods (as described in Section 4.5). Proposed changes to a document version are written into a form of protected and attributed intentions list, a *doculett.* A doculett is only irrevocably applied to a document instance if the integrity of the resulting version can be guaranteed. Doculetts are normally, by default, applied to the most current document version[6]. The details of what constitutes integrity for a document type is for a user group to define in the form of integrity checking methods. Hence, changes effected by a transaction are written to a visible document version only after they have been validated by the application of the checker methods. Successful validation results in transaction commitment, which is the incorporation of changes into a document version which then becomes visible.

4.4 Document Semantics

As with many kinds of forms in every day use, the contents of one folio of a DODA document may be closely associated with the content of another; for example an application form that requests both the date of birth and age of the applicant. The set

[6] A doculett's application to an older versions is not precluded but will result in branching version histories for the document instance. This form of archiving is not considered within this thesis.

of relationships that exists between document folios are what we term the document's *semantics*; these in turn partially define document type. It is possible, therefore, for a set of folios to be referenced by several different document types[7].

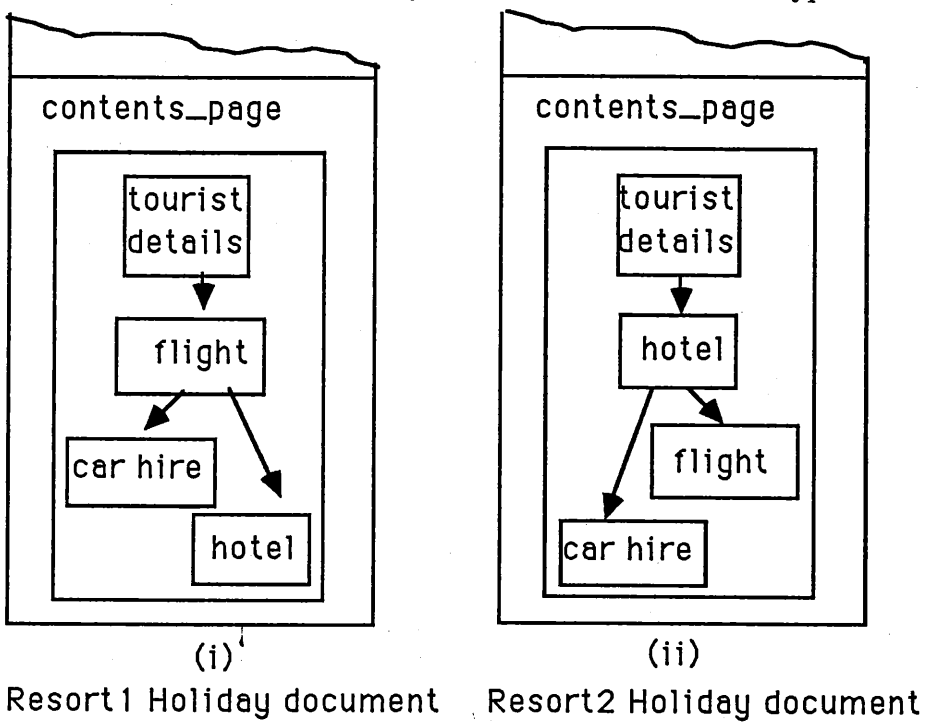

Figure 4.3: Types of Holiday Booking Documents

The scenario of travel booking used by Gray in connection with nested transactions [Gra81], serves to illustrate a folio set representing multiple document types. In the scenario a potential holiday maker contacts a travel agent requesting certain dates at a particular resort. The travel agent performs a transaction that makes reservations for air travel, car hire and hotel accommodation. According to Gray, the travel agent sets in motion three sub-transactions to book air travel, car hire and hotel accommodation respectively and receives results which are examined for conflicts such as the hotel booking expiring the day before the traveller's return flight. In DODA, travel arrangements could be developed and recorded in some type of 'holiday' document, each instance of which would be shared between a user group of travel agent (on behalf of the holiday maker), airline, car hire company and hotelier. Imagine that

[7]Arguably such variations may be more appropriately considered as sub-classes of the same type of document.

for one resort, Resort1, the airport handles only three flights per week, but hire cars and hotel rooms are plentiful; in another resort, Resort2, there are many flights per day, ample car hire and restricted accommodation availability. The user group may choose to represent this simple semantic information by configuring the folios within the holiday documents of Resort1 and Resort2 differently, as illustrated in Figure 4.3. Documents of type 'Resort1 holiday' may represent hotel and hire cars bookings as subordinate to airline booking, because air travel is the scarce resource at that resort. In contrast, a 'Resort2 holiday' document might treat accommodation as the primary concern.

As suggested in Section 4.3, a suite of computer programs could be represented as a DODA document type, with each program of the suite contained within a folio. Within a document application of this type the folios' associations are of a familiar and well understood kind, for such dependency relationships are commonly expressed within the UNIX tool MAKE. In DODA terminology, the software's relational dependencies would be referred to as the compile/link semantics.

The common feature of a document interpreted as a software system and a document as a single program (given in Section 4.3) is that each document folio forms a *semantic unit.* Thus, by suggesting that a folio may itself be a document, we are suggesting that a document type can be designed to represent the '*granularity*' of semantic relationship most appropriate for the requirements of the document application. The abstraction of the folio as a semantic unit allows for the specification of a wide range of document semantics. The primary task for a user group in specifying document types, within the DODA framework, is ascertaining which folio granularity is most appropriate to the document application. Frequently the relationships will be hierarchical. However, not all relationships between document parts are of this kind e.g. the relationship between a proof reader's comments and the original text.

At least some element of a document's semantics may be representable in the document's physical structure[8]. For example, in a 'genealogy' document type (representing a family that avoids intra-marriage) the folios may be organized hierarchically as a

[8]Other elements of a document type's semantics may be represented and usable via the type's method.

tree[9]. Other document types will exhibit different forms of folio relationships, for instance, a 'motor insurance' document type, as shown in Figure 4.4, may be configured

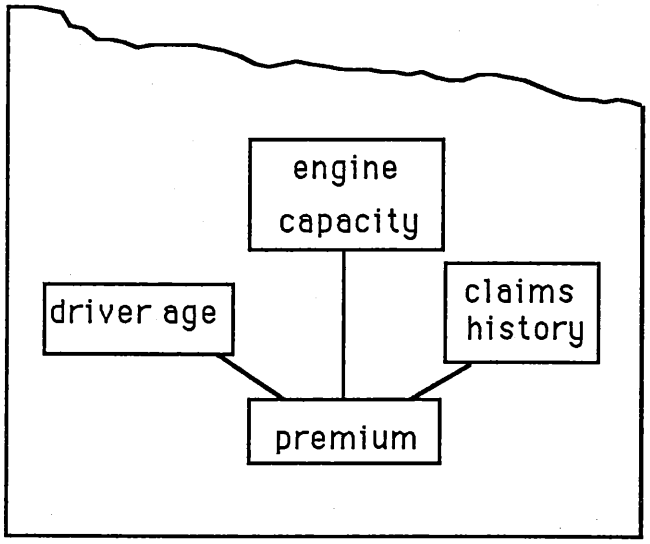

Figure 4.4: A Representation of a Motor Insurance Document

as a graph of folios, with the premium payable folio showing a dependence upon the contents of driver's age, engine capacity and previous claims folios. The structure of a simple program document can be visualized as Figure 4.5. This particular program document comprises six folios; a header, source code folios, object code folios and an executable program folio. Methods for this document type would include such operations as Edit-folio and Compile-folio. In this example, disregarding the role of methods, the document semantics are defined by dependency relationships between certain of the folios, expressed as the directed, acyclic graph pictured. Folios B and C are dependent on folio A. This is because the header is 'included' in each source code folio and changes to folio A, for instance the redefinition of a constant, may affect the content of folios B and C. Folio D is dependent on folio B as the object code of folio D must result from the compilation of source code folio B. Similarly folio E

[9]The utility of such an approach is indicated by the Amoeba file service[Mul85], which is able to deduce transaction conflict information from the flagged accesses made during the page tree's traversal.

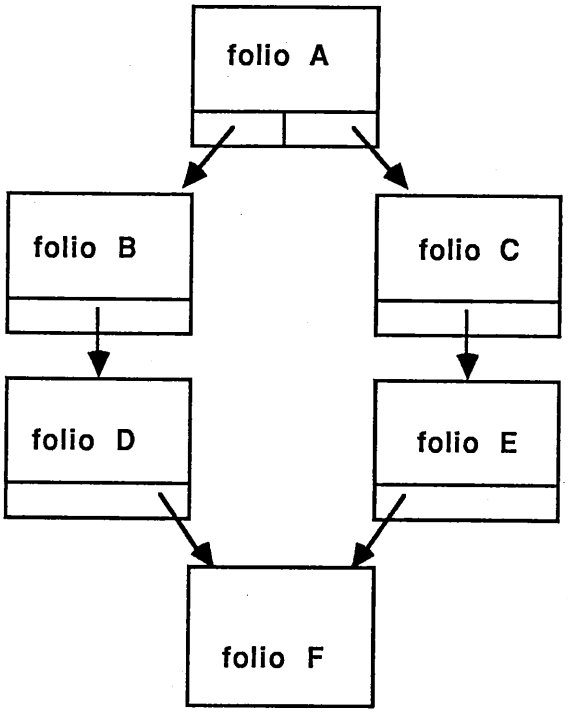

Figure 4.5: A Representation of a Program Document

depends on C. Folio F is dependent on folios D and E as it arises from the linking of these object code folios. To maintain program document semantics, i.e. to guarantee the consistency of this program document, the executable folio, F, must have resulted from source code folios A, B and C. Conversely, processing that resulted in updates to be written to the source code of folio B, for example, would also requires appropriate updating of folios D and F in order to maintain the document's consistency.

Representation of a document type's semantics expresses direct relationships between data folios and information concerning each i) method application to a folio or a number of folios[10], ii) the inter-relationships between methods and iii) the rules governing the order of application of methods[11], for these may also have a bearing upon the folio granularity that the user group determines to be most suitable.

The set of valid possible future states, from which the version history is formed[12], is also determined by the document's type. DODA does not prescribe which method or

[10] A single method may take more than one folio as an argument.
[11] In order that a conflict specification can be produced.
[12] As a document passes from one state to any one of a finite set of possible future states.

notation presents the most suitable means by which to declare document semantics[13]. Although DODA does not force a user group to subscribe to any particular form of semantics expression, the assumption is made that folio inter-relationships, expressed in some way, can be reduced to the equivalent of a case table of conditional method application sequences e.g. Figure 4.6 which examines the possible application of the methods READ, WRITE, APPEND and EXECUTE to various of the folios of the program document[14] mentioned early (Figure 4.5); F_A, for example, denotes folio A. As with the declaration of semantic information, DODA provides a number of alternative 'placements' for semantic information within a document type. The user group is free to express the type's semantics at what it deems to be the appropriate abstraction level for such representation. To illustrate, the contents_page may be a simple list and representation may be made by directly linking related folios, via folio held references, as illustrated by Figure 4.5; a user group may determine that the contents_page is the most appropriate level or the structural representation of a document's type may be concealed within a graph of the access control lists for the document. It must be noted however, that a document type's structure may be logically independent of the document protection measures, such as access control, imposed by DODA[15] (and elaborated in Chapter 5). It follows that when protection is independent of document structure, a document type should be *foliated*, i.e. sub-divided into folios, so as to provide the most convenient structural representation for concurrent processing. Generally, this would mean choosing a folio granularity such that updates involve only new text, or other new folio contents, being archived for each committable transaction. This consideration is intimately related to the granularity of method applications.

[13] Path expressions have been suggested as one possible means, for example. It seems likely, however, that the suitability of any one method will depend upon the document type to which the method is being applied.

[14] Such actions would be expressed in a doculett.

[15] The logical independence of document structure and document protection is not a necessary condition of documentness, for access control may be a semantic consideration in some document types.

```
CASE doculett-action OF
      READ      : consistent := TRUE ||
      WRITE     : CASE folio OF
                        F_A : IF doculett includes update of ALL subordinate folios
                                  ((W(F_B),R(F_B),C(F_B),W(F_D) |
                                  W(F_C),R(F_C),C(F_C),W(F_E)),
                                  (R(F_D),L(F_D) | R(F_E),L(F_E)), W(F_F))
                              THEN consistent := TRUE
                              ELSE consistent := FALSE;
                        F_B,F_C : IF doculett includes update of dependent folios
                                  THEN consistent := TRUE
                                  ELSE consistent := FALSE;
                        F_D,F_E : IF object code from certified compiler AND
                                     doculett includes update of subordinate folios
                                  THEN consistent := TRUE
                                  ELSE consistent := FALSE;
                        F_F : IF executable code from certified linker
                              THEN consistent := TRUE;
                        ELSE consistent := FALSE ||
      APPEND    : CASE folio OF
                        F_A,F_B,F_C : IF user has 'signed' that no dependent folio
                                         is affected AND append is only action
                                      THEN consistent := TRUE
                                      ELSE treat as ≡ WRITE folio;
                        ELSE consistent := FALSE ||
                                  .
                                  .
      EXECUTE : IF folio executed is F_F THEN consistent := TRUE
                ELSE consistent := FALSE ||
END || (* of CASE doculett-action *)
```

Figure 4.6: Aspects of program document semantics

4.5 Document methods

Document versions are developed through transactions, as will be elaborated in Chapter 6. Each instance of a particular document type is manipulated by its associated type methods, applied by users directly or by functionaries (on behalf of users). Thus, completing a document by filling folios with the required text (numbers or graphics) entails applying these methods in some order to a subset of the folios of each document version. A method that causes an update can be viewed as a unit of change[16]. A sequence of method applications to a sub-set of a version's folios may constitute a transaction e.g. a transaction to check the preceding year's accounts and provide details of single debits of over £100 is of this kind when applied to a bank account document that is structured as monthly statement folios; alternatively a single method applied to a single folio may implement a whole transaction e.g. changing the surname in a personal details folio may be performed as a transaction. Crucially, a transaction represents a co-ordinated set of folio changes. The most appropriate abstraction for a transaction (single method or method sequence) is related to the nature of the document type, as with foliation of document contents e.g. an alternative structuring for the bank account document might be the arrangement of a year's financial transactions into a single folio.

The analogy with document contents folios is deliberate, for within DODA each method is held within a *method folio*, in exactly the same way that data is held in a data folio. Thus DODA makes a literal interpretation of the object oriented paradigm by making an object type's methods integral with object type instances; each document version is the possessor of its own methods[17]. DODA places methods in folios because i) a method is an abstraction of the document type's semantics and the folio is the recognized unit of that document's semantics; this approach offers a consistent approach to document structuring and ii) by placing methods in folios they are protected from tampering, guaranteed to have integrity and, if subject to amendment, can be changed in a carefully controlled and visible manner.

[16]Not all methods cause version updates. Non-update methods, such as an audit method, can nevertheless be seen as units of activity, if not change.

[17]Because DODA allows unchanged folios to be shared between versions of an instance, the 'general' method folios are likely to be shared across all versions of a document instance.

In principle method folios are indistinguishable from data folios because of the form of protection and attribution. It will be useful to provide, in the contents_page, a folio description which distinguishes data folios from method folios. Like data folios, method folios are shared between document versions, may also be inherited from other document types[18] and most importantly may be subject to change; the integrity management scheme that allows checking of transactions against user defined criteria offers the means to safeguard the user community from 'ill advised' method changes e.g. the user group could impose the rule that a method update proposal needs to be vetted by a number of users. In practice, method folios are likely to be more static than other folios within a document type; although in principle method folio change is not precluded, a user group may actually not endorse method change in a particular document type. Updating of methods is rigorously controlled by the same integrity safeguards that apply to all folios (further details appear in Chapter 5). Such an environment ensures that only approved methods are used upon a document version and yet offers flexibility because document methods need not be rigidly determined at the document type's instantiation.

Within all DODA documents there appears a limited set of, what might be called, general methods concerned with instantiation of document types and safeguarding the integrity of document versions during processing e.g. a transaction validation method. The Create_version method, for instance, is responsible for creating a version index_page and entering the appropriate folio references for new and shared folios. The folios that implement this method set are not identical in every document type[19] since implementations will vary between different document types. But all DODA documents are required to possess a representative of each method from this set[20].

With the provision of 'safeguarding' methods for transactions upon folios, new methods may be introduced to a document type and different versions of a particular document instance may provide different implementations of a method. The latter provision ensures that DODA documents can adapt to heterogeneity, not only to the variety of hardware and/or software environment of an open system, but also the

[18]The burden of document specification and implementation is on the user therefore it is particularly desirable to offer re-use of method code.

[19]Identical method implementations appear only in document types between which method inheritance has taken place.

[20]As is the case with the installing functions in C^{++}.

heterogeneous requirements presented during a document's life-cycle, such as variations of access control, concurrency control and perhaps even changing version control requirements and altering folio relationships.

In Section 4.2 we suggested that document foliation has interesting implications for the provision of document access control. The interest is manifold and results from access control lists being considered as folios. Firstly, it is crucial that the integrity of each ACL is maintained; the folio protection scheme in place in DODA can be used to maintain the integrity of the access control structures. Secondly, ACLs can be economically stored as folios because folios can be shared between versions. Thirdly, because foliation provides semantic structuring there is the potential to create ACL folios applicable to a wide variety of data or method granularities e.g. an ACL for a whole transaction involving multiple data and method folios. Fourthly, DODA takes the view that within state-based access control the access permissions of users may need to change according to state changes within the document and its folios. Therefore, we argue, it may be necessary for folios' ACLs to change in line with document changes. Consequently placing ACLs within structures (folios) designed to facilitate the process of controlled change is beneficial. Finally, assuming there will be ACL change during a document's lifetime, a uniform approach to ACL change would be to provide methods by which to manipulate ACLs (i.e. ACL folio updating methods). The question of whether the ACL manipulation methods can themselves be changed then arise, and so on. DODA compromises and provides access control in the form of access control methods that are applicable directly to data (and perhaps method) folios[21].

A change in the relationships between folios, as implied within Section 4.4, may be described as a change to the document's semantics and therefore constitute a change to the document's type. Such adaptability could be considered as a controlled form of *dynamic sub-typing* of document objects; it combines flexibility of implementation with guarantees of document integrity.

The 'proof' of integrity of a folio or document is a valid protection number. The

[21] An ACL method is an alternative representation for an ACL. However, it also has the advantage that a single ACL method applicable to a particular data folio can provide access control information appropriate for the variety of states the accessed folio may take. In other words a single ACL method may encompass a number of ACL's for a particular folio.

principle behind protection numbers is that any change to the content or control information of a folio will result in a corresponding change to its protection number. In a hierarchically organised folio structure, for instance, a change to a 'leaf' folio would effect a change in the superior folios' protection numbers and ultimately the document's protection number. The use of protection numbers has implications for information sharing.

Sharing folios between document instances, as document instance versions share folios, is possible in principle and may be desirable in practice e.g. the sharing of employee personal details information within company documents. The crucial issue when processing shared folios will be maintaining document integrity across instances. Implicit in the notion of a transaction is the concept of change. Change to a document implies change to the document protection number. Therefore change to a shared folio implies change to the protection number of any document instance sharing that folio. Thus a transaction that changes a shared folio is, effectively, a *multiple document transaction.*

The use of multiple document transaction requires considering i) the provision of links between instances[22], ii) sub-typing and iii) implications for the placement of archives at nodes e.g. perhaps it may prove necessary to impose the restriction that sharing instances must be managed within the same archive[23].

In order to provide a uniform view of document types the folio structure of a document may be deemed transparent to the users. Under these circumstances users may be required to access them through the interface of *navigation methods*[24], in a similar way to that suggested by [PEea90]. Since navigation methods offer a means of presenting a variety of *views* of a document's structure to users they may perhaps (in a future version of DODA) be utilised for the provision of more restricted forms of read access than are currently offered[25].

[22]May also requires consideration of links between document types if different types are permitted to share (inherit) folios.

[23]This is not an unreasonable restriction for it is implicitly imposed upon instance's versions currently. The application of such a restriction would result in a uniform treatment of folio sharing within DODA.

[24]A document's folio structure is potentially very complex. Navigation methods assist in managing that complexity.

[25]We propose an investigation of read access control as a piece of future work.

4.6 Summary

What has been described is a document processing system in which any document's type specific 'details', encapsulated within folios, folio structures and methods, need to be known only to the document object itself. The index_page structure of document objects is generic and is recognised by the functionaries of the system. Functionaries are able to read and understand all index_pages and locate the data folios and method folios needed for processing.

The uniform abstraction of documents' data and methods as folios, the specification of access control as method folios and uniform provision of protection_numbers for folios and documents, together facilitate the provision of highly adaptable document processing environments. Dynamic typing, or sub-typing, of document objects is made possible by using such an abstraction. However, the flexibility is, paradoxically, carefully controlled in order to maintain the integrity of document instances, as we shall see in the next chapter.

Chapter 5

Document Integrity

5.1 Introduction

Having presented an outline of the architecture of documents in the preceding chapter, we now explain how a document instance actively maintains its own integrity. DODA uses the concept of integrity to unify the notions of application semantics, concurrency control within an application and various aspects of document security. Within this chapter we demonstrate the utility of adopting such a unified approach.

5.2 Document protection

5.2.1 Notarisation

The purpose of notarisation of data is to provide a tangible assurance that the data has not been corrupted either accidentally or deliberately. Notarisation is offered by calculating an encrypted checksum value over the data and appending the resulting value to the data. DODA does not control reading of documents. The data remains in clear text, and notarisation does not provide data secrecy. However, by simply augmenting the data with appended information about the data's originator and the

creator of the encrypted checksum before *shrinkwrapping*[1] the data, the notarised 'augmented data' can also provide assured information suitable for inclusion in an audit trail.

The process of protecting data through notarisation is two-fold. There is the performance of the shrinkwrapping and subsequently the notarisation checking to ascertain whether or not tampering has occurred since notarisation took place.

To notarise data that data is put through a checksumming algorithm to produce a value that is dependent on the content of that data. Thus any change to the content of the data is reflected in the value produced by the algorithm. To hamper system participants from changing data and simply recalculating a checksum value for that changed data, shrinkwrapping must involve some 'secret' component. Conventional wisdom is that it is preferable to provide secret keys rather than secret algorithms. Thus in DODA the algorithm by which checksums are computed is public; the anti-tamper device relies on the use of private keys for the calculation of encrypted checksums. A user is not permitted to notarise the data she produces, for we do not trust users to safeguard secret keys. A *notarisation server or notary*, which is trusted to maintain a secret key, is placed locally to each user to provide the data notarisation service[2].

The algorithm implementation is in the form of a *notarise method* held in a document folio, as with other methods. However, the notarisation method is different from other method folios in one respect only, namely its folio protection_number (Figure 4.2) *must* be provided by the *single trusted functionary* within DODA (see Section 6.3.2). For reasons of efficiency and consistency of approach, a local copy of the notarisation method will be held by each notarisation server. The notarisation method is the method least suitable to change and therefore the most amenable to inheritance by many different document types and to sharing by different instances of a document type. It is difficult to envisage this method folio not being shared by all versions of a document type instance.

[1]The term shrinkwrapping derives from the form of packaging of many products, such as commercially produced sandwiches, and is used here to indicate that the data etc. may be seen but not touched, except detectably.

[2]The issue of locality of notarisation services in relation to users is an interesting one. We do not trust a user to hold a notarisation key yet a user must be able to pass her data to the notary and be certain that it has not been corrupted prior to its notarisation. It highlights challenges in the implementation of a DODA system. It will be discussed further.

The RSA algorithm, for instance, which produces encrypted checksums over data as a single operation (c.f. producing a clear text checksum which is then encrypted), would provide a suitable basis for notarisation in DODA. The algorithm makes use of asymmetric key cryptography; a private key is used with RSA to provide the initial encrypted checksum value for data and its (claimed) public key counterpart provides checking by running with the decryption algorithm. When data has maintained its integrity between notarisation and notarisation checking the encrypted checksum values obtained using private and public keys will be identical. A mismatch between the values obtained indicates an integrity breach of the data, caused by tampering either with the data or with key values.

Although DODA does not restrict the implementation of notarisation to the RSA form outlined above, the utility of an approach based upon a public-key system is clear; the use of a public key encryption technique within notarisation facilitates notarisation checking.

Each notary uses a unique secret key. The ability to verify a notarisation must be provided for functionaries and users. Consistent with other provisions in DODA, the public-key of each notarisation server in use with a particular document type instance is made available through each document version. The public key of each notary in use is held within a folio of a document version[3]. As with the notarisation method folio, each *public-key folio* must have its folio protection_number provided by the *trusted functionary*. A notary may enclose its public key with data it has notarised as an identifier that could be used to verify the self-integrity of the notarisation[4]. However, by reference to the appropriate protected public-key folio it could be proved whether an acceptable notarisation server had undertaken the notarisation concerned.

5.2.2 Version Archiving

Document versions that are archived are not fully encrypted and are potentially readable by anyone. However, the integrity of documents is guaranteed at commit time

[3]One key, one folio or a folio for all public keys - this is the user's choice. Note too that DODA provides a form of free-standing public key certification.

[4]To ensure the data integrity the user would have to check the data within the notarisation against the original.

and archived documents (and their integrity) are protected by notarisation. The contents and all the administrative information, including access control and auditing information, pertaining to a document version is included in the notarisation process. The result of document version notarisation is the version's *protection_number* and, as mentioned in Section 4.2, each document version of every type of document possesses such a protection_number.

The purpose of a version protection_number is to serve as a guarantee of the version's integrity. The protection_number, like other notarisation, is verifiable by the means described in Section 5.2.1. As assistance for integrity checking, the identity of the notarisation service producing a protection_number is itself included within the notarisation. For checking, a notarisation algorithm is run using the public key folio of the notarisation service claiming responsibility for the protection_number's production[5].

A version's folios are included within the version's protection_number and folio changes are detectable at the document level. Each folio version has an associated *folio protection_number* derived from notarising its contents and control information[6]. These protection numbers are included within the base-page of the document and hence influence its protection number. Thus given the integrity of a version's folios, a document's protection_number can be derived from all the folios' protection_numbers and the version's control information.

The integrity protection scheme requires checking the validity of all changed component notarisations[7]. This is potentially an onerous task in large, highly structured documents and raises the issue of whether there is any way in which to reduce this burden. The form of document structuring, in other words the foliation of document types, will have a bearing on the ease of notarisation and will be an important consideration in choosing the semantic granularity of the document type. Since folios themselves may have sub-component folios, a folio's protection_number may be a

[5]In implementation the 'identity' of the notarisation service may be provided simply by reference to the public key folio needed for notarisation checking.

[6]In order to reduce update effects, it may be helpful to separate, by providing separate protection_numbers for each component, a folio's control information and content.

[7]The identity of the notarisation service producing a folio protection_number is an item of the folios control information. Checking of folio protection_numbers is carried out in a similar manner to other notarisation checks.

notarisation of its sub-components. The extent to which folio versions are sharable between document versions and the patterns of such sharing influences, and yet is influenced by, the requirements of component notarisation.

Both folio protection_numbers and the document version's protection_numbers are tied into the version when that version is archived. This is also the point at which decisions about version sharing are made. Part of the archiving process, that of creating a new version, is concerned with permeating throughout the new version the consequences, for folio protection_numbers, of folio content changes. Hence, amongst the version creation method tasks is that of tracing a versions folio graph and checking which folio protection_numbers are affected by the updates. The method then provides, in the same vein as the Amoeba copy-on-write mechanism, duplication of a folio's control information (and recalculation of protection numbers at appropriate levels) for folios affected by the change. Only folios which are unaffected by a change can be shared across versions. Herein lies the explanation for the DODA abstraction that separates the control information of a folio, kept in the document version's contents_page, from the contents itself.

5.2.3 Secure Communications

Secure non-local communications need be provided between users and local notaries. The security requirement is that this functionary must be protected to a degree equivalent to running that functionary on a separate host, in a locked room, contacted by users and other functionaries only via a secure mailing system[8]. A remote procedure call mechanism may be used as an alternative, providing that it provides an equivalent service. Assuming such a communication facility is available in order to instantiate and maintain the notarisation functionaries, the successful checking of notarised information is sufficient to guarantee that information's integrity. Notarisation of data securely associates a user-provided user-identification with the data. However, establishing whether the named user was actually the source of that data is a task that involves an authentication procedure.

[8] A user A cannot masquerade as a user B.

5.2.4 Authentication

DODA assumes the use of a trustworthy authentication protocol by which users and functionaries can be authenticated to the Visibility_server. There are many documented authentication protocols e.g. [NS78, NS87, OR87]. Nevertheless, in making this assumption we acknowledge that current authentication mechanisms have associated problems[9] [NHSS87] and that the choice of mechanism is consequently not straightforward. To address some of the issues raised by Notkin, the design of DODA utilizes notarisation to discourage integrity breeches and audit trails to provide traceability.

5.2.5 Key management

The 'classic' problem in relation to encryption is how to provide secret keys securely. By making use of a public-key encryption scheme the difficulties are somewhat reduced [RSA78] in DODA. However, in such schemes, public keys must be certified by an 'authority'.

In Section 5.2.1 we alluded to the fact that each notarisation server's public-key is held within a public-key folio. Within DODA, performing a document transaction that places a public-key into such a public-key folio is equivalent to the certification of that public-key; hence, the process requires the assistance of the single trusted functionary appearing in a DODA system. Public-key placement is one of the components of start-up of a DODA system and it is performed as a transaction and in a way consistent with all other transactions performed within DODA systems. It is one of the instantiation transactions and results in a notary that is operative because not only can the notary then provide a notarisation service but also the means by which to check its work is available.

An 'embryonic' notarisation service is instantiated with an identity and the means by which to generate an encryption key pair. Thus each embryo has access to a *key-*

[9]These include, according to Notkin, identifying the actual function of authentication and authorization, identifying sources of distrust and diversity with respect to authentication and accommodating the need for local autonomy within global authentication environments.

generation method and a *notarisation method*[10]. The key generation algorithm is run locally to the notarisation server and the resulting secret key is securely stored[11]. The notarisation server notarises, using its secret key, a copy of its public key along with its own identification. The notarised material is then sent to the trusted functionary that performs validation upon this transaction. The trusted functionary authenticates the sender i.e. the notarisation server, in order to check its access rights and validates the integrity of the transaction's contents. Once the public-key's status has been established, a public-key folio version is produced[12] and made visible in the same way as other transactions. This is an example of a document transaction that involves the update of a single folio[13]. Commit performs a public-key folio write operation.

It is important to note that the notion of key folios does not undermine the DODA abstraction because, as with method folios, a key is a unit of protection for version integrity and each notarisation service to which a key relates is known to the document.

There are two ways in which public-key accessibility may be provided. A notarisation server may attach its public-key which becomes part of the augmented data. The detection of tampering of such notarised data would not clearly distinguish whether it was the original data that was tampered with or whether the public-key may have been corrupted, or whether a secret-key violation may have taken place. However, the 'definitive statement' of each notarisation server's public-key is accessible from the archived and protected public-key folio relating to the identified functionary.

[10] A copy of each method type will be stored and protected within a document version, but for accessibility the notary has a local copy of the notarisation method. Arguably, perhaps, placing a key-generation method within the document instance is helpful only for the introduction of new notarisation servers.

[11] DODA does not state how this secure storage is to be achieved.

[12] The folio version has the notarisation server as the public-key folio's 'attributed user' and the trusted functionary as the 'authorisor', because the trusted functionary notarised the folio.

[13] There is one unusual aspect to this transaction. The doculett of this transaction is not of the form (method, data, result), because key pair generation method must by definition produce a unique key pair each time. If not performed at the notary then there is the problem of distributing the secret key secretly.

5.3 Access Control and Monitoring

DODA is not suitable for secret information for it does not, in its present form, provide read access control. Access control rules, concerning updates, are part of a document's type. The DODA system provides access control to the level prescribed by a document's user group. In keeping with the optimism in the rest of the system, access control is in place to guarantee that 'subversive' actions cannot become visible within a document version. It does not, however, prevent unacceptable actions from being attempted by users within and outside the user group. DODA prescribes, however, that a user offering a transaction proposal for validation has based that proposal on a document version whose integrity has been guaranteed i.e. the user read an archived version. Hence, there is actually a restriction, for the user must obtain the archive reference in a notarised form.

5.3.1 Read Control of Document Archive

No authentication of readers is the normal operation of DODA. However, it would be possible to support more restrictive models of read access control. For instance, a user wishing to employ locks (Section 5.4) requires authentication at the read phase of such a transaction; equally the reading of 'secret' document contents which are stored as fully encrypted text would necessitate the authentication of the read accessing user. The ability to preserve the version's integrity is the main factor determining the timing of user authentication.

Generally reading a document or folio version is an unrestricted right, for version reads do not jeopardize version integrity. As a consequence, document caches can be provided locally to users to offer version (copy) availability. The integrity of cached versions can be checked, for, as we saw in Sections 5.2.1 and 5.2.2, each version has the means by which its own integrity can be checked. A user can prepare a transaction on the basis of any available version (even a version not guaranteed to have integrity). However, when presenting a transaction proposal for validation the user must present proof that the transaction was based upon an archived committed version, as we shall see in Section 5.4.3. A successful check of this is a prerequisite for other forms

of validation check to be attempted. Thus a user who 'experimentally' prepares a transaction without direct reference to the archive, would need to obtain an archive version reference before submitting the transaction proposal for validation.

DODA prescribes that a user offering a transaction proposal has based that proposal on a document version whose integrity has been guaranteed i.e. the user read an archived version. Hence, there is actually a restriction on 'observed' read access, for the user must obtain the archive reference in a notarised form. The route to the archive is through a particular functionary; the, so far, mysteriously mentioned *trusted functionary*, known as the *Visibility_server*[14]. Its role is to issue, on request, a notarised copy of the reference to the Current_version document in the archive[15]. Taking $(data)_{K_v} = N$ to mean N is a notarisation that is the result of notarising *data* using the secret key, K, of the Visibility_server, v, then the base_reference (i.e. a notarised version_reference) sent to the requesting user can be expressed as a notarisation N_{bf} appended to the attributed version reference

$$N_{bf} = ((version\_id, protection\_number, server\_id, user\_id)_{K_v})$$

$$base\_reference = ((version\_id, protection\_number, server\_id, user\_id), N_{bf})$$

Note that the user_id does not need to be authenticated, it is simply a copy of that received by the Visibility_server with the user's request. However, more stringent forms of read access control would require authentication to be conducted at this point[16]. The received notarised reference can be checked for integrity, as with any DODA notarisation, by the user referring to the public-key of the notarisation server, in this case the public-key of the Visibility_server.

The received and notarised reference provides the starting point for a transaction representation. It must appear as the first element in a transaction's *doculett*, as we shall describe in the next Section. Note that read access to the document version

[14] Chapter 6 describes how the archive itself is not maintained by the Visibility_server but by a particular functionary type, the *Archive_gnome*. However, notarised version references, to be passed subsequently to the Archive_gnome, originate with the Visibility_server.

[15] For a document type that has a non-linear version history, the Visibility_server's role is, in principle, the same. But the definition of what constitutes a Current_version in such a system is a more open question with which we will not deal.

[16] Observe that the access control exerted over a user validating a transaction of read restricted data is exactly equivalent to that of ICAP[Gon90], in which a user is authenticated to receive a capability to transact and is also access checked when she exercises that right.

and the document folio level are treated in like manner. A read reference for a folio would differ from that for a document only in so much as a folio_id and folio protection_number are quoted, rather than their document version equivalents.

5.3.2 Doculetts

Each doculett is a transaction proposal. It is akin to an intentions list and is a transaction representation in which there are essentially two components - a reference to the document version upon which changes are based and a record of the proposed changes. The former provides a *base_reference* that is received by a user from the Visibility_server at transaction initiation. The latter is provided by a user who could, as we have mentioned, have to undergo an authenticated identification before receipt. At the end of the read phase of processing, a doculett represents a completed, but unvalidated, transaction.

Once the base_reference has been received, the 'record of the proposed changes' element of a doculett is provided, essentially, by a user or several users working as a *contraction team.* As we have suggested previously and shall make clear in Chapter 6, a number of functionaries will necessarily be involved in the doculett's construction also. However, for the purposes of our doculett description, it is adequate to consider the role of only one functionary type, the notarisation server.

A document type method applied to a folio version provides the basis for constructing document transactions within DODA. The composition of each doculett reflects directly this unit of action, for each doculett is constructed as a series of such units; the ordering of the units indicates the order in which the actions must be performed. This seemingly conventional transaction representation is enhanced within DODA, however, by the association of an action with a user (or more precisely a user_id), thereby providing an *attributed unit of action.* Each unit must be notarised to provide a guarantee of its integrity. Employing the notation introduced in Section 5.3.1, a unit composed of a method call, m_1, on a data folio, d_1, by attributed user u_1, through local notarisation service n with secret key K_n, would be notarised to result, N_1, thus

$$(m_1, d_1, n, u_1)_{K_n} = N_1$$

and the attributed unit, say t_1, would then comprise

$$t_1 = ((m_1, d_1, n, u_1), N_1)$$

Clearly, many actions by method applications on document data folios will involve the input of further data by the user, e.g. $(m_1, (d_1, data), n, u_1)$. However, for simplicity in further examples we will ignore this fact. o

The use of this transaction representation has several implications. The first is related to the personnel involved in a transaction. A doculett can be built up incrementally either by a single user or by a co-operating group of users. In the single user case, it is the initiator who received the notarised version reference who provides all the transaction's units. Thus the previously encountered user, u_1, working as described above, may build up a doculett for a transaction, T, from two attributed units t_1 and t_2[17] such that $T = (base\_reference, t_1, t_2)$. A doculett produced by a team of developers may have each action attributed to a different member of the team.

The second implication is related to each user's domain of *responsibility* and *obligations of trust.* There is a restriction in the case of a co-operative group transaction[18]. It is that the user initiating a co-operative transaction must take overall responsibility for that transaction. Overall notarisation of the units also provides a direct mapping between domain of responsibility and the attribution; we refer to this notarisation as a notarisation for *semantic transaction integrity.* Further, the mapping can be seen as an indication of required levels of trust between the transaction participants. Thus, although the mapping does impose the restriction that co-operative transactions must have a single point of responsibility, it is appropriate for the user to take that responsibility, for she is the only transaction participant to receive her 'transaction base point' from a guaranteed trustworthy source, namely the Visibility_server. Other members of the team propose units of action on the basis of the doculett provided by them by, in the first instance, the initiator[19]. Thus, the DODA abstraction of doculett provides the attribution equivalent of "no taxation without representation".

[17] $t_1 = ((m_1, d_1, n, u_1), N_1)$ and $t_2 = ((m_2, d_2, n, u_1), N_2)$.

[18] This restriction is explicitly stated within access control information held upon folios, as we shall see in Section 5.3.3.

[19] There are further ramification of the doculett representation of responsibility. These relate to the concept of *optimistic validation* carried out by a particular system functionary. Its role is to provide a form of transaction translation process that is similar in concept and result, though not in design, to the coercion mechanism within COSMOS.

The third point is that interestingly, from both the simple data integrity and the semantic transaction integrity point of view it is necessary to notarise a doculett in units and as a whole. Notarising the whole doculett is necessary to prevent apparently legitimate doculetts being constructed by 'cut and paste' of correctly notarised units of other doculetts. Thus, DODA provides semantic integrity at little more cost than that required for simple data integrity. Hence, our user, u_1, will actually build up transaction, T, from t_1 and t_2 above, through the notarisation, N_T, of the attributed transaction,

$$N_T = (base\_reference, t_1, t_2, n, u_1)_{K_n}$$

and the attachment of this to the doculett thus

$$T = ((base\_reference, t_1, t_2, n, u_1), N_T)$$

5.3.3 Modification Control

Although a user group has free read access to document versions, modification access is rigidly controlled by checking the access rights permitted to an accessing user against those granted within an ACL. The access protection provided by DODA ACLs is fine-grained protection (also described in [Low92]) because the ACL represents information about a user's rights to apply named methods to the data folio to which the ACL relates. The form of access control list advocated for a DODA data folio is shown in Figure 5.1. Each entry in the list indicates whether, or not (symbolized by R and N in the list respectively), a named user has the right to call a particular named method upon the folio concerned. However, not only does an ACL indicate the possession and denial of a method right to a user, but also whether that user is granted the capacity to delegate a particular right to some other user, symbolized by D[20]. Such an ACL applies to each folio of a document instance.

The use of the R, N, D rights scheme within ACLs forms the point of connection between semantic transaction integrity (mentioned in connection with transaction notarisation) and its maintenance by DODA's access control mechanism. In operation, the prerequisite for a user to initiate a period of co-operative transaction development

[20]The range of fine-grained permissions owned by a single user can can be seen as contributing to a document processing 'role', akin to those advocated by Leland et al.

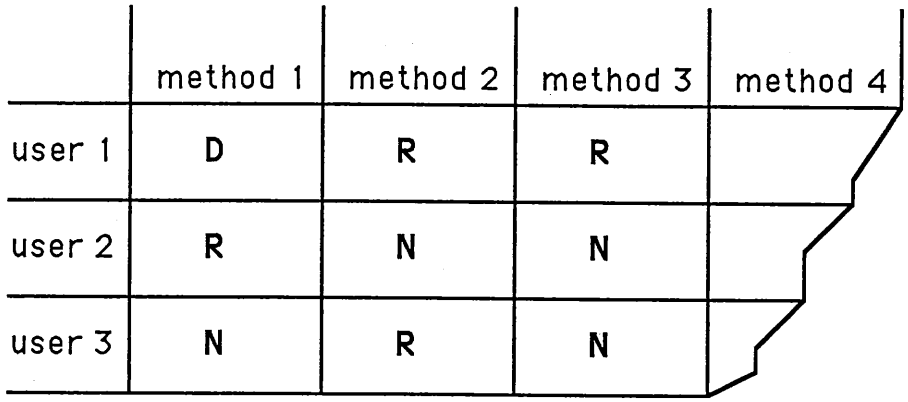

Figure 5.1: A Representation of a Data Folio ACL

is the possession of a delegate right for any delegated operations. The initiating user is thereby explicitly taking responsibility for delegated actions. The utility of this dual approach in representing users' trusts and responsibilities was suggested in Section 5.3.2.

Access rights are applicable at the folio level, and conceptually an access control list exists for each folio of a document. ACLs are actually provided as methods in order that folios containing information that is subject to the same access control regime may share this access information between folios. ACL method application and ACL checking is treated in a manner consistent with all other method applications upon a folio.

In Chapter 4, we indicated that document foliation could be conducted so as to present a set of folios involved collectively in a transaction as a single 'high level' folio and thus provide a *unit of transaction.* The use of such a structuring technique assists the provision of access control for transactions. If, for example, an ACL is applicable to a composite folio that is subdivided into the data and method folios for

implementing a particular transaction, that ACL folio represents access control for a whole transaction. Additionally as ACLs are actually provided in the form of method folios, these folios can be placed within a document type's structure in such a way as to provide an indication of their applicability[21].

ACL folios should, in order to maintain consistency of approach, be associated with all types of folios, data folios, method folios, public-key folios and even audit folios. But there are several exceptions in reality. In normal operation when free read access is available, neither an audit folio nor a public-key folio need such fine-grained modification control. This is because neither folio type is available for modification by a user. In the case of the former folio type, it *must not* be overwritten by any system participant and in the case of a public-key folio, only the Visibility_server and the appropriate notarisation server are involved in the creation of a new version.

Within document types that allow for method changes there is the need for ACLs to control the change. DODA suggests that method change folios themselves should be access controlled, however how far a user group continues this 'controlling the controllers' approach is an issue on which the DODA model makes no practical prescription; it is left to the discretion of each user group.

Another implication of the provision of an ACL as a method is that it may be run by the proposer of a transaction or action as an 'unofficial' check that she has the necessary access rights for the transaction's performance. The fact that a check was made would not be noted within the transaction's doculett; it is simply an act of self-reassurance by a user.

The most common form of DODA modification access control can be seen as optimistic; the functionaries that perform checks on proposed transactions (Section 5.4.3 discusses this) do examine the access control rights relating to a user_id i.e. those pertaining to a claimed identity, but the identification claim is substantiated by authentication only as the penultimate step of validation, once the other aspects of integrity of the transaction have been established. Thus access control decisions can be considered, in the analysis framework given in Section 2.3.3, as the late binding

[21] In Chapter 6 we will refer to all ACL checking necessary for the validation of a transaction as the *entitlement checking methods.*

of rights. Once the doculett has passed the data integrity checks, and checks related to the form of the transaction, it has a high probability of being committed and its changes being made visible. Only when the odds of doculett commit are high is the doculett put before the trusted Visibility_server to scrutinize the checking process and conduct authentication of any users claiming involvement[22]. An access right can only be said to have been exercised by a user after successful authentication because that is when the user's actions are incorporated into the document. Thus, an entry in an ACL does not, strictly speaking, notify of a user's right to apply a method to a folio. For, as we suggested above, a user can 'unofficially' exercise any method application upon a copy of a version. Rather an ACL entry indicates the right of a user to have any effected changes made visible.

5.4 Concurrency control

Concurrency control, like access control, is provided through methods. The nature of the checks undertaken by the *concurrency checking method*, varies according to the semantics of the document type. However, the feature common to concurrency control of all document types is that it is a check upon accesses in the context of the document instance's version history.

DODA operates optimistically, for a user produces a doculett in the optimistic belief that the document version changes it represents are valid and will, therefore, contribute to the document type's development. Opening a transaction involves a user being sent a reference to the Current-version document instance. The user is at liberty to take a copy of this current version into a local cache and use the copy as a basis for document changes. In DODA it is safe to replicate folios and/or document versions. Committed folios are never updated, only superseded, as each committed document forms an *immutable object*, as in [WBMN88], which may be used for auditing purposes. Concurrency checking methods can be applied to a doculett by a user for personal reassurrance, as with ACL methods, but such method applications will not be acknowledged by the system as part of the process of validating a trans-

[22]In a long-term transaction it is possible that a user may not be available to be authenticated. However, DODA makes provision for a form of subcontraction of tasks which may provide the means by which a transaction that involved such a user can nevertheless be committed.

action. An acknowledged concurrency check i.e. one that forms part of the DODA validation process, involves a method application to a doculett by a type of functionary called the *Submission_agent.* To facilitate availability there may be a number of Submission_agents which can make and attribute the validation method calls.

The failure of a transaction to validate merely because it is not based upon the now current version represents an unacceptable processing overhead. Thus in DODA the Submission_agent may be empowered, if the user group so choose, to conduct 'transaction fixing', similar in some ways to the COSMOS coercion mechanism i.e. by getting the most current version_reference from the Visibility_server without direct user intervention.

A doculett may be applied to a Current_version that has superseded the version upon which the doculett is based. Within validation of the doculett, however, reference will be made to all committed document versions concurrent with that doculett i.e. those version committed between the transaction's start and the present, inclusive. Under such circumstances it may be that a doculett represents a transaction that can, in principle, guarantee to maintain the document's integrity, but details such as document references, for example, may be inappropriate. Such details may be 'fixable' by the Submission_agent that may act on behalf of user group as a subcontractor redoing certain operations in a traceable manner. Because of this, it is possible that validation of a doculett superceded by the Current_version will not fail. The doculett may simply be applied to the present Current-version during commitment. The approach has been taken in order to minimise transaction conflicts of the type encountered by longer term document processing transactions.

Optimistic concurrency control and locking are complementary mechanisms, states Mullender [Mul85]. Optimistic concurrency control maximizes concurrency and works best when updates are small and unlikely to conflict. DODA takes an optimistic approach to processing not simply to improve document availability for concurrent transactions but also because validation i.e. integrity checking, is beneficial even in a serial update environment. Validation guarantees not only document consistency but also that development procedures have been followed and have been "seen to be followed". In DODA the optimistic approach benefits a wider range of transaction

types.

If locking were to be used within DODA, the ease with which locks may be applied would be an important consideration in foliation of that document type. However, the crucial issue would be where to situate the locks? Within the Visibility_server? within the document? The use of advisory, non-enforced "locks" that declare the intention of a user to undertake a transaction whose validation is of particular importance to the system would seem an appropriate approach in a processing environment operating upon a "social contract".

5.4.1 Synchronization and Semantics

DODA does not use transaction timestamping. However, validation does require knowledge of the duration of a validating transaction in order that recently committed transactions ie. those that were in progress concurrently with the validating transaction, can be identified and examined, during validation, for conflicts. If a document identifier is implemented as an identification number then as a document develops the document identification number may be incremented with the commitment of each succeeding version. Thus the identifier takes the role of a *version number*, providing an ordering to the document versions.

Doculetts represent possible future states of the document and only become part of the document after successful validation. It *really* is optimistic because if it is possible for the changes to be consistently applied to the *now Current_version*, they will be.

DODA is able to take a less restrictive view of serialisability than conventional optimistic database schemes, e.g. Kung and Robinson[KR81], by taking account of the document's semantics[23]. It will be recalled from Chapter 2 that the use of semantics in conjunction with synchronization is not new e.g. Walpole et al. [WBHN87], Blair et al. [BLM+86], Ladin et al. [LLS90], Garcia-Molina [GM83] and Marsh [Mar91] all report its use. The use of semantics is seen as particularly useful for the commitment

[23]The 'description' of concurrency control appearing in this Section should more properly be called a 'proposal' for synchronization in DODA. The speculations are based upon the validation scheme devised for an *exam document* type, processed within DODA's precursor system. However, there appears to be no major theoretical objections to the proposed scheme.

of long-term transactions. The provision of shadow objects has been favoured to provide object availability during long-term transaction processing, e.g. Marsh [Mar91], Harrison et al. [HOS90]. Both the conflict resolution tool described by Marsh, and Harrison et al.'s scheme [HOS90] suggest that conflict resolution be provided by the directed involvement of users. Additionally, Harrison et al. [HOS90] make use of historical information. In Garcia-Molina's scheme, before processing begins users inform the transaction processing mechanism of transaction semantic types, how to divide transactions into steps, allowable interleavings and recovery information.

DODA advocates an approach similar to that of Garcia-Molina i.e. users provide processing information prior to processing. However, within DODA information such as how to divide transactions into steps is not confined simply within the concurrency checking method but may actually be represented within the document's structure. Concurrency control is one of the considerations influencing the foliation of a document type. Thus in the most simple case, that of a transaction represented as a single method application to a single data folio, the 'placement' of the data folio in the document's folio graph indicates how to apply the concurrency checks. To illustrate, consider the program document introduced within Chapter 3.

5.4.2 Read phase

A user requests a base_reference for a doculett. A single user or a group of co-operating users, each assisted by her local notary, construct a transaction description, a doculett. The end of the read phase is marked by the notarised doculett being passed to a Submission_agent for validation.

5.4.3 Validation phase

A set of transaction checking methods that together provide the validation process are applied to the doculett, amongst them the concurrency checking method. The result of such checks may be successful validation or validation failure. Validation ensures that only non-conflicting and prescribed changes are performed upon a document. The validation method, applied by a Submission_agent, examines whether the sequence

of operations performed upon the data is a legitimate sequence and whether, when the document's Current_version is different from the doculett's base_reference, the operation sequence can be consistently applied to succeeding Current_versions of the document[24].

5.4.4 Commit phase

Commitment is the creation of a *version of the document.* It is the point at which proposed changes become incorporated into the document and therefore visible to other users. The version is guaranteed to be *consistent*, by the consistency criteria defined by the user group, and represents the state of a document at a given instant in time. Hence it is akin to a 'snap-shot' in a historic database. Importantly, producing a new Current_version necessitates recording details of the doculett that created the version. Such details include the identity of the transaction initiator and the order in which methods were applied to named folios.

5.5 Visibility Control

Tentative versions represent possible future states of the document and only become part of the document's version history after successful validation. The transaction validation process prevents a user's mistakes from affecting other users and guarantees that the results of subversive actions will not become visible throughout the system. A version becomes visible once it has been proven to have integrity. Integrity checking is performed by one of the several Submission_agents within the system. A Submission_agent is not assumed to be trustworthy, neither does it conduct authentication upon the user submitting a transaction for validation.

A version's visibility is the result of a single action within the Visibility_server, namely the updating of the *version_reference within the Visibility_server.* Once visible (and prior to the commit of a succeeding transaction), the version's version_identifier be-

[24]The document type may determine that a sufficient condition for document consistency is that the operation sequence is applicable to the now Current_version of the document, whether or not it is also applicable to the intervening document states.

comes the version reference that is notarised and passed to users wishing to initiate transactions, i.e. this version forms the base_reference for doculetts.

5.6 The Audit Trail

The form of doculett prescribed in Section 5.3.2 directly provides almost all the information required of an audit trail. However, access control information has not been discussed in connection with a doculett. The final unit of attribution to become associated with a transaction is the application of an ACL method folio to a doculett. Returning to the doculett representation used previously, we can express the action of the ACL method, m_{acl}, applied by the Visibility_server, v, to the doculett f_d as (m_{acl}, f_d, v, v). The action is performed by and attributable to the Visibility_server. This trusted functionary also acts as the action's authorisor, hence the duplicate v elements (within the expression (m_{acl}, f_d, v, v)). Once notarised by the Visibility_server, producing notarisation N_v, the enhanced doculett becomes the audit trail folio, f_{at}, with the following content[25],

$$f_{at} = ((m_{acl}, f_d, v, v)), N_v)$$

User_ids are noted within the doculett, but what is not provided is a guarantee that a claimed user_id is *rightfully claimed.* However, such confirmation is made just prior to doculett commit and the commitment of a doculett is the evidence of successful authentication of the transaction's participant(s). The doculett of every committing transaction becomes that transaction's audit trail and is stored with the appropriate document version within an *audit folio.* The cost of providing the audit trail is a single folio write operation.

5.7 Integrity Breaches

DODA treats all types of integrity breaches in a uniform manner. Document protection and the management of document processing have been devised with the express

[25]This statement assumes we are disregarding the control information of the folio.

intention of maintaining the integrity of document type instances. Notarisation enables attribution of optimistic actions and authentication of users underpins access control. A 'good guy', a member of the user group, is kept good by a set of functionaries that cooperate with him in document transactions to make transaction checks and control visibility. A 'subversive' may take actions but cannot make those actions visible and thereby interfere with the goodies, for his identity and 'senseless' actions will be detected by the functionaries. However, there are two sources of vulnerability in DODA systems, namely the

- impersonation of a legitimate user and
- acquisition of notarisation secret keys.

Protection is based on the assumption of trustworthy authentication. If an impostor were to acquire a legitimate user's unique identification and were to authenticate successfully, the imposter would take on the rights of the impersonated user. However, the impostor would be unable to perform any but integrity maintaining actions upon a document version. Accepting the undesirability of an intruder's membership of the user group, nevertheless, the integrity of the document instance remains[26]. However, a threatening situation would arise if an imposter were to take on the access control rights of a highly privileged user which conveyed the ability to, for instance, redefine the integrity criteria of a document. This highlights the need for the "social contract" to adopt procedures that necessitate all significant document changes to be meditated by several users from the user group. This also illustrates the potential danger of dynamic method implementations in DODA and indicates the need for method update transactions to be those which require a number of participants to provide human monitoring of the process.

Within DODA we have extended the notions of Gray [Gra85] to suggest that in a secure system a security break is akin to a fault. There are two novel aspects to this. Firstly, the application of ideas from the fault tolerance field to the area of access control violation; secondly adopting a concurrency control rollback technique for more general system recovery. Following the discovery of 'violation' in DODA,

[26]This indicated that the abstraction of transactions provided in DODA is inkeeping with that described by Gleeson[Gle90].

the document is rolled back to a version of development at which the document was known to be 'chaste'. If an intrusion such as that described above is detected and the point in the document's history at which it occurred can be identified, then rollback can take place in a process akin to lazy rollback [JM86]. An impostor would have the access rights of the impersonated user including, perhaps, the right to amend document method folios. DODA would require the document to be developed consistently unless the impostor had gained the rights to change the several methods that enforce consistency checks upon transactions and thus violate the document's integrity. All transactions committed on behalf of the impostor would be attributed to the impersonated user.

The failure of a notarisation is likely to mean that the notarised data has been tampered with. It may also indicate that a notarisation service has been breached and the rescindment of the service's key pair is necessary. DODA's document protection scheme is designed to render the chances of breaching a public and secret key in unison low. Doculett validation will ensure that no committed version has been affected by such a breach. Thus the rescindment of a notarisation service's key entails only the Visibility_service reperforming the transaction described in Section 5.2.5 to produce a new Current_version document containing a new public-key folio for that notarisation service.

5.8 Summary

This chapter has described the means by which a DODA document can maintain its integrity during processing. The control of version visibility is the point of union of data integrity control, user access control and concurrency control. Within this chapter we have also described the means by which document, folio and doculett integrity can be seen to have been maintained. Such explicit displays of integrity are the means by which cooperative document development between untrusted users, through the mediation of a set of largely untrusted functionaries, can be achieved.

The provision of the techniques for integrity maintenance and integrity monitoring is a necessary, but not sufficient, condition to guarantee the progressive development of

document versions that possess integrity. Progress can only be achieved if document transactions can be committed. Thus, the key to providing a development environment in which doculett integrity and, therefore, document progress guarantees can be made is the provision of a document development protocol which directs the use of these outlined techniques. A document processing protocol, and the document functionaries that carry it out, are the central themes of the next chapter.

Chapter 6

Document processing

6.1 Introduction

Previously we have examined the ways in which DODA makes provision for data integrity and semantic transaction integrity. However, a means of enforcing the use of these techniques was not suggested. The objective of this chapter is to demonstrate that a transaction protocol, which implements a set of processing rules agreed by a user group to be appropriate to their needs, can provide guarantees about the integrity of transactions and document versions. In this chapter we acquaint the reader with the way in which DODA achieves cooperative document development and the 'personalities' involved in processing. These contributors to a document's development are *users* and *functionaries*; productive co-operation is achieved by offering a document *development protocol.*

6.2 A Distributed Processing Protocol

For distributed document processing DODA provides a single *Visibility_server* and a single *trusted archive*, which may be distributed, to provide a Current_version document reference to any user wishing to initiate a document transaction. Additionally, untrusted local archives may be maintained. In Chapter 5 we described the way in which open read access to system documents via the recognized archive, or locally

cached document copies, and strict control over document modification is provided by DODA. The protocol presented below is applicable to the optimistic form of processing described in that chapter. Any of a range of more restrictive read access policies could be enforced by the system. Of necessity these would involve the Visibility_server authenticating the identity of the user wishing to prepare a transaction at transaction initiation. The processing rules presented within this section provide a guide to the definition of a range of transaction protocols. However, the protocol outline below is not immediately applicable to restricted forms of access. A user group wishing to enforce such an integrity control during processing would prescribe a transaction protocol that reflected the application's requirements for integrity control.

At its most abstract, DODA's document transaction management protocol can be considered to enforce the optimistic processing cycle i.e. read, validate and commit if validation has been successful (Section 3.3). However, open document read access and a commit process that necessitates a simple version reference overwrite by the Visibility_server (as shown in Chapter 5) result in the protocol being primarily a catalyst for successful transaction validation. Conceptually the protocol is a finite set of targeted messages, each message requiring a particular response from its recipient. In implementation protocol communications may be E-mail contacts. An appropriate response is to apply a referenced document version method to a data folio reference e.g. the Submission_agent receiving a validate transaction message with a doculett applies the validate method to the doculett in an attributable way.

The basis of the protocol is the principle that any document information passing between users and/or functionaries in the system must be notarised, as described in Section 5.2.1. Thus, an audit trail for a transaction is gradually constructed as processing progresses. At any stage it is possible to ascertain what has already been done and which user proposed or which functionary performed any particular action. The audit trail is incorporated into a new version of the document at commit time.

What follows is a brief description of the transaction protocol employed within DODA. The description is applicable to a 'simple' transaction [1]. It does not consider the case

[1] A 'simple' transaction is viewed, in this case, as one without nested subtransactions, that is performed by a single user and validates against only the Base_referenced document because no system updates have occurred since the transaction was initiated.

in which a user explicitly requests particular versions of particular folios as the read-set of the transaction. Neither does the account describe the way in which a transaction involving nested subtransactions, possibly performed by a number of cooperating users, would be handled by DODA; nor does the description cover transactions that validate against multiple committed document versions. In this latter case there may be benefit in supplying any user currently engaged upon a transaction with notification of system updates of the Current_version document reference which accompany transaction commitments.

In this scenario we assume that each contributor has an identity, a local notarisation service, called a *Notary*, to which un-notarised data can be passed safely and that any message sent arrives at its destination. In order to express the protocol succinctly, we use an extended form of the notation introduced within Chapter 5. Thus, a user, u, has access to a Notary, N_U, that uses secret-key, K_U, and has supplied public-key k_u. It will be recalled that the public-key of each Notary in the system, including the Visibility_server's, is available from a public-key folio. Notarisations carried out by the user's Notary are denoted by N_{K_U}, while their corresponding checks are stated as $\overline{N}_{k_u}$. When referring to notarisations take $(u_{N_{K_U}}$ to mean $(u,(u)_{N_{K_U}})$. Likewise for the functionaries that are involved in processing, each will have an identity, a local Notary providing notarisation using a secret-key and the facility to check that notarisation using the corresponding public-key. In the case of the Visibility_server the associated facilities are v, N_V, K_V, k_v, N_{K_V}, $\overline{N}_{k_v}$; for the *Submission_agent s*, N_S, K_S, k_s, N_{K_S}, $\overline{N}_{k_s}$; the *Archive_gnome* attributes are a, N_A, K_A, k_a, N_{K_A} and $\overline{N}_{k_a}$.

Additionally we need a way of expressing information about messages. The content of a message is denoted within square brackets, e.g. $[message]$; a message's type as a superscript prefix, e.g. $^{message-type}[message]$; the source and destination of a message is denoted as a subscript postfix, e.g. $^{message-type}[message]_{u\rightarrow v}$. The final assumption that is made in the protocol example is that all notarisation checks succeed.

1. User sends an *initiate_transaction* request to the Visibility_server.

$$^{initiate-trans}[(u)_{N_{K_U}}]_{u\rightarrow v}$$

2. Server receives the user's request[2] and checks the notarisation;

$$(u)_{N_{K_U}} = (u)_{\overline{N}_{k_u}}$$

3. The Visibility_server produces a *base_reference*, as described in Section 5.3.2[3], and sends it to the user in a *start_transaction* message. We trust the server to provide a reference to the Current_version document.

$$^{start-trans}[(base\_reference)_{N_{K_V}}]_{v \to u}$$

4. User receives reply and checks the notarisation;

$$(base\_reference)_{N_{K_V}} = (base\_reference)_{\overline{N}_{k_v}}$$

5. User constructs a transaction doculett, D, based upon one or more units of attribution i.e. the application of verified and protected document methods to the referenced document's folios. Each unit, t, is of the form described in Section 5.3.2,

$$t = ((method, data, N_U, u)_{N_{K_U}})$$

a doculett, D, is formed by appending attributed units of the form of t, to the base_reference,

$$D = (base\_reference, t_1..t_n)$$

6. When the transaction has been prepared, the user attaches her user_id, the Notary local to the user attaches its identity to the doculett and then notarises this transaction proposal. Thus, the doculett becomes identified with a particular user and is protected by an 'anti-tamper' device 'fitted' by an identified Notary;

$$(D, N_U, u)_{N_{K_U}}$$

7. User sends a *validate transaction* request to a Submission_agent, accompanied by the notarised doculett.

$$^{validate-trans}[(D, N_U, u)_{N_{K_U}}]_{u \to s}$$

[2]Should we decide to extend the protocol in such a way that any user with an outstanding transaction is sent an updated, Current_version reference when a system update occurs, then it will be necessary, at this point, to record the user's identity within an Outstanding_transactions list (possibly having authenticated the user first)

[3]By definition a base_reference is a notarised item.

8. Submission_agent receives the request and checks its notarisation;

$$(D, N_U, u)_{N_{K_U}} = (D, N_U, u)_{\overline{N}_{k_u}}$$

9. The Submission_agent then proceeds to make a number of further checks using a composite *notarisation checking* method. The checking method examines the form of the transaction, in particular its notarisations of the base_reference, received in stage 4,

$$(base\_reference)_{N_{K_V}} = (base\_reference)_{\overline{N}_{k_u}}$$

and the units comprising the doculett, built up at stage 5 of the protocol, e.g.

$$(method, data, N_U, u)_{N_{K_U}} = (method, data, N_U, u)_{\overline{N}_{k_u}}$$

10. The Submission_agent checks that the doculett has been prepared in conformance with the protocol in that it represents consistent notarisation.

$$(m_{n-check}, (D, result = true), N_S, s)$$

The Submission_agent adds an attributed unit to the doculett. The unit attributes the checking method invocation, and result, to the Submission_agent. If $t_n = (m_{n-check}, (D, result = true), N_S, s)$ then $D = D + (t_{nN_{K_S}})$

Note that if there is a notarisation failure at stage 8 or stage 9 then the Submission_agent would return a conflict list to the user. This conflict list would indicate the reason for the failure i.e. which notarised unit did not bear a valid notarisation.

11. The Submission_agent carries out *entitlement checks* by applying appropriate ACL methods to confirm whether the attributed user_id is entitled to perform the operations listed in the doculett,

$$(m_{e-check}, (D, result = true), N_S, s)$$

Again note that if there is an entitlement failure at this stage then the Submission_agent would return a conflict list to the user indicating which method invocations were not permissible.

12. The Submission_agent adds another attributed unit to the doculett. The unit attributes the entitlement checking method invocation, and result, to the Submission_agent and incorporates this into the doculett.
If $t_e = (m_{e-check}, (D, result = true), N_S, s)$ then $D = D + (t_{eN_{K_S}})$

13. The Submission_agent then applies the *concurrency checking* method to the doculett in order to ascertain whether this transaction conflicts with any concurrent and already committed transactions[4],

$$(m_{c-check}, (D, result = no\_conflict), N_S, s)$$

The failure of a concurrency check results in the Submission_agent returning a conflict list to the user indicating the folio versions with which there was an inconsistency.

14. Assume that concurrency checking succeeds. A notarised entry is added to the doculett that attributes the concurrency checking invocation, and result, by the Submission_agent.
If $t_s = (m_{c-check}, (D, result = true), N_S, s)$ then $D = D + (t_{cN_{K_S}})$

15. Part of the transaction's validation has been conducted [5]. The Submission_agent prepares a new document version using the vetted doculett and the *version create* method provided for the purpose.

$$(m_{v-create}, (D, result = version\_reference), N_S, s)$$

16. The created version, say t_v, which we assume to be correctly prepared, is notarised by the Submission_agent's local Notary, $(t_{vN_{K_S}})$ and sent, in an *archive-transaction* message, to the Archive_gnome,

$$^{archive-trans}[t_{vN_{K_S}}]_{s \to a}$$

17. The Archive_gnome receives the message and checks the notarisation,

$$t_{vN_{K_S}} = t_{v\overline{N}_{k_s}}$$

[4]It will be recalled from Chapter 5 that the concurrency checking method will resolve all but semantically inconsistent transaction schedules.

[5]If any of the checking methods fails, a notarised entry that attributes that checking method invocation, and result, to the Submission_agent is nevertheless added to the doculett and the augmented doculett notarised. However, the result in the case of failure includes a list of folios (i.e. folio references) with which conflict was detected.

18. The *version archive method* of the base_referenced document is invoked and the call attributed to the Archive_gnome[6],

$$(m_{v-archive}, t_v, N_A, a)$$

The version archive method conducts the following,

- securely archives the new version, directing folio sharing where applicable and calculating new folio protection numbers (Section 4.2) where necessary [7]. But this pre-committed document version will initially have its document protection_number and authorisor fields filled by the Archive_gnome;
- stores the version at an identified location, i.e. returns a reference for the archived document,

$$(m_{v-archive}, (t_v, version\_reference), N_A, a)$$

- attributes the archiving to the identified Archive_gnome.

$$(m_{v-archive}, (t_v, version\_reference), N_A, a)_{N_{K_A}}$$

19. The Archive_gnome has the document's archive reference notarised, $(version\_reference, N_A, a)_{N_{K_A}}$ and sends this to the Visibility_server within a Commit_transaction message,

$$^{commit-trans}[(version\_reference, N_A, a)_{N_{K_A}}]_{a \rightarrow v}$$

20. The Visibility_server checks the notarisation of this message,

$$(version\_reference, N_A, a)_{N_{K_A}} = (version\_reference, N_A, a)_{\overline{N}_{k_a}}$$

[6] Alternatively the preparation of a new version, using the version creating and archiving methods, could be performed by the user directly. The Submission_agent's role would then be that of a version checker that authorizes the version. However, the presented protocol steps are preferable for reasons of efficiency and consistency of approach. To illustrate this consider the case of a doculett validating against a series of versions. Although the doculett may detail a validatable sequence of actions, the transaction must be applied to the now Current_version, rather than the version cited within the doculett. Such a case would involve both the user and the Submission_agent running the version create method within the same transaction.

[7] DODA provides the equivalent of a copy-on-write mechanism [Mul85] for folios affected by document changes but have not necessarily had their own contents changed i.e. folios whose fprotection_numbers have changed perhaps due to a change in a sub-ordinate folio's contents. As a consequence, an error must be signalled if there is an attempt to overwrite a folio protection number of a previously archived folio during proposed version archiving. This may, for instance, indicate a violation of the version archive method.

21. The Visibility_server now checks the checkers by invoking the *protocol checking method* that checks that notarisation, entitlement and concurrency checks have been performed using the prescribed methods and that the notarisations of the method invocations are valid. This involves using the public-key folios relating to the Submission_agent and the Archive_gnome.

    If the protocol check fails then the doculett has been prepared in an 'unconventional' way and DODA cannot vouch for the resulting document version's integrity. Hence a conflict list is sent to the user outlining the reasons for the failure.

22. Assuming all entry notarisations are correct. The Visibility_server now *authenticates the users* that have participated in the transaction's development.

    If the authentication of any user should fail then again the user that submitted the doculett for validation is informed.

23. Assuming authentication succeeds. The Visibility_server checks the Base_reference of the new proposed version against its Current_version reference. If the references are the same (i.e. the proposed version has been based upon the Current_version) then the server completes the archived document version by entering its identifier, v, as authorisor and calculating and inserting the document's Protection_number into the version's index page;

    If the new version has been constructed upon an older committed version then, in effect, validation is incomplete. Since we have established that the new version has integrity because it has been prepared in accordance with the 'social contract' and we assume a low degree of conflict due to concurrency the probability of validation failing at this stage is low. Therefore DODA allows the Visibility_server to perform the role of a Submission_agent at this stage and conduct the concurrency aspect of validation between the proposed new version and those Current_version documents committed since the proposed version was archived. Failure of this subsidiary validation causes a conflict list to be sent to the initiating user. Successful validation causes the Visibility_server to provide a Protection_number for the new version[8].

[8]Successful validation may involve some amendments to the document.

24. Once the new version has been 'tamper-proofed', it can be made available to the user group throughout the system. Hence, Visibility_server updates its Current_version reference with the new document version's identifier and the version becomes *visible.*

25. The Visibility_server sends a *committed_transaction* message to the user that constructed the transaction to confirm that it was applied to the document on her behalf.

$$^{committed-trans}[(v)_{N_{KV}}]_{v \to u}$$

26. The user receives message and checks its notarisation to confirm that her transaction has committed.

$$(v)_{N_{KV}} = (v)_{\overline{N}_{kv}}$$

In the early stages of the protocol no user or functionary authentication takes place, although notarisation checks are frequent. Once the user has received a version reference, the base_reference, from the Visibility_server and the notarisation of that reference is shown to be 'good', then access to the securely archived document provides access to a range of system-provided, document verification methods, that are themselves verifiable. The user is offered a veritable 'battery of reassurance', for instance the version's Protection_number is part of the base_reference and so the user can check this. It is a requirement of the version create method that it creates a new document version *only* if the doculett includes entries reporting the successful application of the notarisation, entitlement and concurrency checking methods. The version produced must have its new, and therefore unshared, folios attributed to the named user responsible for the corresponding doculett unit and authorized by their identified Notary.

The result of successful transactions is the development of a sequence of versions of the document within the archive. The transaction management service comprising protocol, functionaries and methods, prevents the incorporation of unauthorized document transactions into document development. The division of labour within the management service has been chosen so as to provide a generic document processing facility. All document types will be developed with the aid of users, Notaries, a Visibility_server, an Archive_gnome, Submission_agent(s). Actions that require a detailed

knowledge of a document's structure for their performance are implemented in the form of document types' methods that are called by users and functionaries.

In Chapter 3 we declared that for every type of document instantiated in the DODA environment, a pre-defined set of methods must appear as a subset of a document type's methods. The identified subset of methods are those used by the functionaries within transaction validation and commitment, namely notarisation, entitlement, concurrency and protocol checking methods and version create and archive methods. Such methods, used in conjunction with one another and co-ordinated via a transaction protocol also prescribed by a user-group within a method, maintain document integrity. Note that there is an implied restriction on the application of the protocol checking method; it is that this method must be applied by the trusted functionary, the Visibility_server.

6.3 Functionaries

Functionaries, as stated in Chapter 3, are small pieces of software[9] that invoke a specific set of methods of documents. By providing a uniform interface, the document *index_page*, to all methods of all document types, the functionaries are applicable to all document types' methods. Their role in document processing is that of *detectability enforcement.* Any action performed by a functionary must be detectable and will be traced by the trusted server before a transaction's commitment. In turn, each functionary ensures that a user's actions are traceable. Notarisation, and the accessibility of protected notarisation checking facilities, is the key to this provision.

6.3.1 Users

The *user group* is a geographically distributed group of people who work cooperatively upon producing a document. The group is responsible for detailing a document's design by specifying the document's structure in terms of the document type's foliation, its methods and the integrity criteria that must be upheld to facilitate progress. Once

[9]By inference, a 'small' piece of software is less easy to subvert because its size precludes the concealment of 'trojan horses'.

a document type has been instantiated, collaborative document development may require user to user communications, perhaps by E-mail, hence each user has a unique identity that can be presented, within a notarised message, to other participants in the system. The user's local notary is trusted to notarise data as received, including identity information. But users are not trusted to maintain encryption keys for use within document operations[10].

The user group may include not only people but also utility programs. For instance, a utility such as a standardized compiler may be regarded as an attributable user that can fulfill only one role, that of a subcontractor responsible for folio compilations. The alternative would be to provide such a utility as a document method. Within DODA it is considered more effective to treat utility programs, such as compilers and linkers, as registered subcontractors, rather than compiling the utility software into a DODA method folio for reasons that are discussed in Section 6.3.4. Whether or not DODA has to trust that a user has presented an internally consistent transaction proposal depends upon whether the user group is able to specify internal consistency constraints and thus provide some form of internal consistency checking method alongside the concurrency checking method[11].

6.3.2 Visibility server

The Visibility_server provides document version visibility and safeguards document instance versions, by 'guarding the gate' to the version archive. The Visibility_server's primary task is to maintain a Current_version reference that refers to the most up_to_date document committed within the version archive. Like all versions visible within the archive, the Current_version document instance has guaranteed integrity. A base_reference provided directly to the Visibility_server, as described in Section 5.3.2, is the only means through which access to the archive can be made. A verifiable base_reference is a user's ticket to replicate whole document versions or folios for storage in a local cache during transaction preparation. The Visibility_server's

[10]We maintain a separation between the issue of secret key use within an authentication process and encryption utilized for integrity in document processing.

[11]Internally consistent program document folios might, for example, involve consideration of whether all the nested functions are used within a module or whether the programmer has correctly used by value and by reference parameters.

provision of base_references to documents or folios can be seen as the equivalent of the management of user views of the data.

There is a single Visibility_server for each DODA system instantiation. However, the Visibility_server need not represent a bottleneck because DODA provides local version caching and a validation process that makes possible the commitment of transactions based upon document versions older than the Current_version. The replication of self-protected document versions is the key to availability in DODA.

The Visibility_server fulfills two other functions. The first is the calculation of a document version protection_number for a new version created for a committing transaction. The second is the certification of public-keys of notarisation services. The server is the source of trust within DODA and by this act the Visibility_server effectively delegates its authority for notarisation to local agents, the Notaries. Thus, conceptually a Notary is a subcontractor for the Visibility_server.

It may seem to be an inconsistency in the design of DODA that a Visibility_server carries out several different tasks. However, we have been using the term 'Visibility_server' rather loosely, thus far. The name actually refers to a version visibility manager that maintains the Current_version reference and, co-located with this, a 'super' notarisation service. Thus the provision of the Visibility_server mirrors that of other functionaries within DODA; each functionary (and each user) has a finite set of tasks that it is entitled to perform, which are defined within document methods, and all information pertaining to the performance of these tasks must be attributed and notarised by a Notary, local to the functionary.

6.3.3 Notary

Provides a tamper-proofing mechanism for individual folios, doculetts and doculett parts by calculating and encrypting a checksum over the data. Each notarisation service is trusted to maintain a secret key for this purpose. There are multiple Notaries in each DODA system for a local Notary must be supplied for each user and each functionary.

The issue of locality of notarisation services in relation to users and functionaries is important for we need a means by which a user can pass her data to a Notary. Difficulty arises if we cannot be assured that the data on which the user requested notarisation, is the same data that is actually notarised (because we cannot trust the integrity of data until it has been notarised). The issue has yet to be fully explored and represents an important piece of further work in relation to DODA. The work of Low [Low92] suggests a means by which to provide users with notaries within the context of a UNIX operating system, however, providing Notaries in an open environment is more problematic. To co-locate notarisation services alongside users has implications for the vulnerability of that notarisation service. To separate users from notaries requires secure channels between user and Notary, either dedicated channels or necessitating users having secret keys.

6.3.4 Subcontractor

A doculett can be built up incrementally either by a single user or by a co-operating group of users, what we referred to as a contraction team in Chapter 5. This form of working represents a form of non-trusting sub-contraction in which there is a demarcation of responsibilities between the initiator of transaction and subcontractor delegated particular tasks. The user initiating a co-operative transaction must take overall responsibility for that transaction by virtue of receiving her processing information, the base_reference, from a trustworthy source. Other members of the contraction team do not.

However, DODA offers another form of sub-contraction, referred to as *trusted sub-contraction*, which has been introduced to overcome the problem (identified by both Garcia-Molina [GM83] and Walpole et al. [WBHN87]) of having to provide special purpose tools within processing systems that make use of semantic information. Trusted sub-contractors can be seen as single facility users.

Within DODA it is considered more effective to treat utility programs, such as compilers and linkers, as *registered subcontractors*, rather than compiling the utility software into a DODA method folio. One reason for this is that such standardized software is likely to be relatively static software and is less appropriately made available via a

document tailored for update operation as this would incurred unnecessary overhead. Also, when such software is upgraded with a new release, there may be a period during which the software users, as well as the software's developer, produce *bug reports.* To provide software with these characteristics within a document itself would have implications for the form of integrity guarantees it is possible to make. Another, and more important, reason is that by adopting this approach DODA can provide a flexibility of service equivalent to that achieved by English et al. [PEea90]. Their system is not open but makes use of LISP run-time binding facilities to integrate, on the fly, any required document processing service that is currently running above the system wide interpreter. DODA achieves this by allowing the flexible introduction of users into the user group by the use of role_ID capabilities, without the need for all DODA sites to run in a LISP environment. Additionally DODA can provide access control to subcontractors, not provided by English et al.

It may be that an authentication service, acting as a registered (and of necessity, trusted) sub-contractor, could be made available to users that wish to collaborate on a particular transaction. It may be appropriate to employ an editing tool as a registered subcontractor. The user would make 'naive' document edits, present these to the edit tool to be translated into a doculett to be sent for validation.

6.3.5 Submission Agent

The Submission_agent is the functionary responsible for conducting the validation of the user-provided part of the transaction. It performs conflict detection and, where possible, conflict resolution. The conflict in question is a conflict between the prescribed (by the user group) *integrity standard* and the integrity standard observable within a doculett. The Submission_agent examines the contents and form of, and the personnel involved in a transaction. If unresolvable conflict is revealed, the Submission_agent informs the transaction initiator of the nature of the conflict in terms of folio version identifiers in which, or with which, conflict has been detected. Thus if no conflict is detected, updated folios need only be archived, or will need to be created and archived. If resolvable conflict is detected the Submission_agent will follow the user-provided guidelines for resolution of that form of conflict and will create folios

appropriately, for archiving.

6.3.6 Archive Gnome

The Archive_gnome copies and sends out visible document and folio versions on the request of users. The archive is a simple depository of document instances and document folios. The version history of a document is stored for the sake of recoverability and auditing. Archiving is provided efficiently by allowing document versions to share folio versions, provided that such sharing does not undermine document integrity. There may be proposed document versions within the archive that are not yet visible beyond the realm of the Archive_gnome and the transaction by which they were created; because a version's folios are stored within the archive before the version is committed.

6.4 Summary

In this chapter we have introduced a simple transaction protocol that illustrates the way in which collaborative development of a DODA document could proceed. DODA prescribes that a specification of such a protocol (along with protocol checking facility) be made as part of each document type's definition. Protocol checking may usefully be implemented as a method folio in a similar way to the provision of notarisation checking, access checking and concurrency checking. The crucial role of the protocol is to coordinate the use by users, functionaries and also registered subcontractors, of the document processing facilities made available through a document type.

In the preceding several chapters we have described the components of the DODA architecture. The architecture provides a generic basis, a toolkit, for the instantiation and development of tailored document types defined by user groups. In the next chapter we will examine the implications of such a system, not only for document processing, but also in relation to wider issues of computing not previously associated with the realm of documents.

Chapter 7

The Implications of DODA

7.1 Introduction

Having described the document processing system DODA and the literature that motivated its development, we will now examine whether DODA may throw new light upon any of these works and suggest further work within the area of open distributed document processing.

7.2 The Implications

Within the optimistic framework of DODA we offer solutions to several problems which have been identified in the literature. In particular, DODA offers a resolution of the paradox between security and availability in distributed open systems through the provision of fine-grained access control, document protection via notarisation, 'safe' duplication of documents and functionaries allowing the use of untrusted servers and a validation process that examines document integrity.

In common with other systems DODA adopts an optimistic approach to processing to offer initial access availability of data. However, unlike other systems DODA will validate transactions in non-concurrent update conditions. Validation checks involve not only concurrency checks but also checks upon the process and procedures by which

a transaction was constructed. The crucial point is that optimism is used precisely because we wish to conduct such validation checks upon completed transaction proposals, called doculett. It is this approach that enables DODA to maintain document integrity and thereby guarantee that the defined "social contract" has been upheld by all parties, even those who have not 'officially opted in'.

In a wider context DODA can also be seen in a number of different lights. It offers support to the view, expressed in [WBHN87]and [BLM+86], that the usual layered approach to distributed system design[1] does produce a paradoxical relationship between certain functions within systems e.g. data availability and security. The DODA design is evidence that the paradox is not an inevitable result of distributed processing. This thesis suggests, in the light of Gleeson's work, that the 'divide and conquer' approach may itself produce the paradox between the provisions for certain functions because traditional systems have been 'divided' unwisely.

With this in mind, we point to the benefits of abstracting a systems using the principles of abstraction outlined by Gleeson [Gle90], with their emphasis on the distinction between *necessary* and *unnecessary* detail. Following on from this, DODA can be seen as an example of atomicity as stated in [Gle90]; for instance, the concept of a folio as a unit of semantic granularity provides a direct correlation between a document's specification and the implementation of that abstraction. Equally, DODA suggests that some features of traditional systems that have been treated as integral, file storage and concurrency control for example, may not need to be treated as such.

DODA also sheds light on the question raised by Horn, in [Hor89] about the possibility of building an integrated development environment based on object invocation. To unify distribution, persistence and general programming within object oriented systems is seen by Horn as a desirable goal; the development of DODA would suggest that it is also an achievable goal within the area of document processing.

DODA provides a set of guidelines for the design of a range of processing systems for structured document objects. Such an approach is broadly applicable. For example, the Amoeba file system is stated by Mullender [Mul85] to form a basis for the design of

[1] An approach typified by general purpose client-server systems constructed on 'divide and conquer' lines with the separation of functions such as access control and scheduling.

a range of file storage application tailored by users to run above the Amoeba operating system; DODA suggests a means by which to reason about the most appropriate form of structuring for such applications. Moreover, the definition of 'document' taken by DODA is very wide and therefore the approach to distributed systems design that has been developed within DODA appears to be very broadly applicable. The DODA architecture suggests a way in which to provide a general purpose document processing facility that can, nevertheless, be tailored for a particular implementation of document type. Thus DODA makes provision of special-purpose document processing facilities within a general-purpose document architecture. Once instantiated a document type can provide further flexibility because DODA allows document methods to be changed in a controlled manner. Thus DODA provides a form of dynamic sub-typing of documents.

DODA assumes that a user group will devise (and express) document processing procedures that are amenable to validation. In addition, DODA offers heterogeneity by dynamic document sub-typing through the use of multiple method implementations. It also facilitates collaborative working and the use of application software through delegation of access rights to subcontractors. In each case it is the user group that bear the responsibility for these. The specification of document requirements does need further work. However, the approaches of [GM83], [WBHN87],[Mar91], [?] for example, suggest that user specification of semantic information is not an intractable problem in relation to documents.

Our purpose in inferring, rather than explicitly stating, a definition of the term "document integrity" is to convey that we regard integrity to be a document type-specific concept that relates to, unifies and directs the application of various forms of process management used within collaborative document processing. The concept of integrity is our rebuff to the division of function suggested with what we have termed the 'traditional approach'. Integrity is an embracing concept because it represents a user group's interpretation of the semantics of a document application; this interpretation is expressed within the document's methods and is made available to the functionaries that process a document type. Functionaries are the agents that uphold the application's semantics during processing by forcing any and all change proposals to be made in a manner amenable to checking i.e. to be traceable. Successful validation

of document changes provide users with unforgeable guarantees that the semantics of that document type have been respected. Semantics, however, are open to wide interpretation. DODA provides a framework within which it is possible to begin to reason about the application of semantics within a distributed processing environment.

The thorough going use of optimistic processing within DODA is one of its novel features; not only is there optimistic concurrency control but also optimistic access control. DODA functionaries vet transactions in the optimistic belief that the identity claimed by a user and attributed to the user's actions, is the corresponding identity which will be proved during authentication. It is this optimism that allows DODA to provide a single trustworthy authority within the system without inhibiting availability of services. Service availability, in the form of methods availability, comes hand in hand with the availability of documents.

There is acceptance for the view that application synchronization can be related to, or make use of, application semantics. However, the notion of security-related semantics is less accepted. DODA illustrates the usefulness of the notion of security-related semantics. DODA supports the definition of collaborators' roles (as do [Oli90], [NKCM90], [LFK88]) in terms of which operations a user is permitted to perform. However, DODA places no arbitrary restrictions upon the nature of a user's role, as do these other systems. DODA roles are expressed through the medium of folio ACLs which, because of the representation of permissions as i) rights to apply methods to folios and ii) the right to subcontract method rights, allows DODA user groups to express a very wide range of roles. Co-ordination of collaborative work is performed through the access rights, thus the forms of collaboration that DODA can offer are extensive, checkable and enforced.

Note that DODA makes the ability to subcontract tasks an explicit right and use of this right during a transaction i.e. the production of a transaction by multiple users, will be checked during validation. Thus the notion of a user's responsibility for subcontracted work is enforced.

The inability of 'compiled in' systems to accommodate run-time binding was criticized by English et al. [PEea90] and drove the Interleaf System to an implementation built upon a LISP platform which must be running at every site in the system. Their

system, though providing run-time extensibility, makes no provision for concurrent processing, anti-tamper protection or access control of documents. Thus DODA offers additional assistance in processing despite being a compiled in system.

7.3 Suggested Future Work

The DODA project suggests a number of exciting avenues of future research, both research directly related to the architecture as extensions or clarifications of the existing facilities and also research directed towards answering questions raised by the DODA project.

The thesis makes the assumption that DODA applications have access to a secure channels between a user and a local notary and that the Visibility_server has access to a trustworthy authentication services. The means by which these can be provided efficiently has not yet been considered. This represents important work in the realization of DODA.

The abstraction of distributed document processing illustrated by DODA, in particular the production of relatively *stateless* servers, in the form of the functionaries, raises the question of whether the provision of such server processes is a desirable goal of a system abstraction? We do not offer an answer to this question, however the utility of the approach within this system does suggest that it is a question worthy of consideration.

A change in the relationships between folios may be described as a change to the document's semantics and therefore constitute a change to the document's type. Such adaptability could be considered as a controlled form of dynamic sub-typing of document objects. DODA suggests that such functionality may be provided by the use of different navigation methods, each appropriate to a particular folio configuration. By this means DODA may combine flexibility of implementation with guarantees of document integrity. However, within the current version of DODA such provision has not been made; it is worthy of further investigation.

Since navigation methods offer a means of presenting a variety of more or less re-

strictive views of a document's structure to users, we propose an investigation into the provision of read access control as a piece of future work related to the DODA project. Read access control may be offered, for example, by insisting that a navigation transaction[2] is a prerequisite for receiving a (partial) copy of a document version.

The 'proof' of integrity of a folio or document is a valid protection number. The principle behind protection numbers is that any change to the content or control information of a folio will result in a corresponding change to its protection number. The use of protection numbers has implications for information sharing; the implications need to be more fully examine. The crucial issue when processing shared folios will be maintaining document integrity across instances when the relationship between the folio will not necessarily be simply hierarchical (i.e. document structures may be directed acyclic graphs) and may even be dynamic.

The use of dynamic document structures is advocated within DODA and its provision is made through the medium of multiple navigation methods, each appropriate to a particular, or a particular set of, document states. Providing a model for an application's time-variant semantics, as a basis for the systematic provision of processing facilities may be useful.

PREP uses hypermedia. The DODA model does not appear to preclude the notion of a document being a multi-media based entity. With such document formats, the investigation of 'dynamic' would enter the realm of medium changing. Careful consideration and further investigation needs to be made into the semantic implications of such dynamism of document folio representation.

The use of multiple document transactions requires considering i) the provision of links between instances[3], ii) sub-typing and iii) implications for the placement of archives at nodes e.g. perhaps it may prove necessary to impose the restriction that sharing instances must be managed within the same archive[4].

[2]i.e. the application of a navigation method to a document as a transaction that is attributed, validated and if successful recorded in the document's audit trail.

[3]May also requires consideration of links between document types if different types are permitted to share (inherit) folios.

[4]This is not an unreasonable restriction for it is implicitly imposed upon instance's versions currently. The application of such a restriction would result in a uniform treatment of folio sharing

Within DODA each individual folio is provided with a folio base_reference. This provision facilitates folio sharing between versions. However, additional benefits may accrue, such as ease of processing dynamic document structures and provision of non-linear version histories. Examining the feasibility of providing such functionality within DODA provides an avenue for future work.

In the area of access control, Troy et al. state [TKS88] that access control "encompasses the definition of the rules governing the allocation of access privileges". As well as examining access rules in relation to a variety of (primarily) textually based document types, work is required to establish the principles behind the fine-grain allocation of system resources and ways in which such allocations requirements can be expressed. We are suggesting an examination of the semantics of resource use e.g. within an operating system environment. The changing requirements for accessibility to an object over time is an area of particular interest in applications within which data may have a 'life-cycle'.

The work of Low on UATP [Low92] has been strongly influenced by the approach of DODA and suggests a means by which to prototype a DODA system.

Related to the above work, and encompassing it by the definition used within DODA, is the issue of the capture and specification of application semantics. The work assumes that a document application's semantics may be specified in such a way that these may be translated, perhaps in an automated way, into integrity principles. DODA requires semantic information about an application to be translated into an implementation-useful representation, for instance as the foliation of a document in such a way as to localize method applications and minimize the probability of conflict. However, DODA has not established the extent to which this is a realizable aim for any but simple application examples. Further investigation is necessary.

7.4 Summary

The main implication of DODA is that adopting a unified approach in the design of open distributed systems provides a greater range of functionality than is possible within DODA.

through a divide and conquer strategy. The unified approach DODA advocates is to provide a type specific notion of integrity for an object type and the enforcement of rules governing the object's integrity within validation. As a consequence, an optimistic approach to processing is valuable in even a non-concurrent processing environment.

Much further work has been outlined; the proposals focus particularly upon the issue of representation of application semantics in order to provide a notion of integrity for resources other than, and in addition to, textual documents. Our interest in applications beyond document processing indicates the wide applicability that the DODA approach appears to have.

Chapter 8

Thesis Review and Conclusions

8.1 Introduction

The intrinsic value of adopting a coherent approach to controlling change is presumed in previous works and the usual layered approach of separating concerns has noted flaws. This thesis advances the proposal that access control violations, concurrency conflicts, semantic consistency failures, deliberate tampering and accidental corruption should all, for the purpose of detection and prevention, be treated in a uniform fashion; namely as violations of a document-specific notion of *integrity*. From this viewpoint the thesis has offered an approach to distributed document structuring, and a model for highly distributed processing of such structures.

8.2 Thesis Summary

The thesis began by describing the previous work and ideas upon which DODA is built. DODA has been designed as an object-based system to facilitate secure, yet cooperative, document development. The abstraction of document objects and functionaries utilized within DODA would appear to offer scope for the provision of a wide range of processing systems.

A document object is a structured entity composed of sub-components called folios. A

folio may be composed from various forms of data, including user data, representations of document methods or public encryption keys. A document's folios may be concurrently processed by transactions that are validated for integrity before their effects can become visible to the members of a user group. The integrity checks conducted during validation are not simply to investigate concurrency conflicts but examine the personnel involved and whether the correct process of transaction preparation (i.e. the right procedures) were followed. DODA has been able to employ, in a novel yet coherent manner, known techniques from the fields of data protection, access and concurrency control to provide security and availability for collaborative distributed document processing.

Read access to documents is uncontrolled within DODA. However, a user group may wish to impose more restrictive forms of read control; DODA does not preclude this but will not assist. DODA assumes availability of document versions on 'untrusted' serves; therefore any data that requires read protection must be encrypted whenever that document is outside a secure environment. Committed document versions are protected from modification by notarisation and access control list structures associated with each document folio. Access control is based upon the use of ACLs that specify users' permissions for methods[1]. However, the structuring, or foliation, of a document type may allow for the provision of access control at the transaction level when foliation is performed in such a way as to allow, in the most simple case, a transaction to be the application of a single method to a single data folio[2].

The primary goal of DODA was the development of a system to facilitate collaborative development of documents as a means of investigating the paradox between availability and security. The power of the DODA system lies in its ability to provide such a service. The key to providing collaborative processing is the form of transaction representation that DODA uses. A transaction is expressed as a number of 'units', each of which must be attributed to a user and each of which may be attributed to a different member of the user group; the user group includes not only human users but also all DODA functionaries (and, perhaps, pieces of application software). The

[1]Such protection constitutes fine-grain access protection. It will be recalled that access protection also incorporates the idea of semantic protection and concurrency protection (usually performed by locks).

[2]Bear in mind that a folio (data or method folio) may be a composite object. Access to the composite object implies access to its component parts.

attribution of a unit is conducted in an unforgeable way and renders each user's role traceable and therefore checkable. Thus a transaction may express a validatable 'joint venture'. This is because it may be composed of several actions performed by different users and the transaction itself must successfully pass through validation before its results are made visible.

Public key encryption is used within notarisation for document protection. The thesis proposes a new and elegant form of key management to facilitate this provision. In particular DODA provides a form of 'free-standing' public key certification.

The form of transaction proposal advocated, the doculett, ensures data integrity using the anti-tampering devise of notarisation. It also offers traceability of user actions during processing and, therefore, semantic integrity. Semantic integrity, it will be recalled, is the type specific concept that relates data (folios), methods, the user performing a method application and the ordering of performed operation. In addition DODA offers auditability of transactions after commitment.

The serialisation of concurrent transactions is performed by an optimistic concurrency control mechanism. It extends the notion of serialisability used in previously proposed schemes and guarantees that all non-conflicting transactions will eventually be applied to the document; this relies on the definition and use of document semantics. The use of doculetts enables DODA to ameliorate the usual bad consequences of optimistic processing i.e. the need to repeat work following a failure to commit a transaction. The structuring of doculetts into attributed 'work units', combined with the return of a conflict list to a user after an unsuccessful validation, allows the user selectively to reuse 'work units' in the construction of an amended doculett.

8.3 Conclusions

The DODA approach is likely to be of increasing interest in the future firstly because of the growth in the use of open distributed systems and secondly due to the growth of systems required to provided collaborative and auditable processing; for example, in the field of commercial software engineering in which there is increasing stress on

the accreditation and certification of organisations. For a software company to gain certification of conformance to the quality standard ISO 9001 (a guide to 'best practice'), for instance, that company must have in-house procedures that conform with the standard's guidelines and must provide evidence that these procedures are used in everyday practice. Conducting software development under a system like DODA would enforce the use of 'best practice' and permit auditing of software projects.

DODA resolves the paradox between security and availability within open distributed system that are uncontrolled by any central authority. Central to the resolution is the use of optimistic processing and the rigorous control of document version visibility. The system provides unrestricted local access that avoids integrity problems because it also offers firm and unforgeable guarantees of version integrity. Such provision is related to the notion of localization of trusts.

The work demonstrates that adopting the unified approach to distributed processing outlined above, can facilitate protection and availability and, in addition, accommodate heterogeneity. The DODA abstraction provides a document architecture that is general purpose (the functionaries can service many document types) and yet can instantiate many particular document types. This is because the type specific processing information is held within the document instance itself as method folios; a DODA document is a self-managing object. The user group is responsible for the definition of a document object type i.e. the 'template' for instances, including the representation of the folio structures, the interpretation of methods and the prescription of acceptable method orderings. DODA takes a unified approach to a variety of integrity maintenance mechanisms (e.g. types and orderings of updates, access control rules) because the definition of an application's integrity conditions is based upon the semantics of that application.

DODA demonstrates how access control and protection of object instances can also be encompassed under the same umbrella, namely optimistic processing. This work has extended the general notion of optimism, for DODA employs the principle in relation to concurrent processing of transactions and also with regard to access control. Access and concurrency checks are performed at validation.

8.4 Summary

DODA suggests that the provision of a type specific, semantic notion of *integrity* for any object type allows rules for the maintenance of that integrity to be devised. These rules can then be enforced through a validation of processing that is conducted optimistically. Thus validation is a process that is of value in even non-concurrent processing environment. It is this approach that facilitated the development of DODA as a system that resolves the paradox between security and availability within open, distributed computer systems.

The Epilogue

Ode to a Ph.D.?

What I expected was
Thunder, fighting,
Long struggles with men
And climbing.
After continual straining
I would grow strong;
Then the rocks would shake
And I would rest long.

What I had not forseen
Was the gradual day
Weakening the will
Leaking the brightness away,
The lack of good to touch
The fading of body and soul
Like smoke before wind
Corrupt, unsubstantial.

For I had expected always
Some brightness to hold in trust,
Some final innocence
To save from dust;
That, hanging solid,
Would dangle through all
Like the created poem
Or the dazzling crystal.

An abridged version of Stephen Spender's poem "*What I Expected*"

Bibliography

[ABGS86] D. Agrawal, A.J. Bernstein, P. Gupta, and S. Sengupta. Distributed Multi-Version Optimistic Concurrency Control for Relational Databases. In *COMPS COMPCON Spring 1986. IEEE Computer Society International Conference*, pages 416–421. IEEE, 1986.

[ABGS87] D. Agrawal, A.J. Bernstein, P. Gupta, and S. Sengupta. Distributed Optimistic Concurrency Control with Reduced Rollback. *Distributed Computing (Germany)*, 2(1):45–59, 1987.

[BLM+86] G.S. Blair, R. Lea, J.A. Mariani, J.R. Nicol, and C. Wylie. Total System Design in IPSEs. In I. Sommerville, editor, *Software Engineering Environments*, pages 85–103. Peter Peregrinus Ltd, on behalf of the IEE, 1986.

[BNY86] G.S. Blair, J.R. Nicol, and C.K. Yip. A functional model of distributed computing. Technical Report CS-DC-1-86, University of Lancaster, 1986.

[CD88] G.F. Coulouris and J. Dollimore. *Distributed Systems Concept and Design.* International Computer Science Series. Addison-Wesley, 1988.

[Cen91] National Computer Security Center. Integrity in automated information systems. Technical Report Technical Report 79-91, National Computer Security Center, September 1991.

[Coh85] F. Cohen. *Computer Viruses.* PhD thesis, University of South Carolina, 1985.

[CW87] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Proceeding of the 1987 IEEE Symposium*

on Security and Privacy, pages 184–194, Oakland, CA, USA, 27-29 April 1987. IEEE Computer Society.

[DMX91] J. Dollimore, E. Miranda, and Wang Xu. The design of a system for distributed shared objects. *The Computer Journal, Special Issue on Distributed Systems*, 34(6):514–521, December 1991.

[Don81] J.E. Donnelley. Managing domains in a network operating system. In *Proceedings of the ONLINE Conference on Local Networks and Distributed Office Systems*, pages 345–361, 1981.

[EGLT76] K.P. Eswaran, J.N. Gray, R.A. Lorie, and I.T. Traiger. The notions of consistency and predicate locks in a database operating system. *Communications of the ACM*, 19(11):624–633, November 1976.

[Gle90] T. Gleeson. *Aspects of Abstraction in Computing.* PhD thesis, Cambridge University, 1990.

[GM83] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Transactions on Database Systems*, 8(2):186–213, JUNE 1983.

[Gon89] L. Gong. On security in capability-based systems. *ACM Operating Systems Review (SIGOPS)*, 23(2), April 1989.

[Gon90] L. Gong. *Cryptographic Protocols for Distributed Systems.* PhD thesis, Cambridge University, April 1990.

[Gra81] J.N. Gray. The transaction concept: virtues and limitations. In *Proceeding of the Conference on Very Large Databases*, pages 144–154, September 1981.

[Gra85] J.N. Gray. Why do computers stop and what can be done about it? Technical Report TR 87.7, Tandem, 1985.

[Her85] M. Herlihy. Atomicity vs. availability: Concurrency control for replicated data. Technical Report CMU-CS-85-108, Carnegie-Mellon University, February 1985.

[Her87] M. Herlihy. Optimistic concurrency control for abstract data types. *ACM Operating Systems Review (SIGOPS)*, 21(2):33–44, 1987.

[Hor89] C. Horn. Is object orientation a good thing for distributed systems? In W. Schröder-Preikschat, W. Zimmer. Series Ed. G. Goos, and J. Hartmanis, editors, *Lecture Notes in Computer Science. 433. Progress in Distributed Operating Systems and Distributed Systems Management*, pages 60–74, Berlin, W.Germany, 18-19 April 1989. Springer-Verlag.

[HOS90] W.H. Harrison, H. Ossher, and P.F. Sweeney. Coodinating concurrent development. In *CSCW'90 Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 157–168, Los Angeles, CA, USA, 7-10 October 1990. ACM. Sponsored by ACM SIGCHI and SIGOIS.

[HW86] E. Horowitz and R.C. Williamson. SODOS: A Software Documentation Support Environment - its definition. *IEEE Transactions on Software Engineering*, SE-12(8):849–859, August 1986.

[JM86] D. Jefferson and A. Motro. The Time Warp Mechanism for Database Concurrency Control. *IEEE*, 6(86):474–480, 1986.

[Jon78] A.K. Jones. The object model: A conceptual tool for structuring software. In *Operating Systems: An Advanced Course*, volume 60, pages 7–16. Springer-Verlag, 1978.

[Kar88] P.A. Karger. *Improving Security and Performance for Capability Systems.* PhD thesis, Cambridge University, October 1988.

[KH84] P.A. Karger and A.J. Herbert. An augmented capability architecture to support lattice security and traceability of access. In *Proceeding of Symposium on Security and Privacy*, April 1984.

[KR81] H.T. Kung and J.T. Robinson. On optimistic concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, 1981.

[Lam73] B.W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):416–421, 1973.

[LFK88] M. Leland, M. Fish, and R.E. Kraut. Collaborative document production using quilt. In *Proceedings of CSCW'88 Conference on Computer-Supported Cooperative Working*, pages 206–215, 1988.

[Lin87] M.A. Linton. Distributed management of a software database. *IEEE Software, Special Issue on Integrated Environments*, 4:70–76, November 1987.

[LLS90] R. Ladin, B. Liskov, and L. Shrira. Lazy replication: Exploiting the semantics of distributed services. In *Proc. 1st Workshop on Management of Replicated Data*, pages 35–38, November 1990.

[Low92] M.R. Low. Fine Grained Protection in UNIX. Technical Report 130, School of Information Sciences, The Hatfield Polytechnic, March 1992.

[Mar91] S. Marsh. The V Project Manager Tools. *ACM SIGSOFT Software Engineering Notes*, 16(2):58–61, April 1991.

[MT84] S.J. Mullender and A.S. Tanenbaum. Protection and resource control in distributed operating systems. In ??, editor, *Computer Networks 8*, pages 421–432. Elsevier Science (North Holland), 1984.

[Mul85] S.J. Mullender. *Principles of Distributed Operating System Design.* PhD thesis, Vrije Universiteit, Amsterdam, October 1985.

[Nee90] R.M. Needham. *Capabilities and Security.* Workshop in Computing, Series Ed. Professor C.J. van Rijsbergen, Edition Ed. J. Rosenberg and J.L. Keedy. Springer-Verlag (in collaboration with the BCS), Bremen, W.Germany, 8-11 May 1990.

[NHSS87] D. Notkin, N. Hutchinson, J. Sanislo, and M. Schwartz. Heterogeneous computing environments: Report on the ACM SIGOPS Workshop on Accomodating Heterogeneity. *Communications of the ACM*, 30(2), February 1987.

[NKCM90] C.M. Neuwirth, D.S. Kaufer, R. Chandhok, and J.H. Morris. Issues in the design of computer support for co-authoring and commenting. In

CSCW'90 Proceedings of the Conference on Computer-Supported Cooperative Work, pages 183–195, Los Angeles, CA, USA, 7-10 October 1990. ACM. Sponsored by ACM SIGCHI and SIGOIS.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[NS87] R.M. Needham and M.D. Schroeder. Authentication revisited. *ACM SIGOPS*, 21(1), January 1987.

[Oli90] R. Oliver. Protection in a distributed document processing system. *ACM Operating Systems Review (SIGOPS)*, 24(2):56–66, April 1990.

[OR87] D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review (SIGOPS)*, 21(1), January 1987.

[PEea90] P.E.English, E.Jacobson, and R.A. Morris et al. An extensible, object-oriented system for active documents. In R. Furuta, editor, *Proc. of the Int. Conf. on Electronic Publishing, Document Manipulation & Typography. Cambridge Series on Electronic Publishing*, pages 263–276, National Institute of Standards and Technology, Gaitherburg, Maryland, USA, 18-20 September 1990. Cambridge University Press.

[PS83] J. Peterson and A. Silberschatz. *Operating System Concepts.* Addison Wesley, 1983.

[QNA90] V. Quint, M. Nanard, and J. André. Towards document engineering. In R. Furuta, editor, *Proc. of the Int. Conf. on Electronic Publishing, Document Manipulation & Typography. Cambridge Series on Electronic Publishing*, pages 17–30, National Institute of Standards and Technology, Gaitherburg, Maryland, USA, 18-20 September 1990. Cambridge University Press.

[RBG91] M. Reiter, K. Birman, and L. Gong. Integrated security in a group oriented distributed system. Technical Report TR 91-1239, Department of Computer Science, Cornell University, Ithaca, New York, October 1991.

[Ree83] D.P. Reed. Implementing Atomic Transactions on Decentralising Data. *ACM Transactions on Computer Systems*, 1(1):3–23, 1983.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Smi92] G.W. Smith. Modeling security-relevant data semantics. *IEEE Transactions on Software Engineering*, 17(11):1195–1203, November 1992.

[Sno90] J.F. Snook. Toward secure optimistic, distributed, open systems (a draft accompanying transfer report). Technical report, Hatfield Polytechnic, 1990.

[TEH+89] V. Tschammer, K.P. Eckert, J. Hall, G. Schürmann, and L. Strick. OAI - Concepts for Open Systems Cooperation. In W. Schröder-Preikschat, W. Zimmer. Series Ed. G. Goos, and J. Hartmanis, editors, *Lecture Notes in Computer Science. 433. Progress in Distributed Operating Systems and Distributed Systems Management*, pages 174–191, Berlin, W.Germany, 18-19 April 1989. Springer-Verlag.

[TKS88] E.F. Troy, S.W. Katzhe, and D.D. Steinaner. Technical solutions to the computer security intrusion problem. In F.L. Hubard and R.D. Shelton, editors, *Protection of Computer Systems and Software*, pages 179–239. 1988.

[WBHN87] J. Walpole, G.S. Blair, D. Hutchison, and J.R. Nicol. Transaction mechanisms for distributed programming environments. *IEE Software Engineering Journal*, 2(5):169–177, September 1987.

[WBMN88] J. Walpole, G.S. Blair, J. Malik, and J.R. Nicol. Maintaining consistency in distributed software engineering environments. In *Proceedings of 8th International Conference on Distributed Processing Systems*, pages 418–425. IEEE, June 1988.