

SYSTEMS SUPPORT FOR DISTRIBUTED LEARNING
ENVIRONMENTS

Colin Allison

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



2003

Full metadata for this item is available in
St Andrews Research Repository
at:

<http://research-repository.st-andrews.ac.uk/>

Please use this identifier to cite or link to this item:

<http://hdl.handle.net/10023/14519>

This item is protected by original copyright

Systems Support for Distributed Learning Environments



A thesis submitted to the
University of St Andrews
for the degree of
Doctor of Philosophy

by
Colin Allison

School of Computer Science,
University of St Andrews
June 2003



ProQuest Number: 10166799

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10166799

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

TH E520

Abstract

This thesis contends that the growing phenomena of multi-user networked “learning environments” should be treated as distributed interactive systems and that their developers should be aware of the systems and networks issues involved in their construction and maintenance. Such environments are henceforth referred to as *distributed learning environments*, or DLEs. Three major themes are identified as part of systems support: i) shared resource coherence in DLEs; ii) Quality of Service for the end-users of DLEs; and iii) the need for an integrating framework to develop, deploy and manage DLEs.

The thesis reports on several distinct implementations and investigations that are each linked by one or more of those themes. Initially, *responsiveness* and *coherence* emerged as potentially conflicting requirements, and although a system was built that successfully resolved this conflict it proved difficult to move from the “clean room” conditions of a research project into a real world learning context. Accordingly, subsequent systems adopted a web-based approach to aid deployment in realistic settings. Indeed, production versions of these systems have been used extensively in credit-bearing modules in several Scottish Universities.

Interactive responsiveness then emerged as a major Quality of Service issue in its own right, and motivated a series of investigations into the sources of delay, as experienced by end users of web-oriented distributed learning environments. Investigations into this issue provided insight into the nature of web-oriented interactive distributed learning and highlighted the need to be QoS-aware.

As the volume and the range of usage of distributed learning applications increased the need for an integrating framework emerged. This required identifying and supporting a wide variety of educational resource types and also the key roles occupied by users of the system, such as tutors, students, supervisors, service providers, administrators, examiners.

The thesis reports on the approaches taken and lessons learned from researching, designing and implementing systems which support distributed learning. As such, it constitutes a documented body of work that can inform the future design and deployment of distributed learning environments.

Declarations

- I, **Colin Allison**, hereby certify that this thesis, which is approximately 74,000 words in length, has been written by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree.

date 25/6/2003... signature of candidate

- I was admitted as a research student in 1991 and as a candidate for the degree of Ph.D in 1991; the higher study for which this is a record was carried out in the University of St Andrews between 1994 and 2002.

date 25/6/2003 signature of candidate |

- I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of ...*Ph.D*... in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

date 25/6/2003 signature of supervisor ..

- In submitting this thesis to the University of St Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker.

date*25/6/03*... signature of candidate ..

Acknowledgements

The primary thanks for the completion of this part-time Ph.D. thesis must go to my immediate family, Rosa and Martin, without whose extensive support and tolerance this thesis would never have been finished. I must also thank my parents, Chris and John, who, in the Scottish tradition, instilled in me a belief in the value of learning and educational attainment. I also thank many colleagues and friends who have been supportive of the work behind this thesis: Mike Livesey for his insights into virtual time; Alan Ruddle for insight into TCP; Paul Harrington and Feng Huang for their enthusiasm during the Warp project; David Power, Christine Helliard, Donald Sinclair and Rosa Michaelson at the Department of Accounting and Business Finance, University of Dundee, for their close co-operation on Finesse; all the other participants in the TAGS pilots at various Scottish institutions; Alan Ruddle, David McKechnan, Jose Serrano, Martin Bramley, Ross Nicholl and Martin Bateman for contributions towards the evolution of TAGS as both a production service and an experimental framework; Pete Lindsay for suggestions on protocol visualisation; Alan Dearle for finding time for supervision while having other responsibilities; Ron Morrison for having read and helped with the final version; the Postgraduate Committee and the School of Computer Science at the University of St Andrews for their understanding that theses being completed on a part-time basis can take a little longer than originally expected.

Colin Allison

June 2003

Systems Support for Distributed Learning Environments

Contents

1. Introduction

1.1	Thesis Overview.....	1.1
1.2	Distributed Learning Environments	1.1
1.3	The Genesis of Distributed Learning Environments.....	1.4
1.4	Pedagogical Goals and their Technical Implications	1.5
1.5	Users, Roles and Requirements.....	1.8
1.5.1	Learner Requirements	1.8
1.5.2	Academic Requirements.....	1.9
1.5.3	Course Co-ordinator Requirements	1.9
1.5.4	System Administrator Requirements	1.10
1.5.5	Developer Requirements	1.10
1.5.6	Service Provider Requirements.....	1.10
1.5.7	User-Centric Views	1.11
1.6	A Layered Reference Model	1.13
1.6.1	Quality of Service.....	1.14
1.6.2	DLE Middleware	1.15
1.7	Summary.....	1.17

2. Systems concepts used in addressing DLE requirements

2.1	QoS, Computer Networks and the Internet.....	2.3
2.1.1	Meeting QoS Requirements: Asynchronous Transfer Mode (ATM)	2.6
2.1.2	The Internet Protocols	2.8
2.1.3	Patterns of Network Transmission and Communication	2.13
2.2	Coherence, Distributed Systems and Event Ordering	2.17
2.1.1	Passive and Proactive Applications of Distributed Systems Theory	2.19
2.1.2	Lamport's Clock Algorithm: A Proactive Approach	2.20
2.1.3	Group Communication	2.21
2.1.4	Distributed Systems and the Internet	2.26

2.3	Groupware.....	2.27
2.3.1	Groupware Environments	2.29
2.3.2	Sharing, Concurrency and Coherence	2.31
2.3.3	Groupware Environments and QoS	2.34
2.3.4	Security	2.36
2.4	Summary.....	2.42

3 Related Work

3.1	Background.....	3.1
3.1.1	BSCW: Web-based groupware	3.3
3.1.2	PIPVIC: IP Multicast for Educational Applications	3.4
3.1.3	Internet2: QoS	3.4
3.1.4	Internet2: Security Infrastructure	3.5
3.2	dOPT: Coherence in Groupware Systems	3.5
3.2.1	The dOPT groupware model	3.6
3.2.2	Interpreting dOPT	3.10
3.2.3	Discussion: Serialisability in Groupware	3.11
3.3	TimeWarp: Coherence in Distributed Systems.....	3.14
3.3.1	Virtual Time	3.15
3.3.2	Local Virtual Time and Rollback	3.15
3.3.3	Global Virtual Time and Commitment	3.17
3.4	BSCW: A Framework for Web-Based Groupware Environments	3.18
3.4.1	BSCW in Educational Contexts	3.19
3.4.2	Discussion: Web-based Groupware	3.20
3.5	PIPVIC: IP Multicast Groupware in Educational Contexts.....	3.21
3.5.1	The Mbone Tools as Groupware	3.26
3.6	Internet2: A New Infrastructure for Education	3.28
3.6.1	Internet2 and Quality of Service: The Qbone	3.29
3.6.2	Distributed Access Control: Shibboleth	3.30
3.7	Summary.....	3.32

4. Addressing the Coherence Requirement

4.1. Explicit Virtual Time Based Coherence	4.1
4.1.1. Twarp Process Model	4.2
4.1.2. Operational Communications Model.....	4.4
4.1.3. Node Configuration	4.6
4.1.4. Global Virtual Time	4.7
4.1.5. Persistent Object Storage and State	4.11
4.1.6. Lessons Learned from Twarp.....	4.14
4.2. WARP: A Coherence Mechanism for Distributed Groupware.....	4.15
4.2.1. System Overview.....	4.16
4.2.2. Objects	4.18
4.2.3. Atoms	4.19
4.2.4. The Differences between Atoms and Transactions.....	4.19
4.2.5. Resolving Conflict.....	4.21
4.2.6. The Warp Runtime Library	4.29
4.3. A Groupware Case Study: The Warp-based Shared Spreadsheet.....	4.33
4.3.1. Group awareness features.....	4.34
4.3.2. The shared context.....	4.36
4.3.3. WYSIWIS issues in the shared spreadsheet.....	4.37
4.3.4. Editing Sessions.....	4.38
4.3.5. Quality of Service Issues in the Spreadsheet.....	4.38
4.3.6. The Shared Spreadsheet in an Educational Context.....	4.39
4.3.7. Lessons Learned	4.41

5 Addressing the Framework Requirement

5.1 Finesse: Finance Education in a Scalable Software Environment	5.3
5.1.1 Origins of Finesse	5.4
5.1.2 The Finesse Portfolio Management Facility (PMF)	5.5
5.1.3 Finesse Application Overview	5.7
5.1.4 The Finesse Development Process	5.12
5.1.5 Evaluation of Finesse	5.16
5.1.6 Abstracting Generic DLE Features from Finesse	5.20

5.2	The TAGS Framework for DLEs.....	5.21
5.2.1	Users, Roles and Groups	5.22
5.2.2	Events	5.24
5.2.3	Support for Developers	5.26
5.2.4	Resources	5.28
5.2.5	Support for Service Providers	5.33
5.3	Case Study: Group-Based Management of Distributed IT Work Placements....	5.39
5.3.1	The Document Approval Tool (DAT)	5.41
5.3.2	Evaluation of the DAT	5.43
5.3.3	Evolution of the DAT	5.44
5.4	Summary	5.47
6	Understanding the Quality Of Service Requirements for Interactive Responsiveness	
6.1	Interactive Responsiveness in Finesse.....	6.2
6.1.1	Basic Web Operations	6.4
6.1.2	Proxies and caching	6.5
6.1.3	A Structured Timing Model	6.5
6.1.4	Timing Issues	6.8
6.1.5	Measurements	6.10
6.1.6	Analysis and Solutions	6.12
6.2	A Revised Model of Web Delay Analysis.....	6.13
6.2.1	Client Limitation	6.15
6.2.2	Network Limitation	6.16
6.1.3	Protocol Limitation	6.19
6.1.4	Server Limitation	6.24
6.2	Applying the revised model to TAGS	6.26
6.1.1	Client Limitation	6.27
6.1.2	Server Limitation	6.28
6.1.3	Network Limitation	6.29
6.1.4	Comparison of Client, Network, Protocol and Server Limitations	6.29
6.3	Summary: Being QoS Aware	6.30

7 Addressing the Qos Requirements for Interactive Resources

7.1	Interactive Responsiveness – A Multiple Server Approach	7.3
7.1.1	The Case for Replication	7.6
7.1.2	A Replicated Resource Architecture	7.7
7.1.3	Types of Replication and Coherence	7.9
7.1.4	Replication Strategy	7.16
7.1.5	Coherence Strategy	7.17
7.1.6	An analytical model of coherence delay	7.22
7.1.7	Case study – The TAGS Notebook Resource	7.24
7.2	Interactive Continuous Media – Adaptability and Integration	7.28
7.2.1	Adaptation Strategy	7.30
7.2.2	Integration Strategy	7.37
7.2.3	Case Study: Small Group Collaboration	7.40
7.2.4	A Scenario Driven Testbed for Conference Resource Development	7.42
7.3	Summary	7.43

8 Conclusion

8.1	Distributed Learning Environments	8.1
8.2	Coherence	8.2
8.2.1	A Shared Spreadsheet	8.3
8.3	QoS	8.3
8.3.1	Understanding Delay in Server-based Resources	8.4
8.3.2	Addressing Server Limitations	8.6
8.3.3	Addressing Timeliness	8.7
8.3.4	Integration of QoS-Oriented Support Features	8.8
8.4	The Framework	8.8
8.4.1	Domains	8.11
8.5	Future Work	8.12
8.5.1	Automatic Group Maintenance	8.12
8.5.2	Automating QoS-Oriented Runtime Services	8.15
8.6	Concluding Remarks	8.16

Systems Support for Distributed Learning Environments: Introduction

1.1 Thesis Overview

This thesis reflects how trails of interests, investigations and implementations gradually converged into the unifying topic of system support for distributed learning. This introductory chapter explains the concept of a “distributed learning environment”, why it is of significance and worthy of research, and what it requires in terms of systems and network support. The second chapter identifies and describes particular systems and networks concepts used to address the needs of such environments. These include specific concerns within the wider categories of computer networks, distributed systems and groupware systems that are applicable to distributed learning environments. Chapter 3 describes and critically evaluates some other research projects that are related to the systems issues identified in Chapters 1 and 2. These issues form the focus of the investigations described in the following four chapters:

Chapter 4: the use of virtual time to provide coherence in distributed groupware

Chapter 5: a framework for the development, deployment and management of web-based groupware in an educational context

Chapter 6: identifying, modeling and measuring the sources of delay in interactive responsiveness, a key Quality of Service requirement of distributed learning environments

Chapter 7: the design of architectures and prototyping of systems that support interactive responsiveness requirements, and timeliness requirements for video conferencing

Finally, Chapter 8 concludes this thesis.

1.2 Distributed Learning Environments

Distributed Learning Environments (DLEs), built from communications and information technology, are being cited as solutions to the ambitious political goals of better

education, wider access and lower costs in the education and training sector. The most recent major report on the future of higher education in the UK¹ stated:

“Advances in communications and IT (C&IT) will radically alter the shape and delivery of learning throughout the world. Over the next decade ...UK institutions will rely heavily on C&IT to teach quality, flexibility and effectiveness of higher education. The potential benefits will extend to, and affect the practice of, learning and teaching and research. C&IT will have a central role in maintaining the quality of higher education in an era when there are likely to be continuing pressures on costs and a need to respond to an increasing demand for places in institutions. C&IT will overcome barriers to higher education, providing improved access and increased effectiveness, particularly in terms of lifelong learning.” (Dearing 1997)

Reciprocally, with so much political emphasis being placed on DLEs, there is an emergent software industry in their production, and even speculation that they could be the next “killer app”(Finlay 1999). Surveys and initiatives concerning the use of information systems in the education sector now routinely refer to entities such as “virtual learning environments” (VLEs) as though they are some form of panacea for under-resourcing. Many of the current commercial VLEs, such as WebCT (WebCT: 1999) and Blackboard(Blackboard 2000), are seen as a way forward by policy makers who take Dearing (and similar reports) at face value without any great understanding of the technological or pedagogical issues involved. Indeed, these systems offer little more than a means of uploading lecture notes and other course materials to a web site where students can find them and read or download them. Many academics who are not computer literate use these systems as a user-friendly interface to the Internet protocols for file transfer (ftp) and hypertext transfer (http). At a slightly more adventurous level these systems may be used as web-based shared workspaces, as epitomised by BSCW (Basic Support for Co-operative Work on the Web)(Bentley et al. 1997), an early web-

¹ The Report of the National Committee of Inquiry into Higher Education (1997), chaired by Sir Ron Dearing, was charged with making recommendations for Higher Education for the next twenty years.

oriented, document-centric, groupware system, which is described in Chapter 3 under Related Work.

The problem is that the adoption of what is easy to accomplish on the web is then labeled as "e-learning" without any reference to pedagogy, and with little or no understanding of the underlying technology. An increasingly voiced criticism of this use of the web is that the role of the teacher is reduced to that of a selector of material (information) for learners to look at, and that learning is then confused with information transfer. Similarly, the heavy use of multiple choice questions for web-based assessment reflects the use of what is technically convenient rather than what is pedagogically effective.

On the technical side these e-learning platforms are distributed systems in the simple sense that any client-server web interaction is distributed. As there is little attention paid to any networking or systems issues in these platforms it is not uncommon for them to collapse under load, misbehave when used in some unexpected manner, be unavailable at critical times, and be unable to accommodate innovative or effective pedagogies. For example, the shared spreadsheet described in Chapter 4, and the Portfolio Management Facility that uses real-time stock market data that is described in Chapter 5 could not be accommodated by those platforms.

Although it may seem obvious that the successful deployment of DLEs depends on the use they make of the network and systems infrastructure upon which they are built, online learning packages such as WebCT, Blackboard and others that are compared in (Britain & Liber 1999) ignore such issues. Indeed, there seems little concern from such packages, and their evaluators, for quality of service (QoS) issues of any kind.

This thesis contends that DLEs should be treated as QoS-aware distributed interactive systems, and documents the construction, deployment and evaluation of DLEs according to this principle. The benefit is that DLEs become more useable and effective, and better able to stimulate and support innovative and effective pedagogical practice.

This approach necessitates the use and adaptation of systems concepts and mechanisms drawn from the fields of computer networks, distributed systems and groupware systems.

1.3 The Genesis of Distributed Learning Environments

What is meant by a DLE in this thesis? The term is intended to capture the use of network-based distributed application environments that are used directly by learners and teachers for educational purposes. The ubiquity of the Internet means that any educational institution can now exploit and share its teaching expertise and educational resources through online access programmes – this type of activity is no longer the preserve of specialist institutions. Indeed, if access to quality education is to be widened, it is imperative that all institutions play their part. The routine use of DLEs creates requirements for systems support and a need for appropriate software architectures.

DLEs differ in several important respects from earlier computer-based and distance learning scenarios. Online environments are intrinsically multi-user – where a student may previously have worked in isolation on a computer, DLEs have the potential to:

- Support teamwork, through various forms of conferencing and shared tasks.
- Introduce real world input into the study environment. For example: finance students studying fund management can work in groups to run a virtual portfolio of shares whose value is updated in real time by data from the London Stock Exchange; video streams of real-time surgery can be transmitted live to the desktop; meteorology students can access satellite weather data in real time and compare their predictions with those of the local news programs.

Although DLEs have opened up numerous new possibilities for core teaching and learning activities, their impact is significantly wider. In the same way that the application of digital computing has changed the way that scientific programmes are envisaged and carried out, the nature of the entire educational process is also changing. The use of distributed systems in education is affecting not only the content and format of learning resources but also a multiplicity of management tasks such as assignment tracking, marking and grading, student record maintenance and cohort analysis. When a learner is working in a networked environment it is possible to monitor and analyse the

use they make of the system and feed this information back into the educational process, on an individual basis, thereby supporting student-centred learning. The use of a distributed client-server structure in DLEs also means that the QoS being delivered to a user can be monitored and measured. Analysis of this type of information can inform the future development of a DLE's content, architecture and selection of infrastructure options.

Some exponents of DLEs see them as replacing social learning and sidelining the lecturer and campus experience. There is nothing inherent in the concept of a DLE that supports these views. On the contrary, a DLE can reinforce the uniqueness and quality of education in different institutional contexts. It is noteworthy that MIT has provided free public access to many of its online teaching resources (Long 2002), because it believes that an effective education requires the experience that can only be gained by attending an institution in person, that is, *social learning*.

So, it is important to state that opting to use DLEs does not imply the replacement of educationally stimulating experience by "distance" or "virtual" learning. It follows that the process of DLE construction must be flexible and accommodating, and not prescribe or restrict the uses of IT in education. Accordingly, this thesis adopts the concept of a *framework* approach to facilitate the research, development, deployment and management of DLEs. This approach provides flexibility by separating the generic requirements of DLEs from the specifics for any particular locale or subject-specific learning resource types. That framework is grounded in the established principles of interactive distributed groupware systems.

1.4 Pedagogical Goals and their Technical Implications

From the perspective of good educational practice the following features should be supported in a DLE:

- group working should be routinely supported as well as the more traditional mode of the solitary learner
- learning resources should be interactive, engaging, and responsive

- real-world input should be easy to incorporate, as should simulations
- students should find themselves at the centre of their online environment, with their individual needs addressed
- wider, more flexible access to education should be provided, often referred to as “anytime/anywhere” learning

Those pedagogical goals imply certain technical requirements.

- *Support for group working* means that online resources may be shared and used by multiple concurrent readers and writers. This requires concurrency control, and raises the usual associated concerns with liveness, safety and fairness. If two or more members of a group are modifying the state of a shared resource, at the same time, it is important that the resource is not left in an unintended state. If the shared resource is distributed, replicated or cached, it is important that different users do not end up with different versions, while believing them to be accurate copies of a single authoritative version. In addition, multi-user awareness is important for teamwork. Members of a group need to know what others are doing, and have appropriate communication options to co-ordinate with each other. For example, text-based, audio or video conferencing may be used to augment shared resource interaction.
- *Interactivity* means that online working is much more than simply browsing or downloading lecture notes that have been placed on the Web. Of course, interactivity is essential for any form of learning. If a student goes into a library and does not open any books, they will learn nothing. The issue here is that if they do open a book and are faced with a page displaying the message “waiting on network host for next paragraph” they are unlikely to be motivated to use that form of learning technology. Furthermore, interactivity in this context, unlike a library book, refers to the ability of the user to explicitly change the state of a learning resource instance.
- *Responsiveness* is essential for creating an interactive feel when working across the network. It means that the delay between a user making a request and a result being

returned to them should be no longer than some period that is considered reasonable within a particular context. (A user working from home over a relatively low bandwidth modem line will have lower expectations than one who is attached to a high-speed campus network). If a network service is perceived to be “slow” then it will be seen as an unproductive use of time, and may be abandoned.

- *Real world input* is facilitated by direct Internet connection to dynamic data sets. For example, the Finesse portfolio management facility (Power et al. 1998) allows the management of a portfolio of shares by a student group. The groups maintain their portfolios by buying and selling shares chosen from a database of live values for companies that are quoted on the London Stock Exchange.
- *Student-centred* means understanding users as individuals and implies strong monitoring capabilities, be they for manual inspection, automatic adaptation, or reflective feedback for learners. In an interactive online environment it is of course possible to implement extensive activity logging, which can then be “mined” for useful feedback on a student’s use of a set of educational resources. This feedback can be used to inform the resource development and adaptation process and thereby enhance the learning environment.
- *Anytime/anywhere* implies a resilient, highly available network service. Availability means coping with faults or slow downs in responsiveness. Providing a service based on multiple servers can negate periods of non-availability caused by server crashes and peak loads. These servers may be distributed across a wide area, possibly globally. If the same set of servers is also used to share the load then *replication* becomes an attractive strategy as it can, in principle, also support performance by providing an appropriate ratio of users to servers. This approach is further useful as a practical means of implementing *incremental scalability*, which may be necessary to cope with a dynamically changing user population.

1.5 Users, Roles and Requirements

What *types* of user does a DLE have? End-users typically fall into one of three general roles:

- learner
- academic
- systems-related

Within these general roles it is possible to list examples of specific roles and responsibilities, as shown in Table 1.1. A user may of course have more than one role, especially if they are academic staff. It is also possible for a staff member to be a student.

<i>General Roles</i>	Learner	Academic	Systems-related
<i>specific role examples</i>	Science student Arts student Postgraduate Student Part-time Student Member of tutorial group Member of module class Member of degree programme cohort	Tutor Lecturer Demonstrator Course Co-ordinator External Examiner Head of School Director of Teaching Subject-specialist	Systems Administrator Service Provider Software developer

Table 1.1: Some of the roles taken by end-users of a DLE

1.5.1 Learner Requirements

Learners are usually not computing specialists. They are increasingly likely to be computer literate in the sense of being able to use a web browser, e-mail, and applications such as a word processor. It is therefore important that an online environment is accessible in the sense of being usable. The W3C guidelines (W3C 2002) are often used as a reference point for checking on the accessibility of a web site. These guidelines include provision for students with special needs. Pedagogical requirements must be included: group activities require group communication and awareness facilities; interactivity should be of the sort that requires the student to engage with the environment and be able to change its state, rather than simply clicking to select pre-

defined options; real-world input should be incorporated where appropriate; the DLE should be available anytime, from anywhere. Students also frequently wish to know how they are doing, relative to other members of their tutorial group or module cohort. Feedback such as appropriate class averages can be generated automatically and made accessible to them. Further types of reflective feedback may include attendance and performance records for a series of lectures or tutorials.

A student may also need to attend a tutorial remotely, using video conferencing for example. In this case the conference resource should be able to adapt to the nature of the remote network connection.

1.5.2 Academic Requirements

Academics may be computer literate, but are rarely computing specialists with an understanding of operating systems. Accordingly, a DLE must be able to provide useful abstractions for them to work with. These are likely to include roles such as *student*, *tutor*, and *course co-ordinator*, and concepts such as *group*, *course*, *module*, *class* and *assignment*. At the same time, lecturers require freedom and flexibility to create their own operational environment, so an approach that is too prescriptive is likely to prove unpopular and counter-productive. The provision of simple abstractions that can be used as building blocks is therefore attractive. Particular constructions can be saved as templates consisting of certain components and the relations between them. These can form the basis of generic models which can be used by those academics that are not directly involved in the DLE design and construction process to focus their requirements and evaluate the capabilities of a particular type of DLE, prior to choice and commitment.

1.5.3 Course Co-ordinator Requirements

This is a role allocated to some academics which requires an overview of all users and resources associated with a module or course. Whereas a tutor may be concerned with individual student progress a co-ordinator may be more concerned with the returning of marks to students on time, student attendance at lectures, tutorials and practical sessions, the collation of marks and the issuance of warnings (to both staff and students) when

due dates have been missed. It follows that a co-ordinator's role requires a different view and set of capabilities than those allocated to a tutor.

1.5.4 System Administrator Requirements

While a DLE should empower academics as much as is possible there will inevitably be tasks in its creation and maintenance that require computing expertise. These may include operating systems and web-server configuration, security measures, and server configuration. A system administrator is likely to need a global view of a DLE, and have special privileges, in order to see the problems that are reported, and then readily fix them.

1.5.5 Developer Requirements

There should be no restriction on the nature of an educational resource. It may be a simple timetable, an automated assessment exercise or an interactive multi-user simulation. In contrast to learners and tutors, developers are computing specialists, in that they develop and maintain resources. It is productive to let people play to their strengths, and in the case of developers this means allowing them to focus on the objectives required by subject-specialists without worrying about the implementation of generic facilities for the deployment, distribution, access control, and other management tasks associated with a learning resource. A DLE should provide a useful set of generic services and interfaces for developers. For example, generic models for distribution allow a developer to choose the conditions under which their resource can be replicated, made available, copied, re-used, and accounted for.

1.5.6 Service Provider Requirements

A service provider requires a framework that is compatible with prevalent ICT, rather than one that will only function effectively in laboratory conditions. So, being able to dovetail with commodity technologies such as relational databases, Internet protocols, web servers, and PC hardware clusters is an important consideration.

Responsiveness and availability should be supported. Supporting responsiveness means ensuring that the network connection is of adequate bandwidth and latency and that the server(s) can respond in adequate time, even under peak loads. If there is a high variance in loads and a dynamically growing user base *scalability* becomes a major issue. Initially provisioning a monolithic system of sufficient capability to meet peak demands is not a good solution as small changes in the scheduling of content or the size of the user base can quickly invalidate predictions about peaks. Availability means coping with failures. Failures can occur in either the network or the server. A scheme to maintain responsiveness and availability under varying loads is therefore highly desirable.

1.5.7 User-Centric Views

All users have a common usability requirement, namely a personalised, custom view of a DLE. This avoids information overload and provides access to relevant resources. Such an interface may be referred to as a user-centric portal. In other words, the framework should provide facilities for maintaining knowledge about individuals' roles and associating it with their online identities. These relationships can then be used to dynamically generate home pages. The framework can also provide further presentational options for individuals.

In Fig. 1.1 each user has their own view of the DLE. This simple example scenario features five users: three students, an academic staff member, and a systems administrator.

- Alice is only taking a language module and only sees a set of interactive language exercises.
- Bob is taking Languages and Maths. He sees the same language exercises as Alice, a Geometry tool, and an assignment management tool for handing in work, and receiving marks.
- Carol is only taking Maths. She sees the Geometry Tool, plus her own part of the assignment manager.

- Dan is both a Course Co-ordinator and a Maths Tutor. He takes Tutorial Group Tut-1. He sees the Geometry Tool, a view of assignments limited to his own tutorial group, and also a view of the whole class, useful for a course co-ordinator.
- Emma is a systems administrator, and can, in principle see anything in the DLE. This does not preclude a resource type from providing its own privacy mechanisms.

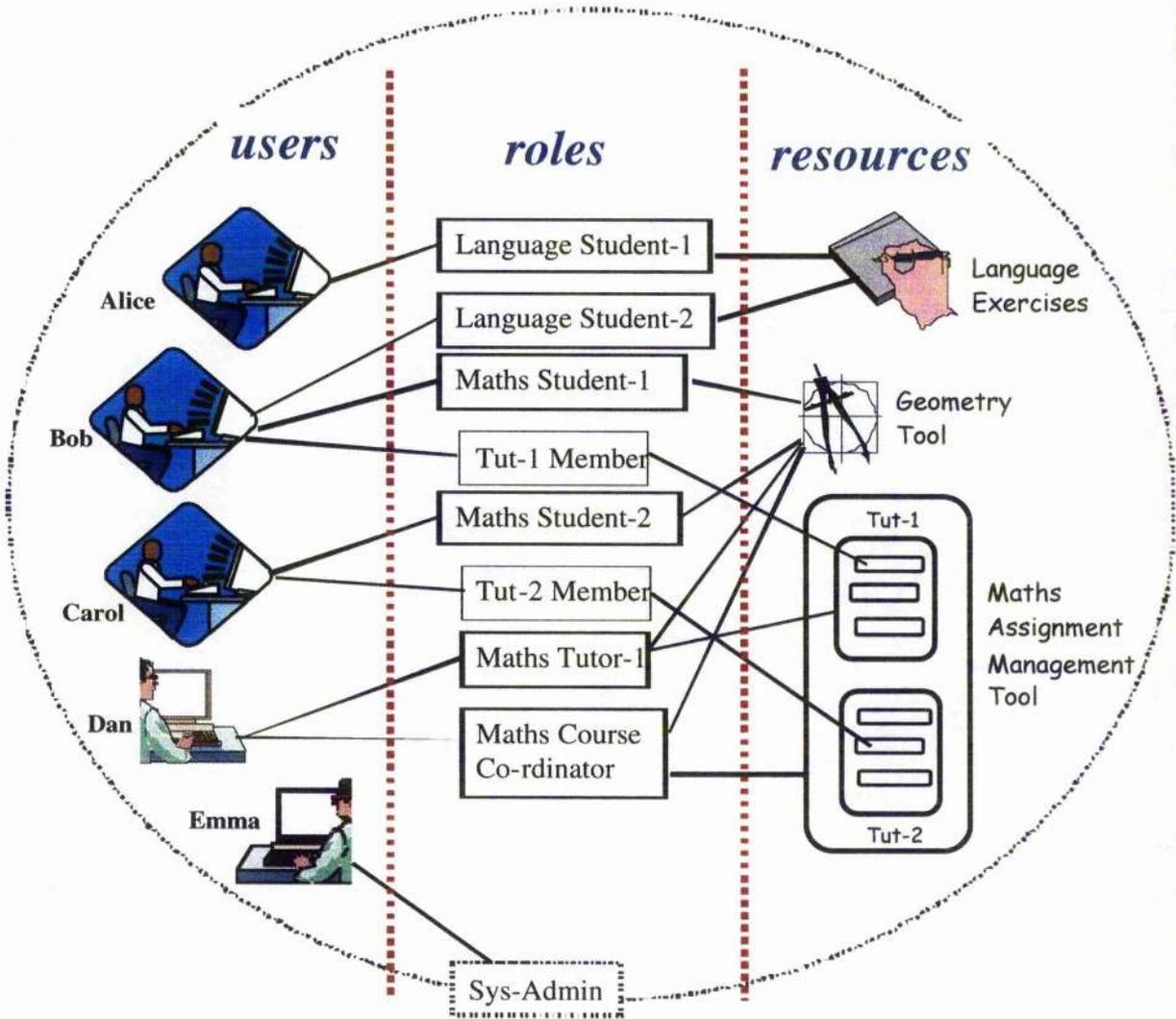


Figure 1.1: User-centric views of a DLE

The customised view delivered to each user in Fig.1.1 is based on their roles, and the qualification of the relationships between these roles and resources, based on the privileges associated with each role, and the user's identity. One of the systems support

issues for DLEs is to allow for this flexibility while maintaining the integrity of the underlying shared data structures, and the security of views – it would be inappropriate for Bob to be able to change the mark allocated by his tutor, or for Carol to see other student’s assignment submissions.

1.6 A Layered Reference Model

Figure 1.2 illustrates a layered approach to identifying the user-oriented features that a DLE should provide, and the systems concerns involved in supporting and realising these features. These concerns are mostly found in the middle layer. That is where the pedagogically driven abstractions at the upper layer must be realised using a (largely given) general-purpose infrastructure such as the Internet, and where the usability of a DLE from a customised user’s perspective should be understood and supported.

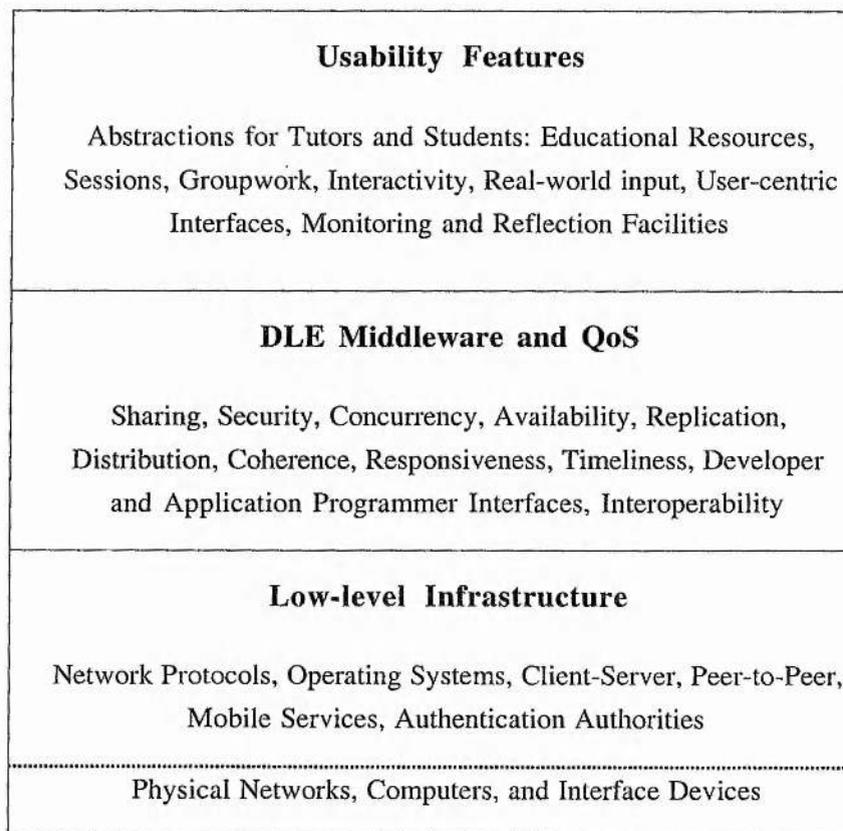


Figure 1.2: A DLE Reference Model

It is unlikely that a DLE framework would seek to develop new low-level infrastructure components, as deployment is an issue, but such work is certainly not precluded if a

pedagogical concern cannot not be addressed in any other way. The general approach taken is to work with existing hardware, operating system, networking and security technologies. The middle layer is characterised by two concepts – Quality of Service and Middleware.

1.6.1 Quality of Service

The term “Quality of Service” is most closely associated with computer networking, but for QoS to be meaningful to *users* of a *service* it should be viewed as a set of systemic properties resulting from the *combined behaviour* of all the intermediate components between end-users and a service. Support for this holistic view of QoS, can also be found in the ISO Open Distributed Processing program, where it is loosely defined as “... a system or object property, that consists of a set of quality requirements on the collective behaviour of one or more objects”(ISO/IEC 1996). More particularly "the role of QoS management is to ensure that applications are able to specify and obtain the quality of service they require for their correct execution" (Coulson & de MEER 1997). So, the main issue is what we are prepared to accept as the correct execution of a DLE.

In the example scenario shown in Fig.1.1 we could list QoS requirements as including security of views and access privileges; authentication of identity; integrity of shared resources in the face of concurrent updates; and responsiveness. If any of these requirements is not met, then we could say that the DLE is not executing correctly.

Certain QoS parameters are outwith the direct control of the DLE – the lower layer in Fig.1.2 for example. It is maintained in this thesis however, that a DLE must be *QoS-aware*. So, even if it cannot dynamically obtain more network bandwidth, or magically replace a client’s operating system with a different one, it can at least pinpoint the source of the problem with a view to reporting and adapting. If the Geometry Tool in Fig.1.1 should render a new image, on the users display, in response to user input in less than one second, then this performance target should be measurable.

1.6.2 DLE Middleware

Middleware in general consists of augmenting a given infrastructure with an extra layer of software that adds value in some useful way, and presenting appropriate programming interfaces and abstractions. The functionality provided by operating systems tends to be basic and general, reflecting their role in enhancing the value of an underlying collection of hardware and I/O connections. When developing novel types of application it is often necessary to write some code that augments the basic facilities of the OS, but which is not necessarily specific to a particular application. A network application-level protocol such as the Secure Shell (Security 2001) for example can be considered as middleware in that it provides a functionality – end-to-end secure communication – that is not present in existing OS or network layers. Middleware is also used to meet some common needs of modern distributed applications where a legacy OS is inadequate. Application checkpointing for example would be a useful OS service (Allison 1994) but is almost always met by middleware such as libckpt (Plank et al 1995).

Middleware is associated with security and interoperability in the context of the DLE oriented Internet2 (IP2 1999) and ScotCIT programmes (Scotcit: 1998). Middleware can also be thought of as a means of meeting the QoS, in the wider sense, required by applications. In the context of DLEs middleware is concerned with the functionality and performance necessary to support the resources and the organisational framework used by end-users. DLE middleware issues (see Fig.1.2) include Sharing, Concurrency, Security, Coherence, Responsiveness, Availability, Replication, Distribution, Scalability, Timeliness and Interoperability.

Sharing implies multi-user concurrent access, necessitating facilities for concurrency control and conflict resolution. For example, if the Geometry tool in Figure 1.1 allows for co-operative design, then distributed concurrency control must be implemented to maintain the coherence of a DLE, that is, the maintenance of shared resource integrity when there are multiple concurrent accesses, even when the resource is distributed and/or replicated. The time needed to ensure the integrity of a shared resource (commit

protocols for example) can be at odds with the important end-user requirement for *responsiveness*, possibly the single most critical QoS need (Ramsay et al. 1998). Sharing also implies security. Security concerns include the maintenance of identities and controlled access to resources in the DLE. Structured protection facilities are necessary for both privacy and sharing.

Availability means coping with failures that can occur in either the network or the server. Providing a service based on distributed servers can negate periods of non-availability caused by server and network outages. These may be distributed across the wide area, possibly globally. If the same set of servers is also used to share the load then *replication* becomes an attractive strategy as it can, in principle, support *responsiveness* and *availability*, and it also offers a practical means of implementing incremental scalability. *Scalability* is a major concern for service providers - to what extent does the increased management overhead of replication and distribution limit their scope? How scalable should a system be? If there is a high variance in loads and a dynamically growing user base then initially provisioning a monolithic system of sufficient capability to meet peak demands is not a good solution as small changes in the scheduling of content or the size of the user base can quickly invalidate predictions about peaks. Consider an induction period for a class of a hundred students who are to use a DLE - that period can easily generate a load an order of magnitude or more than the average.

Timeliness is a generalisation of delay sensitivity that encompasses both *responsiveness*, and the requirements of continuous media resources, such as audio and video conferencing. Ideally, a DLE should support integrated video and audio resources to enable lecture distribution, small group tutorial meetings and team meetings. This QoS parameter is difficult to mitigate for at the middle level if the underlying infrastructure offers no guarantees of bandwidth and delay. The timeliness requirement for interactive continuous media stands in strong contrast to *reliable communication* in that bit-perfect copying of continuous media is not considered essential, but time of delivery is critical.

Finally, *interoperability* refers to the openness of a DLE with respect to de facto standards supported by other systems. Supporting LDAP-based authentication services

allows prospective users to use digital credentials maintained by a separate administrative body, external to the DLE. Educational content labeling and packaging standards such as IMS (IMS Global Learning Consortium Inc. 2000), and SCORM (Advanced_Distributed_Learning_Inc. 2001) are attempting to facilitate the routine import and export of resources between DLEs. At time of writing it is far from clear that any of these putative and competing standards should be embraced, as they are far from complete and seem better suited to the needs of the airline industry training programmes where they originated. The general principle however of using a component-based approach to system construction, can be endorsed as it allows for sharing and re-use of educational resources.

The TAGS framework (Allison et al. 1999, Allison et al. 2000, Allison et al. 2001a, Allison et al. 2001b) can be regarded as middleware for DLEs. It provides support for access control and group communication suitable for DLEs that in some way echoes basic OS support for these functions, but is at the right level for DLEs. It has provided a testbed for QoS analysis tools for DLEs, and extensions for distributed replication and integrating video conferencing.

1.7 Summary

A DLE is an Internet-based distributed system used in the delivery and management of educational processes. DLEs should support teamwork, real world input and anytime/anywhere learning. These pedagogical goals imply certain technical requirements, which are summarised in Table 1.2, grouped under the high-level headings of Coherence, Framework and QoS.

<i>Coherence (Chapter 4)</i>	<i>Framework (Chapter 5)</i>	<i>Quality of Service (Chapters 6 and 7)</i>
<ul style="list-style-type: none"> • Multiuser Applications • Teamwork • Information sharing • Concurrent Access • Distribution • Shared Resource Integrity 	<ul style="list-style-type: none"> • User Centric Views • Role Based Access • Group Membership • Services for Developers • Security • Interoperability • Abstractions for non-specialists 	<ul style="list-style-type: none"> • Responsiveness • Timeliness • Identifying Sources of Delay • Modeling delay • Measuring delay components • Service Architectures to support: <ul style="list-style-type: none"> <i>Scalability</i> <i>Availability</i> <i>Network Adaptability</i>

Table 1.2: DLE Systems Requirements

Chapter 2 identifies and describes the systems concepts used to meet DLE systems requirements, Chapter 3 describes and evaluates related work, Chapter 4 describes work which addresses the coherence problem, Chapter 5 describes the design and evolution of a production framework for DLEs, Chapter 6 is concerned with QoS – understanding and measuring the sources of delay as perceived by the end user of a DLE, Chapter 7 describes QoS-oriented service architectures for distributed replication and continuous media, and Chapter 8 concludes.

2 Systems Concepts used in Addressing DLE requirements

The previous chapter introduced the concept of a distributed learning environment (DLE), and identified three categories of systems requirements: QoS, Coherence, and a Framework. This chapter outlines the systems concepts used elsewhere in this thesis to meet these requirements in the support of DLEs. These building blocks are drawn from the established areas of computer networks, distributed systems and groupware. Table 2.1 shows an outline of this Chapter.

DLE System Requirements	Systems Concepts used in addressing requirements	Issues and Mechanisms
QoS (Section 2.1)	Computer Networks	Bandwidth, Timeliness, Reliability, Jitter Application classes ATM and QoS IP and QoS Patterns of network transmission and communication IP Multicast
Coherence (Section 2.2)	Distributed Systems	Event ordering and concurrency Group communication The Internet as a deployment platform
Framework (Section 2.3)	Groupware	Role-based access Shared Environments QoS requirements Security

Table 2.1: Systems concepts used in addressing DLE requirements

Section 2.1 outlines and discusses issues surrounding network QoS. Although QoS is typically associated with the requirements of interactive, continuous, time-sensitive media - video conferencing and Internet telephony for example - it is shown in this section that these types of traffic must be seen within the overall context of a range of different classes of network applications. As the Internet is the effective platform for the deployment of DLEs, its protocol semantics are assessed with reference to QoS. Network-level support for group communication is also discussed.

Section 2.2 details the particular aspects of distributed systems that are relevant to this thesis - event ordering, concurrency, group communication and coherence. Coherence

refers to the extension of concurrency control policies and the enforcement of event ordering across a distributed system. This is particularly relevant to interactive shared resources, where there is often a tension between integrity and responsiveness. Hence, to the extent that responsiveness is a QoS issue, an important aspect of the coherence problem is one in which the integrity of some shared state must be maintained for multiple concurrent readers and writers, without lengthening response times to unacceptable levels.

Section 2.3 describes *groupware*, a type of distributed system where user and group interaction is of prime importance. As the focus of group work is often some shared object or task, potentially involving multiple concurrent readers and writers, such resources require coherence. Much groupware starts from the point where group organisation and allocation of resources is a given (possibly seen as some middleware or OS function), but in a DLE it is important to have a framework for these functions. Accordingly, the concept of a groupware environment is introduced, and this provides the basis for the DLE framework described in Chapter 5. A distributed groupware environment or framework for distributed learning must have a security model. Security concepts are discussed and some current Internet security mechanisms are listed.

The Chapter concludes with a summary of the systems concepts identified, what their relevance is to a DLE, and in which parts of the thesis they are primarily used. However, it should be stressed that a DLE represents a critical combination of these concepts, and that their interactions can be the source of conflicting demands.

2.1 QoS, Computer Networks and the Internet

QoS-awareness is an important feature of DLEs. A DLE consists of a rich mix of network application classes. It is important to understand how to formulate the QoS requirements for a particular application, and also how to evaluate the success of the infrastructure and middleware in meeting these requirements. While it may not be possible to directly change the QoS that parts of the infrastructure provide, being aware of the real nature of such problems is important, as adaptations and recommendations can be made to make better use of the infrastructure.

The concept of QoS originated in the context of network infrastructure, and, although purportedly concerned with the end user's experience when using a network application, it is still in the infrastructure area that it is most advanced in terms of agreed issues, standards and actual implementation. The QoS that an application requires from the underlying network infrastructure can be broken down into four major distinct qualities: *bandwidth, delay, jitter and reliability*.

Bandwidth

Analogue bandwidth is measured in cycles per second, or hertz, whereas digital bandwidth is measured in bits per second. The different encoding and modulation schemes used by various network technologies can mean that the effective digital bandwidth can be more or less than the nominal analogue bandwidth available – modems for example (3.3KHz | 28Kbps), or ethernet (20MHz | 10Mbps). It is generally only meaningful to talk about “raw” bandwidth with respect to the physical and data link layers. The effective bandwidth obtained by a network application is measured end-to-end, and it is this figure that is meaningful in the context of QoS.

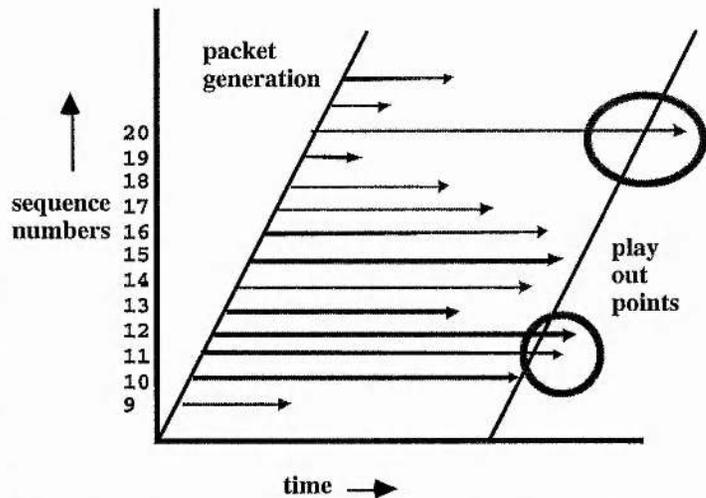
Delay

Even where massive bandwidth is available delay is still a separate component, fundamentally limited by the speed of light. In practice transmission through copper media is an order of magnitude less and transmission through fibre is even slower. The practical ramification is that a remote procedure call from St Andrews to San Francisco has a built-in delay of 100 milliseconds before any overheads are taken into account. As

network traversal always involves some degree of store and forward delay, and processing overheads, the end-to-end delay is typically substantially higher than the theoretical minimum. The significance of delay to an application can be characterised as its *delay sensitivity*. In some cases e.g. e-mail, ftp, the acceptable delay can be measured in minutes, or even hours. In strong contrast, an interactive video conference requires data to be delivered within milliseconds or it is too late to be of use. As mentioned previously, interactive responsiveness is a critical QoS parameter for the end users of network applications.

Jitter

Jitter refers to the variance of inter-arrival gap in continuous media traffic. Play-out buffers can be used to smooth out the jitter effect where there is little interactivity. The figure opposite shows a play-out buffer in operation.



The distance between the two diagonal lines is the size of the buffer and corresponds to the time period it can buffer packets over. The circles show packets that have missed their playout window and must be discarded. Where there is interactivity, such as Internet telephony or video conferencing, play-out buffers must be kept small, otherwise the timeliness needed for meaningful interactivity will not be present.

Reliability

Reliability means that data is copied from sender to receiver bit perfect, without loss or modification. Continuous media such as audio and video are *loss tolerant* in that a missing frame or audio packet can go unnoticed by the person using the receiving application. Traditional data networking protocols such as ftp and e-mail assume that

total reliability is essential. Such applications would not be acceptable if bits of data were allowed to go missing. Browsers are interesting in that a Web page can still be usable even if some components (such as advertising images) fail to arrive reliably and cannot be displayed.

Table 2.2 shows a network-centric view of application classes with different QoS requirements.

Application class	Examples	Bandwidth	Delay Sensitivity	Jitter Sensitive?	Reliability (Loss Tolerance)
Filler traffic	news updates	medium	hours	no	no
Unattended data/text transfer	e-mail	medium	minutes	no	no
Unattended bulk data transfer	file transfer programs	plenty	minutes	no	no
Control traffic	UNI.3-1, ICMP, IGMP	low	milliseconds	no	sometimes
Interactive video	video conferencing, distributed classroom	high	milliseconds	yes	yes
Interactive audio	Telephony	medium	milliseconds	yes	yes
Interactive text/data	telnet, X-windows	medium	milliseconds	no	no
Interactive text/data graphical/data	WWW	medium	milliseconds/seconds	no	sometimes
Video messaging	multimedia e-mail	high	seconds/minutes	no	yes
Image messaging	high-resolution fax	low	minutes	no	yes
Video distribution	Television	high	minutes/hours	no	yes
Audio retrieval	audio library	medium	seconds/minutes	no	yes
Financial information	Share prices	medium	seconds	no	no
Distributed file service	databases, network file systems	medium	milliseconds	no	no

Table 2.2: A network-centric view of application classes and QoS requirements

The shaded rows are particularly relevant to DLEs because they represent the wide mix of types of educational resources a DLE is expected to provide, and span the full range of major QoS requirements.

An important distinction between different types of traffic is whether timeliness is more important than reliability. For example, an interactive video-conference can tolerate the loss of some data in transit, but will run into severe difficulties if packets are not i) delivered within a given time frame and ii) suffer from a degree of jitter that requires a play-out buffer of more than 300ms. Stock market share prices are unusual as they are one of the few applications where reliability and timeliness in seconds are both equally

important. The Finesse DLE (Power et al. 1998), described in Chapter 5, includes a portfolio management facility which uses real-time data from the London Stock Exchange. Even minor outages of this data feed are a source of concern to groups of staff and students who work as fund management teams.

Network QoS is difficult to achieve wherever there is contention for finite network resources. It is even more difficult to meet when there is a widely varied set of application QoS requirements. However, mechanisms such as resource reservation and admission control, when combined with traffic shaping and the policing of current connections have been shown to allow QoS requirements to be met. Asynchronous Transfer Mode based networking has been one of the most successful technologies in providing QoS. It is described next, as an example of how to effectively support the type of QoS mix shown in Table 2.2, before we consider the QoS possibilities when using the Internet.

2.1.1 Meeting QoS Requirements: Asynchronous Transfer Mode (ATM)

Asynchronous Transfer Mode (ATM) (Goralski 1995, Handel et al. 1994, Minoli & Schmidt 1997) was developed as a technology which could meet the QoS requirements of real time interactive video and voice and also support the statistical multiplexing required for the cost effective reliable transport of computer generated data. Its proponents held out a vision of a single, multi-purpose network with a common data unit from desktop to desktop. The basic ATM unit of transfer is the cell, a fixed size 53-octet packet consisting of 48 bytes payload and 5 bytes header information. The relatively small size of the cell was intended to cater for legacy telephone networks. As applications other than interactive audio require larger units of data the adaptation layers were specified. These in turn require segmentation and re-assembly (SAR) processing to map them to and from cells. The cost of SAR means that substantial processing power has to be available on ATM end-user network interface controllers (NICs). QoS parameters are often specified in high-level terms of constant bit rate, (non) real-time variable bit rate, available bit rate and unspecified bit rate. ABR or UBR are typically used for carrying Internet traffic.

ATM is connection-oriented. During the connection setup phase QoS parameters can be negotiated. The connection-oriented nature of ATM means that, by controlling the admission of calls to the network, QoS guarantees that have been made to existing connections can be met provided traffic is shaped to behave in predictable ways and policing is used to prevent abuse. Table 2.3 summarises the scope of approved ATM version 4 QoS parameters that may be negotiated on an end-to-end basis (The_ATM_Forum 1996).

<i>Service Categories</i>							
<i>Attributes</i>	CBR	rt-VBR	nrt-VBR	UBR	ABR		
<i>traffic parameters</i>						CBR	constant bit rate
PCR,CDVT	√	√	√	√	√	rt-VBR	real time variable bit rate
SCR,MBS,CDVT	n/a	√	√	n/a	n/a	nrt-VBR	non real time variable bit rate
MCR	n/a	n/a	n/a	n/a	√	UBR	unspecified bit rate
<i>QoS parameters</i>						ABR	available bit rate
Peak-to-Peak CDV	√	√	-	-	-	PCR	peak cell rate
Max CTD	√	√	-	-	-	SCR	sustainable cell rate
CLR	√	√	-	-	√	MBS	maximum burst size
<i>other</i>						MCR	minimum cell rate
Feedback	-	-	-	-	√	CDV	cell delay variation
						CDVT	cell delay variation tolerance
						CDT	cell transfer delay
						CLR	cell loss ratio

√ attribute is specified for service, - not specified

Table 2.3: The Scope of ATM QoS.

ATM has been, and is being, used to support local area, metropolitan area and wide area networks carrying telephone, video and data payloads. In 1996 The Scottish Higher Education Funding Council invested in an ATM infrastructure, believing this to be a crucial move towards sharing of educational resources and widening of access. The infrastructure unfortunately tended to stop at the main campus router, although the Scottish MAN's Video Conferencing Network was a notable exception.

The eventual lack of success for ATM as an end-to-end technology was largely caused by the relatively high cost of running it to the desktop. The original ATM Network Interface Cards (NICs) were only available with optical fibre interfaces, which meant installing fibre to the desktop – an unattractive extra cost as most offices and classrooms are already wired with copper twisted pair cable. Another drawback is that as ATM cells are a small, fixed size they need to be assembled into larger units for many purposes and this high Segmentation and Reassembly cost is either born by the host processor, or by a processor on the NIC. In the case of IP/ATM AAL-5 is specified which supports full size (64K) IP packets, which can require intensive SAR. When SAR is performed on a

desktop host overall system performance suffers, and if it is done on the NIC then the cost of the NIC goes up.

During the initial ATM deployment phase (1994 - 1997), "Fast Ethernet", the IEEE 802.3u 100Mb/s standard (IEEE 1995) was developed and made available to the desktop and LAN infrastructures. Fast Ethernet NICs use existing copper wiring and are relatively cheap. Economies of volume production have resulted in low-priced, high-performance Fast Ethernet *switches*. In addition, there was already widespread support in place for IP/Ethernet. Hence this relatively cheap drop-in upgrade to existing ethernet LAN connections displaced ATM, which was then largely confined to the wide and metropolitan area roles, preventing its utilisation as an end-to-end technology. As QoS is an end-to-end issue this means that i) the strong potential for realising QoS through the use of ATM has not been achieved ; ii) ATM has failed in the desktop market place; and iii) practical solutions using today's deployed technology will be IP-based.

2.1.2 The Internet Protocols

The Internet is the platform for DLEs. It is therefore important to know how it works and what it provides. It is equally important to know what it does not provide, as that functionality may need to be implemented as part of systems support for a DLE. Computer networks use the concept of *layering* to provide a vertical communication abstraction and *protocols* to provide a horizontal communication abstraction. Protocols are rules for meaningful interaction between two or more participants. Each layer is connected by one or more protocols horizontally, and one or more interfaces vertically. A 5-layer model for the Internet (Kurose & Ross 2001) (Tanenbaum 2002) has recently gained credence (Fig 2.1).

<i>Layer</i>	<i>Purpose</i>	<i>Example protocols</i>
Application	Supporting network applications	ftp, http, smtp, ssh, rtp
Transport	Host to host transfer	tcp, udp
Network	Addressing and routing of datagrams through the Internet	ip, icmp
Data Link	Data transport between directly connected network entities	hdlc, ppp, ethernet, ATM
Physical	physical media and technologies: fibre, copper, wireless, infrared, satellite, microwave, dwdm	bit encoding for underlying media type

Figure 2.1: The 5-layer Internet Model

A wide area network, or Internet, facilitates messages to be transmitted from one node to another across many intermediate routers and other devices that are directly connected to each other by physical media links. The current IP protocol, that is, the Internet Network layer, is used by millions of computers. It is an unreliable, best-effort, datagram delivery service. IP is an example of a *connectionless* protocol. Each datagram contains the full destination address and is routed independently from source to destination. As no intrinsic association exists between datagrams, they can disappear, get duplicated, arrive out of order and suffer significant variation in delay. By analogy, if a Ph.D is sent in the post as eight different chapters enclosed in eight different envelopes, all with the same address, there is no guarantee they will take the same route or even arrive in the order they were sent. The data link layer, in contrast to the network layer, is only concerned with the transmission of data between directly connected nodes. The data link level is responsible for providing reliable communication over the underlying physical connection. This is achieved using numbered frames, acknowledgements, timeouts and repeated transmissions. For example, automatic repeat requests are often used in conjunction with sliding window protocols to provide a reliable point-to-point delivery while making efficient use of available bandwidth. These mechanisms are well documented in the literature. Failure detection at the link level is well specified and understood.

Reliable transmission across the Internet is achieved at the transport level by the Transmission Control Protocol, TCP. The primary purpose of the transport layer is to

provide process-to-process communication between nodes, as opposed to network level delivery. It follows that there can be more than one transport level protocol using the same underlying network protocol. In the IP model it is possible to have either a connectionless or connection-oriented transport layer. The two most common Internet transport protocols are TCP and the User Datagram Protocol, UDP.

- TCP is connection-oriented, and provides a reliable byte-stream abstraction.
- UDP is connectionless, it is simply a transport level wrapper around the basic IP network level datagram.

Application protocols are typically represented by processes. Processes are addressed within an Internet node by 16-bit port numbers, relative to the transport protocol. There are potentially 2^{16} UDP ports and 2^{16} TCP ports available in a single Internet node. Some of these ports are reserved for processes which listen on behalf of well-known services such as e-mail, web traffic, and the DNS.

The significance of the relationships between the Internet data link, network, transport and application protocols is this: it is assumed by TCP that a missing packet or acknowledgement is not a sign of a faulty link (as the data link level connections already correct such failures) but is a sign of congestion on the Internet. TCP accordingly backs off, lest it worsens matters by simply re-transmitting lost packets at the same rate and making the congestion worse. This has implications for the Web, as HTTP uses TCP. These implications are investigated in Chapter 6, as part of supporting QoS awareness in DLEs.

At present all IP traffic is treated on a first-come first-served, best effort basis. By the time that a transitory peak has overrun a queue at a router it is too late to take corrective action. There is no means of explicitly signaling congestion in IP. At the network level packets may be lost, delayed, duplicated or turn up out of order. There is no mechanism for end-to-end signaling, no admission control, and no notion of a virtual circuit. For data which has no urgency this can be solved by TCP which provides a reliable byte stream abstraction at the transport level. However for continuous media traffic, or interactive services where timeliness is important, TCP is not a solution. Numerous

solutions have been proposed, but migrating the vast structure that is the Internet to a different way of working is a non-trivial task, technically, economically and politically. Economically, the Internet crosses many different administration and ownership domains, and this exacerbates the technical problem of guaranteeing QoS from end-to-end.

QoS on the Internet

There are three current IP initiatives which are relevant to QoS: IPv6, IntServ and DiffServ.

IPv6 (Bradner & Mankin 1995) is the next generation IP protocol. It is fully specified and there are working implementations available for most operating systems. It is not directly compatible with the current IP, which is version 4, (IPv4). Amongst various changes from IPv4, IPv6 includes a 20-bit flow-id label field in the header of every datagram. The idea is that a resource reservation protocol such as RSVP (Braden et al. 1996) can ask the routers between two or more hosts to guarantee certain traffic shape requirements. After a period of negotiation a unique flow-id is assigned and should be used by all intermediaries to identify which packets belong to that flow. IPv6 has not yet been taken up widely because the main pressure for its deployment, IPv4 address space exhaustion, has been alleviated by relaxing the original IPv4 addressing scheme through techniques such as Classless Internet Domain Routing, Network Address Translation and leasing of IP addresses for the duration of a dial-up session.

The two current schemes for supporting QoS on IPv4 or IPv6 are the Integrated Services model (IntServ) and the Differentiated Services model (DiffServ).

The Integrated Services model (White 1997) attempts to support QoS for particular end-to-end flows. This involves moving the IP routing infrastructure from a datagram-oriented to a connection-oriented model. RSVP is closely associated with IntServ, as a protocol is necessary to set up the traffic guarantees across a circuit of IP routers. In some respects the IntServ model sits well with RSVP and IPv6 using ATM as a carrier. However, the maintenance of this type of state information in IP routers is not attractive to manufacturers, who regard the complexity as potentially damaging to overall

performance and who are also wary of the high costs in R&D, for uncertain short term returns.

The Differentiated Services Model (Blake & al 1998) is more recent than IntServ, and to some extent is a response to concerns about the complexity of IntServ. The basic premise is that even two levels of service in the Internet is one more than is currently available, and hence an improvement. DiffServ posits a small number of differentiated service classes, e.g. bronze, silver and gold. These can be aggregated at the egress and ingress points of domain boundaries without sophisticated signalling. In addition, very importantly, DiffServ can work with existing IPv4 traffic through the redefinition of some standard bits in IP datagram headers that are generally unused for their original purpose.

	Integrated Services	Differentiated Services
Granularity of Service	individual flow	aggregates of similar flows
Signaling	essential e.g. RSVP	existing TOS field
Admission Control	essential, all routers	edge routers only
Scalability	limited by number of flows	limited by number of service classes
Network management	end-to-end circuits	similar to existing IP autonomous systems

Table 2.4: Differences between IntServ and DiffServ

The key differences between IntServ and DiffServ are summarised in Table 2.4. DiffServ is proving attractive to major IP infrastructure providers because, in comparison with IntServ, it requires much less complexity in routers and a generally smaller commitment to change. Whether or not it will help to deliver QoS to the end user remains to be seen. With reference to distributed learning environments, the Internet-2 Qbone (Teitlebaum et al. 1999), described under related work in Chapter 3, is based on DiffServ.

2.1.2.1 Server-side QoS concerns

As infrastructure QoS is an end-to-end property the non-network components involved must also be evaluated. Accordingly, server performance is a primary concern. Several studies have shown that the source of poor response and unacceptable service quality

can be traced to server performance. In (Bhatti & Friedrich 1999) an empirical investigation is reported that involved instrumenting the service traffic of a major ISP. The network run by the ISP consisted of hundreds of servers supporting IP-based services that were accessed via 200 points of presence in the US. Average response times for a 40K text was found to be in the order of 2 seconds, while peak loads could take up to 7 seconds. The major source of variance in delay was server loading. Chapter 6 describes a methodology for identifying the source of poor responsiveness in web connections for DLEs, and has found that server-side performance can be the critical component.

2.1.3 Patterns of Network Transmission and Communication

The network concepts described up to this point have assumed a one-to-one pattern of communication. It is however important to understand how group communication at higher levels can best be supported by network protocols. We distinguish between patterns of *transmission* and *communication*. Transmission is one-directional (simplex) whereas communication implies bi-directional (duplex) traffic.

Patterns of network transmission can be classified as *one-to-one*, *one-to-all*, or *one-to-many*. *Broadcast* may be characterised as one-to-all, *unicast* as one-to-one, and *multicast* as one-to-many. Broadcast and unicast can be thought of as special cases of multicast. Multicasting addresses a subset of all possible recipients. For a broadcast the subset is all recipients, and for a unicast the subset is one recipient.

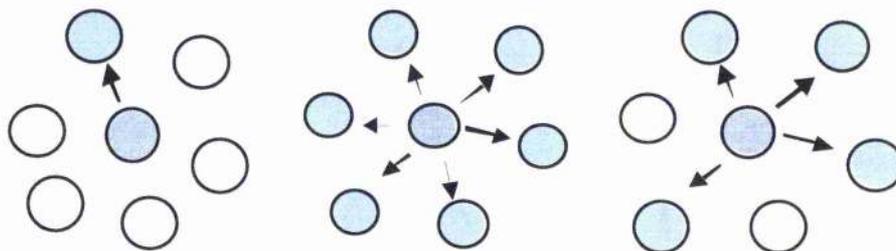


Figure 2.2a: Unicast

Figure 2.2b: Broadcast

Figure 2.2c: Multicast

We distinguish between true multicast and simulated multicast, the latter being implemented by multiple unicast or filtered broadcast. A true multicast is a single

message that is received by all members of a group, regardless of the group size. In the multiple unicast scenario N messages are required where N is the size of the group. In the filtered broadcast situation a hardware broadcast is used for all communication, and software filters are constantly employed to select items of interest. This is objectionable because of the high-load it places on the network and all connected hosts. In true multicast the network interface hardware supports multicast addressing and only passes on messages of interest to the host software. Simulated multicast is useful in situations where no true multicast is available but explicit group communication is still desirable as a means of expressing the solution to a problem.

Protocols supporting these patterns can operate at any level in a layered model. At the data link level the IEEE 802 family of local area networks explicitly support unicast, multicast and broadcast transmission in their architecture and addressing. An attractive feature of shared media local area networks such as ethernet and token ring is that a multicast or broadcast message does not use more time or resources than a unicast message. At the network level IPv4 explicitly supports multicast as part of its addressing scheme (Class D address types). At the transport level UDP supports multicast in that it is a transport-level wrapper around the basic IP datagram. TCP on the other hand is strictly unicast.

A pattern of communication allows for both transmission and reception between a node and its correspondents. Trivially, a one-to-one pattern of communication involves redrawing Figure 2.2a with a two-headed arrow. The same simple modification does not hold for broadcast and multicast. In figures 2.2b and 2.2c each node can in principle broadcast and multicast to the other nodes. But this is not the same as redrawing the arrows with two heads. That implies that the transmitter will receive replies from all recipients. These replies may form part of a reliability protocol feature. It is clear that reliable multicast or broadcast, based on acknowledgements will not scale well.

Many-to-one has emerged as a popular pattern of communication in the Internet, reflecting the request/response nature of the protocols and the client server architecture. A single Web or e-mail server for example manages multiple TCP point-to-point connections from many clients, as shown in Fig.2.3.

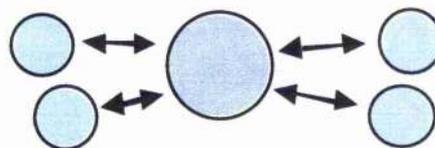


Figure 2.3: many-to-one communication

2.1.3.1 IP Multicast

IP multicast is of particular interest as it would seem to provide a good basis for group communication on the Internet. The potential of the Internet for multicast transmission has long been recognised. Boggs wrote in 1983 “VLSI will soon permit low-cost implementation of multicast mechanisms in hardware, and people should be thinking of applications”(Boggs 1983). The experimental Internet Multicast backbone – the Mbone (Deering 1989) – was developed to support multicast transmission on the Internet. Multicast traffic is efficiently supported by co-operating routers that dynamically maintain a spanning tree of point-to-point links between networks in response to user-initiated joins and removes. A simple example is given in Figure 2.4. In 2.4a multicast transmission is implemented as multiple unicast. This means that the number of messages that the sender must transmit, and the number of routed packets handled by each router, increase linearly with the size of the group. In Fig. 2.4b the routers are multicast aware, so regardless of how many members are in the groups outside of Net 1 the sender only needs send one message to each of the routers. The effect cascades in that networks connected to Nets 2 and 3 by multicast aware routers will only require one copy of each message sent to their multicast-aware routers.

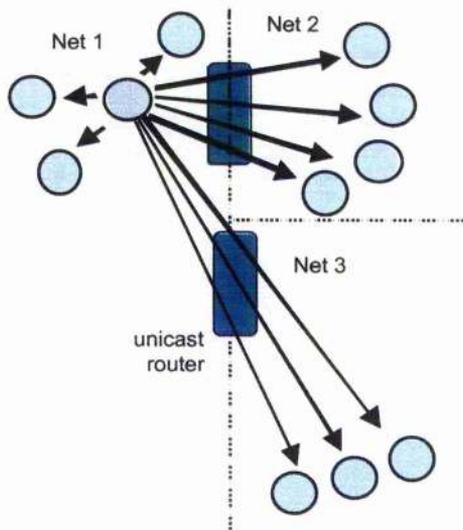


Figure 2.4a:
Multicast using multiple unicast

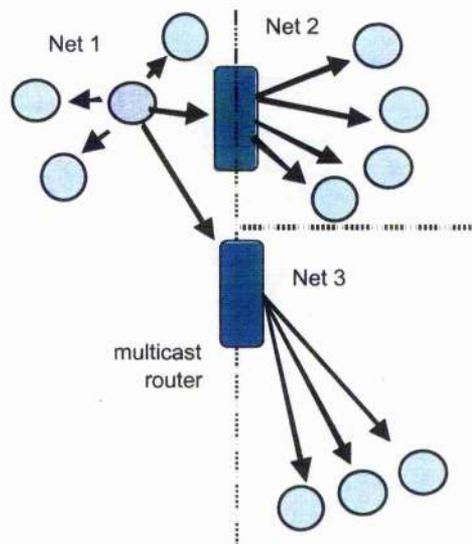


Figure 2.4b:
IP multicast across three networks

The dynamic maintenance of the Mbone is achieved through use of the Internet Group Membership Protocol (IGMP). IGMP manages joins and leaves for leaf nodes, and uses true multicast wherever possible by mapping IP multicast addresses to link-level multicast addresses. That is to say, where a set of IP multicast group members are attached to the same local area network, that network's native mechanism for multicast support will be used.

The static structure of the Mbone is configured manually, as a set of tunnels between multicast routing processes. These processes were originally run on Sun Unix nodes, and did not offer the same throughput capability as large, dedicated unicast routers. However, the efficiency and scalability of the approach resulted in multiple concurrent audio and video streams being transmitted across the Internet. More recently IP multicast routing functionality has been included as part of major Internet routers, thereby promoting the potential for higher bandwidth IP multicast connections.

Some multicast addresses are reserved for special purposes. These include the IGMP join/leave functions for maintaining spanning trees of multicast sessions, and for making session announcements. If a host wishes to announce an event on the Mbone, it can use the Session Directory Protocol, SDP (Handley & Jacobson 1996), to make the event "public". Similarly, any other node connected to the Mbone may obtain the list of

available sessions and join one or more of them. The effect is like that of a spectrum of broadcast channels from which a user can select.

2.2 Coherence, Distributed Systems and Event Ordering

DLEs are a form of distributed system. They involve shared objects which may be replicated, cached, and which are subject to multiple concurrent updates. An understanding of the nature of event ordering in distributed systems is essential for both passive activity such as analysis and debugging, and also for pro-active activity such as the removal of uncertainty through enforcing specific types of event ordering.

The concrete definition of a distributed system used in this thesis is a collection of independent, autonomous computer systems, connected by one or more networks, that communicate and co-operate with each other in pursuit of a task. The collection may be formed from a brief association, or it may be a longstanding arrangement. The networks used in the systems described in this thesis are almost always part of the Internet.

The underlying abstract concept of a distributed system used here is in keeping with the family of definitions which are content to trace their ancestry back as far as Lamport (Lamport 1978). Lamport's seminal paper describes a distributed system as one consisting of sequential processes which may communicate with each other using messages. Events may be related by *causal precedence*. The causal precedence relation " \rightarrow " is defined:

1. If **a** and **b** are events in the same process and **a** happens-before **b**, then $\mathbf{a} \rightarrow \mathbf{b}$.
2. Causal precedence is transitive: If $\mathbf{a} \rightarrow \mathbf{b}$ and $\mathbf{b} \rightarrow \mathbf{c}$ then $\mathbf{a} \rightarrow \mathbf{c}$.
3. If **a** is the sending of a message *m* by one process, and **b** is the receipt of message *m* by another process, then $\mathbf{a} \rightarrow \mathbf{b}$.
4. Two distinct events **{a, b}** are said to be concurrent $\mathbf{a} \parallel \mathbf{b}$ if:

$$\neg(\mathbf{a} \rightarrow \mathbf{b}) \wedge \neg(\mathbf{b} \rightarrow \mathbf{a}).$$

The causal precedence ordering relation is based on logical (relative) time rather than wall-clock (elapsed) time. Causality in this context means that related events are ordered in such a way that one could potentially have caused the other. So, $\mathbf{a} \rightarrow \mathbf{b}$ does not mean that event **a** necessarily caused event **b**, but rather that it *may* have. To emphasise the logic of causal precedence some writers prefer phrases such as:

"**e** may causally affect **e'**" (Peterson et al. 1989).

A common characterisation of a distributed computation which has been derived from Lamport's model is as follows: a distributed computation describes a single execution of a distributed program by a collection of sequential processes $p_0 \dots p_n$. Each sequential process generates a series of events which

- (i) may be local to a process and cause the local state to change
- (ii) may involve communication with another process (a send or receive event).

The local history of a process p_i during a computation is a sequence of events $h_i = e_i^1 e_i^2 e_i^3 \dots$. This may be represented by a space-time diagram such as Figure 2.5. The vertical arrows show the local histories $\{h_1, h_2, h_3\}$ of three processes $\{p_1, p_2, p_3\}$ which respectively consist of events $\{e_1^1, e_1^2\}$, $\{e_2^1, e_2^2, e_2^3, e_2^4\}$, $\{e_3^1, e_3^2\}$.

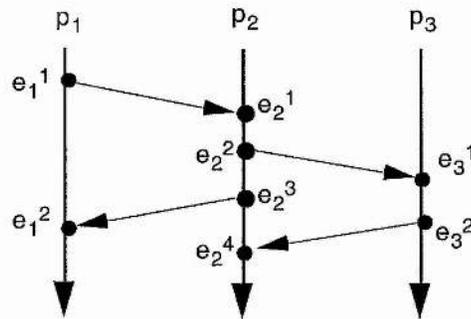


Figure 2.5: A Space-Time Diagram, Unicast messages

The global history of a distributed computation is a set $H = h_1 \cup \dots \cup h_n$ containing all its events. In the case of Fig. 2.5 the global history, $H = \{ e_1^1, e_2^1, e_1^2, e_2^2, e_2^3, e_2^4, e_3^1, e_3^2, \}$.

The diagonal arrows in Fig.2.5 represent unicast messages between processes. Each unicast message is associated with exactly one send event and one receive event.

Causality is transitive. Event e_1^1 causally precedes e_2^1 and e_2^1 causally precedes e_3^1 so e_1^1 causally precedes e_3^1 . The ordered event set $\{ e_1^1, e_2^1, e_2^3, e_1^2 \}$ could represent a remote procedure call (RPC) from p_1 to p_2 , and the set $\{ e_2^2, e_3^1, e_3^2, e_2^4 \}$ could represent an RPC from p_2 to p_3 .

Concurrency is not transitive. The sets of concurrent events in Fig. 2.5 are:

$\{ e_2^3, e_3^1 \}$, $\{ e_2^3, e_3^2 \}$, $\{ e_1^2, e_2^4 \}$, $\{ e_1^2, e_3^2 \}$. Event e_2^3 is concurrent with e_3^1 and e_3^2 but e_3^1 is clearly not concurrent with e_3^2 .

2.2.1 Passive and Proactive Applications of Distributed Systems Theory

A distributed system, as illustrated in Fig. 2.5, consists of a partially ordered set of events. This understanding can be applied in broadly two ways: passive and proactive. In the passive scenario the tagged messages are recorded and analysed. Passive applications aim to achieve non-intrusive monitoring with a view to discovering and understanding some property of the system under observation – the evaluation of a global predicate for example. Proactive applications are used to enforce event orderings

in order to achieve some form of coherence and control. In both modes messages are tagged with some form of timestamp. For example, the use of event tags, as shown in Figure 2.5 could be used as the basis for post-mortem debugging of a distributed computation. In addition to recording the events and their tags, message logs at each site are used to match *sends* to *receives*. A further log is kept of any non-deterministic choices made in each sequential process, such as random number generation. The computation is replayed (deterministically) under the control of a debugger and distributed breakpoints can be set. Live debugging is a more difficult task and must involve some form of wrapper around each process which reports periodically to the debugging controller. Work which achieved this on a transputer-based system is described in (Burgess et al. 1994).

2.2.2 Lamport's Clock Algorithm: A Proactive Approach

Lamport's Clock algorithm (Lamport 1978) is an example of a proactive approach. It is similar to the well known bakery algorithm (Lamport 1974) but is described in the more general conceptual framework of logical time. It effectively removes concurrency from a distributed system by enforcing a total event ordering which is agreed on by all processes (which execute the algorithm). Once a total ordering is agreed then higher-level algorithms such as mutual exclusion can be built on top of it.

- A clock C_i is defined for each process P_i .
- The clock is local to the process. Any monotonically increasing counter will do. It has no relationship with elapsed time and is referred to as a logical clock.
- Event e_i in process P_i is labeled with a timestamp $C_i(e)$.
- An ordering is possible for all events belonging to one logical clock i.e. if a_i precedes b_i then $C_i(a) < C_i(b)$.

An ordering is possible for two events belonging to two different clocks if the events are the sending and receipt of a message i.e. if P_i sends a message to P_j then $C_i(a) < C_j(b)$.

The system is correct iff: for any events a, b : if $a \rightarrow b$ then $C(a) < C(b)$. The rules which will enforce this are:

- Each process P_i increments C_i between any two successive events.
- If event a is the sending of message m by process P_i then m contains a timestamp $T_m = C_i(a)$.
- Upon receiving message m with time stamp T_m process P_j sets $C_j \geq \max(C_j, T_m + 1)$.

We can now order all events in a distributed system totally: if a is an event in process P_i and b is an event in process P_j then $a < b$ if, and only if, either, $C_i(a) < C_j(b)$, or $C_i(a) = C_j(b)$ AND $P_i < P_j$.

Lamport's clock algorithm as described above requires a reliable one-to-one messaging system e.g. a TCP connection.

2.2.3 Group Communication

So far we have concentrated on one-to-one patterns of communication in distributed systems. Two party communication is a special case of group communication. The former is simpler to model and understand than the latter, but in practice DLEs must

support arbitrary groupings of users and shared objects. An interesting consequence of making group communication available is that it changes the way developers approach problems and tackle systems organisation. It can simplify the coding of many parallel and distributed algorithms and provides an appropriate logical basis for group computations. Group communication has been applied in the following areas:

- fault tolerance (Hadzilacos & Toueg 1993) (Montgomery 1994) (Chang & Maxemchuk 1984)
- resource discovery and location in distributed environments (Allison et al. 1996)
- distributed operating systems (Kaashoek & Tanenbaum 1991)
- shared data objects (Bal 1990)
- parallel programming on distributed systems (Babaoglu et al. 1992)
- distributed algorithms (Birman et al. 1991)
- computer supported co-operative work
- Distributed Learning Environments (Allison et al. 2001)

Given the utility and potential of group communication where should it be implemented in the layered model – as part of the networking levels or as a distributed application protocol? Figure 2.6 distinguishes between group communication at the data link, the network and the application levels. One-to-many transmission at the IP network level is explicitly supported by the Mbone (see previous section). As all the work of joins and leaves and efficient data transfer is delegated to the network this would appear to be a good basis for providing group communication at the application level. The drawback with IP multicast is that it offers no end-to-end reliability, and must be augmented in some way if data is intended to be copied bit perfect from a sender to all the receivers in a group.

<i>Application</i>
Distributed and Groupware systems
<i>Network</i>
IP Multicast
<i>Data Link</i>
IEEE 802.x: ethernet, token ring, WiFi,

Figure 2.6: Group communication at different levels

The PIPVIC project, described in Chapter 3, investigated the use of IP multicast-based groupware in a variety of educational contexts.

2.2.3.1 Group Communication and Coherence

Figure 2.5 showed a messaging system where each message has exactly one sender and one receiver. A considerable body of distributed systems research is concerned with *group-oriented communication* in which there are many receivers for each message sent. For example, much work has focussed on extending Lamport's notion of causality into a group setting using a multicast pattern of communication (Birman et al. 1991).

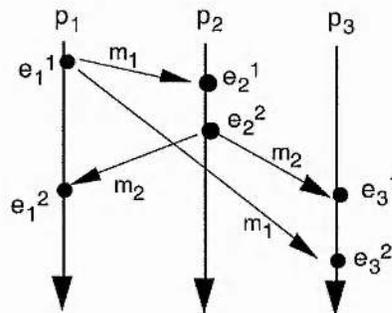


Figure 2.7: Multicast messages, Causality violation

Figure 2.7 illustrates a causality violation in a multicast message system. That is, p_3 sees message m_1 after m_2 although m_1 causally precedes m_2 . This is an instance of the coherence problem.

2.2.3.2 Enforcing Event Orderings in Group Communication

The proactive approach enforces event orderings in order to reduce uncertainty and provide a cleaner basis for application development. In a proactive system ordering semantics are enforced by decoupling the receipt of messages from their delivery to the

destination process. Figure 2.8 shows the main part of processes P1, P2 and P3 in unbroken lines, and their concurrent receiver threads in broken lines.

While the system is now coherent, in that events follow a known type of order (causal), the effect is to slow down the system – the gap between m2 being received by P3 and being delivered could be substantial. This is an example of a recurring problem with maintaining coherence in distributed systems – the tension between time and consistency.

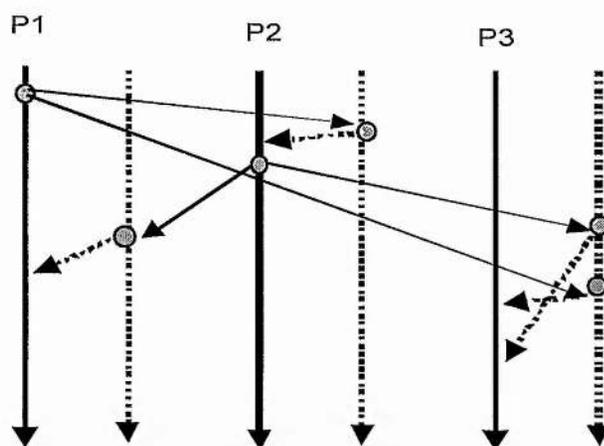


Figure 2.8: The causality violation of 2.7 is repaired

The mechanism that is widely used for achieving this type of effect is a vector version of Lamport's logical clock. Delivery of a message m only takes place when its logical clock value fits the ordering scheme in use. Basically, in an N participant group each node keeps a vector of size N , which contains a logical clock value for all members. So, in the system shown in 2.16 each node would maintain a vector clock of size 3, and only deliver messages when their clock value met the requirement of the ordering semantics. Extending Lamport's simple happened-before to a group system is based on the following vector clock calculus:

Equal	$t^a = t^b$	iff	$\forall i, t^a[i] = t^b[i]$
Not Equal	$t^a \neq t^b$	iff	$\exists i, t^a[i] \neq t^b[i]$
Less Than or Equal	$t^a \leq t^b$	iff	$\forall i, t^a[i] \leq t^b[i]$
Not Less Than or Equal	$\neg t^a \leq t^b$	iff	$\exists i, t^a[i] > t^b[i]$
Less Than	$t^a < t^b$	iff	$(t^a \leq t^b \wedge t^a \neq t^b)$
Not Less Than	$\neg t^a < t^b$	iff	$\neg (t^a \leq t^b \wedge t^a \neq t^b)$
Concurrent	$t^a \parallel t^b$	iff	$\neg(t^a < t^b) \wedge \neg(t^b < t^a)$

We can now say that event **e** precedes event **f** if, and only if:

- **f** has a counter value for the process in which **e** occurred greater than or equal to the number of events in that process up to **e** inclusive, and
- event **e** has a counter value for the process in which **f** occurred strictly less than the number of events in that process up to **f** inclusive

$$e_i \rightarrow f_j \text{ iff } Te_i[i] \leq Tf_j[i] \wedge Te_i[j] < Tf_j[j]$$

The advantage of a vector clock approach is that the partial ordering of events in a distributed system is captured. This form of tagging can be used passively for debugging and replay, or proactively to enforce causal ordering, but without obscuring concurrency.

Vector clocks appear to have been introduced by contemporaneously by Fidge (Fidge 1988) and Mattern (Mattern 1989). Birman and others use them widely in the ISIS toolkit (Birman 1991). Variations such as matrix clocks have also appeared in the literature.

2.2.3.3 A Proactive Group Protocol

The Birman-Stephenson-Schiper Protocol (Birman et al. 1991) is an example of applying vector clocks proactively to maintain either causal or total ordering in a distributed system. The algorithm for enforcing causal order is given below.

1. before a message *m* is sent the sender P_i increments its vector clock $VT_{P_i}[i]$ and then tags the message with the new time, V_{tm} .

2. on receipt of a message P_j does not deliver it to the destination process until two conditions are satisfied:

a) $VT_{P_j}[i] = V_{tm}[i]-1$

b) $VT_{P_j}[k] \geq V_{tm}[k] \forall k \in \{1,2,\dots,n\} - \{i\}$

3. When a message is delivered at process P_j , VT_{P_j} is updated by the following

$$\text{rule: } \forall k, VT_j[k] := \max(VT_j[k], V_{tm}[k])$$

Figure 2.9 shows the BSSP being applied to the system in Figure 2.7 in order to achieve the effect shown in Figure 2.8. The dotted lines are receivers threads that only deliver the message to the main thread when conditions 2a and 2b above are met.

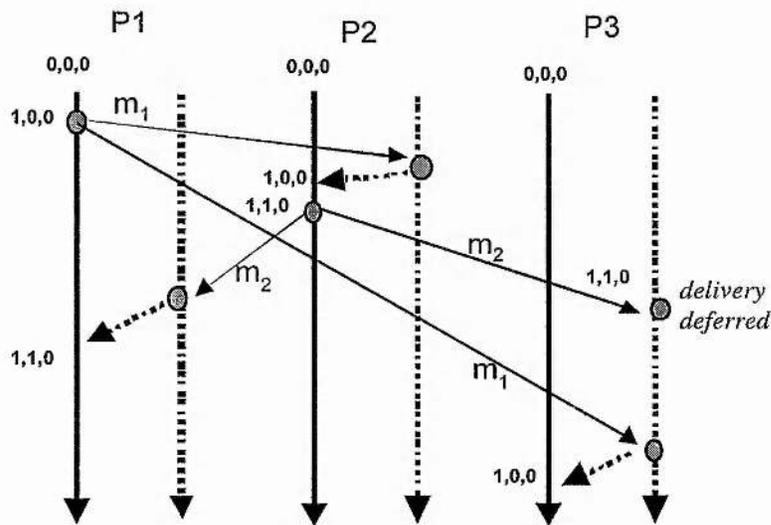


Figure 2.9: Enforcing causal ordering using the BSSP

2.2.4 Distributed Systems and the Internet

In practice the Internet, warts and all, is the deployment platform for distributed learning environments, so it is appropriate to reflect on what that means for distributed systems.

Timeliness

Routes traversed by packets from source to destination are not guaranteed to be stable or, in the case of two-way communication, symmetrical. Furthermore routers can drop packets or fail. This means that all packets are subject to variable delay.

Reliability

Messages may be sent over TCP or UDP, or over higher-level protocols which use these protocols. Messages sent over UDP are unreliable, and may arrive more than once, or not at all, but typically have much lower latency than messages sent over TCP. An application-level protocol such as Remote Procedure Call may be implemented over UDP or TCP. If UDP is used the RPC protocol must implement reliability, whereas if TCP is used reliability is already provided.

Failures

As previously noted, failure detection of senders and receivers is a relatively straightforward feature to implement in synchronous connections at the link-level where the time taken to transmit a message and receive a reply is predictable. Connections across the Internet do not have this property, and this can make end-point failure detection difficult. This has potentially serious consequences. Theoretical work has shown that it is impossible to achieve distributed consensus in an asynchronous system with even one faulty process (Fischer et al. 1985). In summary, applications need to provide their own failure detection and recovery techniques (if required).

Patterns of Communication

The two general distributed systems architectures which have emerged on the Internet are peer-to-peer and client-server. In peer-to-peer systems any computer can play any role with respect to a protocol or application. For example, in a peer-to-peer file sharing system all participants may make their file systems available to all others. In a client-server relationship the server would typically export part of its file system, and the client would use it, but would not reciprocally export any of its files. Client-server models are well suited to the request/response style of communication used by many internet protocols. The client makes a request and the server responds. The Hypertext Transfer Protocol (http), for example, does exactly this.

2.3 Groupware

It has become increasingly important in modern society to have communications and information technology (C&IT) skills for all working environments. In particular,

C&IT has the potential for sharing information amongst groups working on some shared project or task. Computer supported collaborative work (CSCW) refers to methodologies that allow multiple users to collaborate on shared tasks. Groupware refers to tangible, usable applications that implement CSCW methodologies. DLEs offer a significant advantage over more traditional forms of computer-mediated learning through their support for group work and general sharing of information for management and educational purposes (Michaelson 1999, Michaelson et al. 2002). This type of activity has been investigated under the aegis of groupware for several years and it is important to exploit the knowledge that has already accrued in the development of this type of software. For example, role-based access is of interest in a DLE, where students and tutors have different types of access to the same resource.

A popular summation of possible groupware interaction modes with respect to time and synchronicity is illustrated in Fig.2.10. Same time/same place could be envisaged as a small group of people gathered round a computer or in a room with a data projector. Bulletin boards are an example of asynchronous interaction at the same (virtual) place at different times, while the distributed asynchronous variation is exemplified by e-mail.

	Same Time	Different Times
Same Place	face-to-face interaction	asynchronous interaction
Different Places	real-time distributed interaction	asynchronous distributed interaction

Figure 2.10: Groupware Interaction Modes

The type of groupware referred to in the lower left quadrant that specifically supports simultaneous interactions between geographically separated users is termed *realtime distributed groupware* (RDG), and typically exploits networked computers to support common group-oriented interfaces (Ellis et al. 1991). RDG is of special interest as its functionality subsumes that required by other modes of interaction, and *interoperability* between different interaction modes has been identified by Englebart as a key

groupware requirement (Engelbart 1990). Ellis and Gibbs (Ellis & Gibbs 1989) have listed some other requirements of RDG systems as follows:

- there is the possibility of a high degree of access conflict
- response times must be short
- participants are not required to be connected to the same machine
- participants are free to come and go at any time
- participants are not required to follow a pre-planned script
- participants can communicate with each other via an audio-visual channel

A further requirement must be added to the list above: *groupware that is to be effectively deployed must be web-friendly.*

2.3.1 Groupware Environments

In practice, the creation of a single piece of software that supports all necessary co-operative working requirements is unrealistic¹. Accordingly it is more useful to think in terms of environments, where multiple pieces of software, which we will call resources, can be co-located at a group-oriented interface for the convenience of a user. A user's groupware environment may consist of generic collaboration resources such as a scheduled video conference and a shared notebook, and various task-specific shared resources. The meaning of a "group" is central to this type of environment. Figure 2.11 shows an example environment where groups represent the mappings of resources to users, acting as a resource allocation mechanism.

¹ Stallman had a pretty good try with emacs, which ended up looking more like an operating system than a text editor.

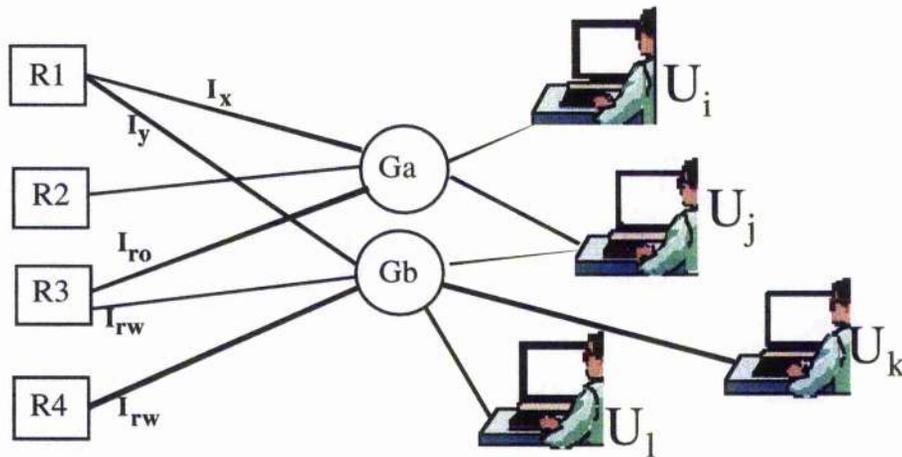


Figure 2.11: A Groupware Environment Architecture

- Resources R1-R4 are allocated to Groups Ga and Gb as shown: $Ga = \{R1, R2, R3\}$, $Gb = \{R1, R3, R4\}$.
- Users i-l are allocated to groups as shown: $U_i = \{Ga\}$, $U_j = \{Ga, Gb\}$, $U_k = \{Gb\}$, $U_l = \{Gb\}$.
- The groupware environment, $GEnv$, for each user is the sum of the resources allocated to each group the user belongs to: $GEnv(U_i) = Ga = \{R1, R2, R3\}$, $GEnv(U_j) = Ga \cup Gb = \{R1, R2, R3, R4\}$, $GEnv(U_k) = Gb = \{R1, R3, R4\}$, $GEnv(U_l) = Gb = \{R1, R3, R4\}$.

A resource may provide different types of operations, and these in turn can be allocated to different groups in the form of interfaces. In Fig. 2.11 R1 has two interfaces, I_x and I_y , with different semantics. These in turn are associated with Groups Ga and Gb, so the types of operation a user can perform on this resource instance will depend on their group memberships. R2 only has one type of interface. R3 exports both read-write and read-only interfaces. User U_j has access to both of these interfaces. R4 is perhaps symptomatic of the core issue in groupware: it exports a single read-write interface, and this is shared between multiple users. How is the consistency of R4 to be maintained in the face of multiple concurrent reads and writes?

2.3.2 Sharing, Concurrency and Coherence

The problem with allowing multiple users concurrent read-write access to a shared object is in maintaining the consistency of that object, or the views of it. The problem arises from the read-write-read pattern of access, or load and store semantics, present in most computer systems. This problem can be illustrated with reference to a simple example. In a banking system transfers of funds are expressed as debits and credits between accounts. Suppose an account A, with a current balance of £20, is the subject of two transfers, Op1 and Op2, that are initiated at approximately the same time:

```
Op1:    debit(A, 30)
Op2:    credit(A, 60)
```

Each transfer consists of a read, a computation and a write. The problem is that both could start by reading the same value, computing with it, and then writing the result back e.g.

```
Op1: local variable = read(A) = 20
Op2: local variable = read(A) = 20
Op1: local variable = account value - 30 = -10
Op2: local variable = account value + 60 = 80
Op1: A = write(A, local variable) = -10
Op2: A = write(A, local variable) = 80
```

The example schedule shown above is only one of the several that are possible. The final value of A could be £80, -£10, or £50, depending on the interleaving of Op1 and Op2. Numerous low-level methodologies and mechanisms have been proposed and implemented to avoid this type of problem. They include mutual exclusion, critical sections, semaphores, monitors, conditional critical regions. As these approaches deal with concurrency by effectively removing it, there is a danger that system throughput will suffer. A further drawback is that it is quite possible to create code that can deadlock or livelock in certain situations. Databases have developed higher-level abstractions which allow for domain-specific safe concurrent scheduling. The most common programming abstraction in the database community is the atomic transaction.

2.3.2.1 Atomic Transactions

Atomic transactions, atomic actions or simply transactions, refer to a programming model which has two basic statements, `transaction_begin` and `transaction_end`. Either everything in between these two delimiters is executed or none of it is. This all-or-nothing property, *atomicity*, is one of the so called four ACID properties (Weihl 1993) of transactions: Atomicity, Consistency, Isolation and Durability.

If Op1 and Op2 are transactions then the problem cannot arise because they are *isolated* from each other and cannot be incorrectly interleaved. In other words, the outcome would have to be the same as if the operations had been executed sequentially. This isolation property is also sometimes called *serialisability*. Note also the implicit parallelism afforded by serialisability: the system is free to run all transactions concurrently or in any order, provided the outcome is the same as some serial ordering. In this example it does not matter in which order the two transactions are run.

To take the example a little further, suppose both transfers were initiated by the same agent, who knows that if an account is in the red at any point then overdraft charges will be levied. A transactional approach would allow the grouping of multiple operations into a single *atomic* event:

```
txn_begin
    Op1:    debit( A, 30)
    Op2:    credit( A, 60)
txn_end.
```

As far as an external observer is concerned the balance of account A changes from £20 to £50 without any intervening steps and no overdraft charges can be levied.

Consistency means that a transaction moves a system from one consistent state to another and preserves system invariants. For example, if the account transfers are all internal to a particular bank then the sum total of funds in all accounts concerned should remain the same before and after any number of transactions.

Durability refers to the relative safety and permanence of the results of a *committed* transaction, which should withstand, for example a power failure or disk crash.

Checkpoint, Rollback, Commit & Abort

ACID properties demand quite sophisticated behind the scenes management. Briefly, at the start of each transaction a snapshot is taken of the current state of all the variables involved. This is called a *checkpoint* and is important because it can be returned to should a transaction fail to complete. The process of returning to a checkpoint gracefully is called *rollback*. A failure to complete is referred to as an *abort*. If a transaction does complete successfully then it is said to have *committed*.

Fault tolerance

Fault-tolerance is a highly desirable feature of any application executing on a distributed platform. One of the main problems encountered when programming distributed systems is that of *uncertainty*. For example a remote machine crash or network partition will dramatically change the appearance and behaviour of a distributed execution platform. Transactions can offer protection against such uncertainty via failure atomicity. If a transaction consists of many operations, then a machine crash at any point before commitment means that the results of all the operations executed prior to the crash are annulled.

2.3.2.2 Databases and Groupware

Concurrency control in groupware demands more cooperation than competition. Users are aware of each other and intend to cooperate, in contrast to the conventional database approach, which creates the feel of a private and isolated session. Perhaps for this reason, the groupware community has declared transactions as unsuitable mechanisms (Greenberg & Marwood 1994). This is discussed further in Chapter 3, as part of the investigation into the dOpt groupware algorithm.

Concurrency control in distributed systems can easily lead to problems such as *deadlock*, *livelock*, or *starvation*. Although the ACID properties of transactions can help in reducing the complexity of the task they do not in themselves offer guarantees against all these potential problems. The Warp coherence mechanism described in Chapter 4 does address these problems and exports a transactional application programming interface that augments the ACID properties with *liveness*, *fairness* and *awareness*.

Liveness means that provided a transaction consists of code that would run to completion within itself, then it will run to completion even when in continual conflict with other concurrent transactions. In a parallel execution environment it means that deadlock cannot occur and computational progress is guaranteed.

Fairness refers to the absence of starvation. Even in a live system it is possible for slower processes to be starved by quicker, more aggressive ones. Warp gives preference to the oldest transaction in cases of conflict. As transactions always complete and transaction ids are totally ordered every transaction will become the oldest at some point in its life and thereby acquire top priority in conflict situations.

Awareness means that participants in a groupware environment are aware of each others activities, and can therefore partition a project into tasks, and avoid or resolve conflict by communicating to each other.

2.3.3 Groupware Environments and QoS

In an Internet-based distributed groupware environment there are likely to be many different types of shared resource each with potentially very different sets of QoS requirements.

Internet-based Groupware Environment <i>multiple resources with different QoS requirements</i>			
shared resources, may be replicated or cached		peer-to-peer multicast media flows	shared resources, may be replicated or cached
Hyper Text Transfer Protocol (HTTP)	other application level protocols	Real Time Protocol (RTP) <i>a time-aware protocol</i>	other application level protocols
Transmission Control Protocol (TCP) <i>two-party reliable data exchange, no time guarantees</i>		User Datagram Protocol (UDP) <i>basis for IP multicast</i> <i>(a wrapper round IP that adds port numbers)</i>	
Internet Protocol (IP) <i>unreliable, best effort, no time guarantees</i>			

Figure 2.12: Protocols involved in QoS for Groupware Environments

Figure 2.12 illustrates the situation, highlighting the two main types of QoS requirements: reliability and timeliness. The application-level protocols used in supporting coherence may use either a reliable (TCP) or unreliable (UDP) transport protocol. IP multicast would seem to be a good basis for coherence protocols based on multicast messages and vector clocks (such the BSSP described earlier), and the protocols involved in delivering time-sensitive media streams will almost always use UDP. So, is IP multicast an appropriate basis for both reliability and timeliness?

2.3.3.1 IP Multicast as the Basis for Groupware

At a superficial level IP multicast would seem to provide a good basis for high-level group communication on the Internet. Indeed a UK wide project, PIPVIC (UKERNA 1999), which sought to use the Mbone tools in a variety of education settings is described in Chapter 3.

One of the main motivations behind the development of the Mbone was the need for a scaleable and efficient means of distributing real-time audio and video traffic. The tools and applications associated with the Mbone have accordingly been concerned primarily with continuous media (McCanne 1999), where timeliness of delivery is more important than data integrity.

There are however drawbacks when looking to the Mbone for *reliable* data dissemination. It must be augmented in some other way to ensure that data is copied bit-perfect, if reliability is required. Many networking protocols use ARQ for reliability. The fundamental problem with using this class of protocol in a large scale multicast setting is the “ACK implosion”. A single source can send to tens of thousands of receivers without problem, but if they respond with ACKs or NACKs, the aggregate return traffic will cause severe congestion as it converges on the source. A variety of reliable multicast transport protocols have been developed (Obrzcka 1998), such as RMTP (Lin & Paul 1996) and LRMP (Liao 1999), which seek to reduce the ACK implosion problem. These still only scale to a limited extent however, reflecting the inevitable trade-off between reliability and scalability when feedback channels from multiple receivers to each sender are used.

With respect to groupware environments, consider the case where all participants are connected by audio and video channels, and can edit a shared object at the same time. While the RTP protocol (Schulzrinne & al. 1996) is suitable for audio/visual traffic, a separate protocol must be used for reliable inter-object data transfer. In the groupware environment described in Chapter 4 an IP multicast-based version of RPC-2 (Satyanarayanan 1995) was used for reliable data transmission. In practice it is unlikely that this type of environment would need to support more than twenty concurrently active participants, in which case RPC-2 is quite adequate.

Deployment is an issue with DLEs, and so a further consideration is that IP multicast is by no means universally available. Although it has been around for over ten years it is still not a required part of IP, and ISPs do not often support it, thus ruling out its use for a significant part of the Internet user population. IPv6 (Bradner & Mankin 1995) makes multicast a standard part of IP, but IPv6 is not yet widely deployed.

Although IP multicast continues to have great potential, there is a growing recognition of the need for alternatives to the congestion-prone Internet for large-scale reliable data dissemination. Peer-to-peer architectures and wireless broadcast offer interesting alternatives.

2.3.4 Security

Security has become an increasingly important consideration in all networked computer systems over the last decade. The concept is fairly wide, and is being continually updated, in line with technological change and cultural norms. For example, phrases such as “digital privacy” are now used in the context of pervasive computing, and “trust” has acquired a fairly technical meaning, which complements its traditional moral imperative. This section focuses on the systems support issues involved in providing security in a distributed learning environment.

To some extent a DLE will subsume traditional educational management processes. For example, students might hand in work by hand to a secretary or tutor. That work will be treated confidentially by the marker and form part of the student’s assessed grade. In the online scenario it is therefore important that the *confidentiality* of a student’s work

and marks is maintained and that the *identity* of an individual tutor or student is authenticated. More generally, the integrity of any system hosting a DLE component should be protected.

What is meant by security in a distributed system? A distributed system consists of computer systems connected by networks. A secure distributed system consists of *secure computer systems* connected by *secure network connections*, or *channels*.

2.3.4.1 Properties of a Secure Computer System

Firstly, only authorised users should be able to logon to the system. This may mean passing several levels of physical security in a building, in order to gain access to a console, before being authenticated by the system itself. The means of system authentication such as password should be as difficult as possible for an impostor to guess. For this reason most modern operating systems insist on passwords consisting of strings that do not appear in convenient lists, such as dictionaries, or phone books.

Secondly, once authenticated, a user should only be able to access objects in the system according to their access privileges. The two broad approaches to this are capabilities (Dennis & Van Horn 1966) and access control lists (Lampson 1974).

Finally, an audit trail should be left for all user accesses, in order to unravel security breaches after the event. This has been an optional feature in many operating systems, sometimes not enabled because of its impact on performance.

Most of the work of enforcing security and protection in a computer systems is carried out by the operating system, and the methodologies and means employed are discussed extensively in the literature.

2.3.4.2 Properties of secure communication

The popular scenario used in the depiction of secure communication issues is one where two principles, conventionally named Alice and Bob, exchange messages which are subject to various types of undesired interference from a potentially malicious third party. Either Alice or Bob may be a computer system, or an end-user.

Confidentiality, Privacy.

- i) Only the sender and intended receiver should be able to understand the message. This is the popular notion of secure communication.
- ii) Only the sender and receiver should know of the existence of the communication between them. For example, the knowledge that messages have been exchanged between Alice and Bob at some particular point in time may be significant, without having to know what their contents were.

Authentication

This means that both the sender and the receiver are who they say they are. The identity of a sender, or receiver, is important as confidential information can easily be obtained by a third party if they can spoof identity.

Message Integrity

This means that the message arrives unmodified. It is important in that a small modification to the text of a message e.g. the addition of a few zeros to a quantity in a cash transaction, can have a major effect.

Non-repudiation

If Alice has sent a message to Bob, she cannot reasonably claim that she has not done so. This complements authentication, and is an issue in areas such as e-commerce where an e-retailer may be faced with a customer claiming that they did not place an order.

Efficiency of Security Mechanisms

As much of the focus in network security involves desktop computer systems it is pointless using cryptographic algorithms that will take hours to encrypt brief e-mail messages.

2.3.4.3 Cryptographic Building Blocks

Secure communication is implemented using cryptographic building blocks. Fundamentals include private and public key systems, hashing algorithms, and time.

Digital signatures, time-sensitive exchanges and certification authorities are examples of higher-level abstractions built from these fundamentals.

Confidentiality can be achieved by encrypting a message.

In a private key cryptosystem, sometimes called a symmetrical key, or private key, system, Alice encrypts a message using the secret key and algorithm, and sends it to Bob. Bob uses the same secret key to decrypt the message. If an intruder obtains a copy of the message they cannot make sense of it without knowing the secret key. In practice an intruder will eventually be able to “crack” it, but if that takes years and the message was only confidential for a few days, then the secure channel will have succeeded in providing confidentiality with regard to the message contents. It has not however provided confidentiality with regard to the existence of the message, nor has it guaranteed that the sender was Alice and the receiver was Bob. There is also still the possibility that the message was tampered with, (without the intruder being able to make sense of it).

There are two notable drawbacks with the private key approach: i) how to securely agree on a secret key in the first instance (before there is a secure channel); and ii) it means that each user must have a shared secret with each possible correspondent – a potentially large and unmanageable set of keys.

These two problems are addressed by public key cryptosystems, which are asymmetric. A pair of keys is generated, one for encryption and one for decryption. One of these is made public. For example, Alice encrypts a message for Bob using Bob’s public key, and transmits it. This can only be decrypted using the (private) decryption key. Only Bob should have a copy of the decryption key. Bob receives the message and decrypts it.

Note that there is no initial key exchange problem, and that users only need a single pair of keys. There are still (at least) two problems: i) public keys algorithms tend to be too compute-intensive to use routinely on desktop computers (the sizes of the keys are > 1024 bits); and ii) what guarantee is there that Bob is actually Bob?

The latter question is dealt with by the notion of a public key infrastructure (PKI). In a PKI a trusted authority, similar to a passport office or register of births and deaths, signs Bobs key with a digital signature. A suspicious correspondent must now mistrust the certification authority in general if they don't believe Bob's credentials. The digital signature is a hash digest of Bobs public key.

A hash function H has the following properties:

- when H is applied to message m , it produces a fixed size message digest, $x = H(m)$
- given message digest x and function H , it is computationally infeasible to find m from $x = H(m)$
- it is computationally infeasible to find any two messages m and m' such that $H(m) = H(m')$.

Digital signatures are also used for e-mail, where provenance is important. Assuming it is beyond the means of Alice's computer to encrypt a long message to Bob using Bob's public key, but that it is important that Bob knows who it is really from and it has not been tampered with, Alice generates a (short) hash digest of the message, encrypts that with her private key, and appends it to the message when sent to Bob. Bob generates a hash digest, decrypts the one that was sent (using Alice's public key), and compares the two, thereby establishing that the message is indeed from Alice and that the message has not been tampered with.

A more general approach to efficient secure transmission of messages is to use the public key system for initial exchange of private keys, which will only be used once, and then to use the private keys for efficient encryption and decryption.

Finally, the PKI approach has yet to be widely accepted. A serious barrier in the UK is the lack of a clear state endorsement of a certification authority. What is happening in practice is that several private PKIs are in use, restricted in scope to institutional or sectoral domains.

2.3.4.4 Internet Security Protocols

Deployment is an issue, and the basis for value-added Internet services such as secure communication. Secure channels can be implemented at any level in the Internet layered model. Figure 2.13 shows the levels at which four currently deployed Internet security protocols operate at. Secure Shell (Security 2001) operates at the application level, the Secure Socket Layer (Freier et al. 1996) operates on the boundary between the application-level and the transport-level, and IpSec (Thayer et al. 1998) operates at the IP network level.

<i>Security Protocol</i>	<i>Layer</i>	<i>Features</i>
Pretty Good Privacy (PGP)	Application	Totally decentralised public key approach. Users attend "signing parties" where they can verify each other directly. "Web of trust". Highly unpopular with governments as it is seen as a loss of state control over the transfer of secret information.
Secure Shell (Ssh)	Application	Replacement for protocols such as telnet and ftp; also offers tunneling for other application-level protocols.
Secure Sockets Layer (SSL)	Transport	Replacement for conventional sockets; Used by the secure http protocol for web transactions.
IpSec	Network	Used for all IP traffic. This means that transport header information is also encrypted.

Figure 2.13: Some Internet security protocols

There is no reason not to use multiple layers of security (other than processing cost and time delay). For example, a user could use PGP to encrypt a mail message m , giving m' , their system could then transmit it via a secure shell tunnel (m'') the secure shell tunnel could run over the secure sockets layer, (so the message becomes m''') and finally, IPsec could then be used to transmit the message over the Internet. At each stage an initial key exchange will have taken place prior to encryption.

2.3.4.5 Security is much more than a technical problem

Security measures are at the best probabilistic, and at the end of the day, no amount of technical safeguards can protect against subversive or insecure behaviour by privileged personnel. Lampson has observed that the idea of a computer system which is attached to a network being secure is “pure fantasy” (Lampson 1998). This rather wry comment is based on the serious problem that much of the utility of distributed systems comes from their extent and openness, which are exactly the features which make them vulnerable to a wide range of intruders.

2.4 Summary

The three major themes of Coherence, QoS and Framework have been identified as important systems areas for supporting DLEs. These in turn draw from established fields of study in Distributed Systems, Computer Networks and Groupware. The topics within these fields are listed in Table 2.5, with a summary of their relevance to DLEs, and where they are encountered elsewhere in this thesis.

Concept	Thesis chapter where concept is used	Relevance to Distributed Learning (examples)
Internet Protocol Semantics	Chapters 6,7	The Internet is the target platform for distributed learning. Its features and problems must be taken into account in the design and implementation of online environments.
QoS on the Internet	Chapters 3,6,7	Bandwidth, Timeliness, Jitter and Reliability are all important for student-resource, student-student, tutor-student interaction and tutor-resource interaction.
Group Communication at the Network Level	Chapters 3,4,7	This is potentially powerful basis for group communication at higher levels. It is particularly relevant to video conferencing.
Event Ordering in Distributed Systems	Chapters 3,4,7	This is important for supporting coherence in shared object interaction and also in the maintenance of replicated resources.
Group Communication at the Application Level	Chapters 3,4,5,7	Group-based learning is strongly endorsed by educationalists and must be well supported. Teamwork involves shared objects and tasks, and intra-group communication. Concurrent access must not result in unintentional outcomes. In the distributed scenario local concurrency control must be subsumed by distributed coherence.
Groupware Environments	Chapters 3,4,7	A framework is necessary for organising users into their roles and groups, and allocating them educational resources appropriate to their course of study.
Security	Chapters 3,5	The identity of an individual tutor or student must be authenticated for educational processes; The security of a student's work online must be enforced as far as is possible.

Table 2.5: A List of systems concepts used in this thesis, and how they relate to distributed learning

3 Related Work

This Chapter reviews some other research projects which have pursued systems issues associated with DLEs. These are:

- Coherence in Groupware Systems – dOPT
- Coherence in Distributed Systems – TimeWarp
- A Groupware Framework for Web-based Shared Workspaces – BSCW (Basic Support for Co-operative Work using the Web)
- IP Multicast-Based Groupware – PIPVIC: Mbone tools in educational contexts
- Internet2:
 - Quality of Service: the Qbone
 - Distributed Access Control for Educational Resources: Shibboleth

3.1 Background

The original problem area under research was coherence in groupware systems. To this end approaches such as Birman's ISIS toolkit for reliable distributed processing (Birman & van Renesse 1993), Liskov's Argus transaction-based programming language (Liskov 1988) and Ellis and Gibb's dOPT algorithm (Ellis & Gibbs 1989) were investigated. These systems all offered support for coherence in the face of multiple concurrent readers and writers. Only dOPT however also explicitly attempted to meet the *user's requirement* for interactive responsiveness. dOPT is a groupware system in which the tension between responsiveness and coherence is explicitly addressed by use of virtual time in the form of vector clocks, and a "distributed OPeraTion" algorithm. The general issue of the roles of serialisability and transactions in groupware are also discussed here as the authors of dOPT, in keeping with other groupware luminaries, contend that atomicity and serialisability are not suitable for groupware. A novel application in the form of a multi-user shared spreadsheet was implemented to exercise dOPT, and although it

showed the dOPT algorithm to be flawed in operation, it proved useful as it showed the potential of that type of multi-user application.

Timewarp (Jefferson 1985) provides an alternative approach to supporting coherence. It builds on Lamport's notion of logical time (Lamport 1978), but adds the twist that time can run backwards as well as forwards, and that messages already sent can be cancelled by the use of *anti-messages*. It was originally used for distributed discrete event simulation on the CalTech hypercube and was then expanded to an operating system, TWOS (Presley et al. 1989), which aimed to provide coherence for any type of distributed application. It is relevant in general because of the use of logical time to provide coherence in distributed systems, but also more specifically as Chapter 4 describes Twarp, a Timewarp programmers toolkit. The lessons learned from Twarp were used to inform the design of the Warp coherence mechanism. A multi-user shared spreadsheet implemented using Warp successfully demonstrated coherence and responsiveness in distributed groupware. Interestingly, there is still no other successful implementation of this type of application that I am aware of, commercial or otherwise, with the capabilities of the one described later in the thesis.

As work was completing on the Warp-based spreadsheet the Higher Education sector in Scotland began to see the benefits of fast wide area networking based on ATM technology. Shared spreadsheet sessions augmented by audio and visual channels were carried out between Dundee and St Andrews on dedicated ATM virtual circuits. These sessions were used to develop a multi-user real time business game for management students, using a customised version of the Warp spreadsheet. This brought into play the educational dimension in real time distributed groupware. Many difficulties were encountered in using this technology for routine lab-based teaching and learning. Perhaps the single most important lesson learned from the use of the shared spreadsheet for teaching and learning was that *deployment is an issue*. Certain assumptions and simplifications made in building systems primarily for research purposes, such as the Warp spreadsheet, meant that they could not readily be used in a routine computing

environment, and quite possibly not in any context which did not offer “laboratory conditions”.

At the same time as the fast ATM networks were beginning to open up new possibilities for networked learning environments the World Wide Web had matured in the form of the Mosaic browser and rapidly grew to become the single most heavily used information sharing application in the history of computing. The deployment problem was “solved”. The web became the de facto target platform for all implementations. Teaching and learning resources had to be web-friendly. One benefit was that web-based applications developed in a research setting could be much more readily deployed in an educational environment.

3.1.1 BSCW: Web-based groupware

The impact of the Web on the way we think about distributed systems and groupware cannot be overstated. BSCW (Bentley et al. 1997) was an early and popular exploitation of the web architecture for groupware purposes. It is a thin-client, heavy-server approach, relying on server-side processing and HTML to provide the user interface. The central, and indeed only, metaphor used in BSCW is the *shared workspace*, which is effectively a document repository. The notion of collaborative authoring in BSCW is to lock a document, take a copy of it, edit it, and put it back. This has less sophisticated notions of coherence and coarseness of granularity than those found in dOPT and TimeWarp, but on the other hand has been easy to deploy due to its strong congruence with standard web technologies. It also shows some of the limitations of the web. In terms of the groupware space/time matrix shown in Fig.2.14 web-based distributed groupware can be neither realtime nor synchronous. This has fostered an approach to teaching which is very content-oriented, so the selection, organisation and presentation of content becomes an end in itself, rather than a component in the learning process. Following on from this is a belief that learning can be effected by the production and absorption of byte-sized chunks of information, that these can be prepared by experts, and then be made available for global consumption.

3.1.2 PIPVIC: IP Multicast for Educational Applications

Many conventional approaches to groupware (dOPT, BSCW) regard audio and video channels as totally separate systems, highly desirable but disjoint from the core system which is concerned with the integrity of shared resources. The Mbone tools turn this approach upside down: all groupware is based on real-time IP multicasting. The use of the Mbone tools (Jacobson et al. 1995) (McCanne 1999) (see Table 3.1) for educational purposes was piloted in the UK for a ten month period beginning in January 1999 under the aegis of the PIPVIC project (UKERNA 1999). The primary focii of these pilots were to i) research the use of desktop audio and video conferencing in educational applications, ii) make network QoS measurements of these applications, and iii) evaluate of participants' perceptions of QoS. Although Mbone audio and video tools are not comparable to conventional groupware systems, tools such as the "NTE" Network Text Editor (Handley & Crowcroft 1997) and "WBD" Whiteboard (Highfield & Hasler 1999) are. These differ fundamentally from conventional groupware systems by showing no concern for coherence.

3.1.3 Internet2: QoS

The current Internet has no notion of "quality of service" (QoS). The Internet-2 project (IP2 1999) includes the provision of QoS in its key goals. The Internet Engineering Task Force has developed two models for supporting different types of service, Integrated Services (IntServ) and Differentiated Services (DiffServ). The Internet2 Qbone (Teitlebaum et al. 1999) has adopted the IETF's DiffServ model for piloting at least one other class of service for Internet traffic, the Qbone Premium Class, which aims to have minimal delay, no loss and no jitter.

However, to what extent is the network the problem? Internet2 networks such as Abilene and vBNS already run at 2.4Gb/s or higher bandwidths, which, as they are shielded from the normal exigencies of the commercial Internet, is good enough quality for "advanced applications". So, one of the main challenges is for the server processes and the client system software to keep up, and match the network performance. Simply

delivering a stream with high network QoS to an existing desktop computer is not enough in itself. Throwing more network quality at certain applications does not improve the end-user's experience.

3.1.4 Internet2: Security Infrastructure

In a distributed learning environment there are many different roles each with their own requirements for privacy and confidentiality, so the traditional concerns with security in distributed systems are very relevant. These issues include authentication and access control. For example, if user Alice is already a registered and a trusted user with Institution I, but requires access to remote educational resources held at Institutions R and S, how can this be satisfied in a secure manner that satisfies all stakeholders' interests? This is a key question for the Internet2 middleware programme¹ and has been partly addressed by the Middleware Architecture Committee for Education working group through their development of Shibboleth, which is an implementation model for web-based resource sharing and access control across administration domains.

3.2 dOPT: Coherence in Groupware Systems

dOpT (the distributed operation transformation) is originally and fully described in (Ellis & Gibbs 1989). The algorithm attempts to meet most of the key requirements of real-time distributed groupware. It is particularly concerned with minimising interactive response times for users (who may be on opposite sides of the world) while maintaining shared document integrity. dOPT based applications include Grove (Ellis et al. 1988), a shared document outliner, and CoEd (Holtz 1992), a shared text editing system. The two mechanisms used by dOPT are vector clocks, (see Chapter 2) and an operation transform table, where combinations of actions originating at different sites asynchronously are resolved according to the rules in the table.

¹ <http://middleware.internet2.edu>

3.2.1 The dOPT groupware model

A groupware system is modelled as a set of sites with unique ids and a set of parameterised operations. Operations are application specific: $O = \langle O_1, O_2, \dots, O_n \rangle$. For example, a text edit operation may take the form $O_3 = \text{delete}(n)$ where n is a character or word offset. A shared object is replicated at each site and may be written or read arbitrarily by participants. Each site runs three processes: generate and queue operations, receive and queue operations, execute operations. After an operation has been executed it is logged. Each site maintains its own log. If an incoming request is identified from its *state vector* as being potentially in conflict with a local operation that has already executed, then it is compared against selected log entries, and if necessary transformed.

3.2.1.1 Communications, Event Ordering and State Vectors

Inter-site communication assumes a reliable message handling service. No global clock is assumed and an event-ordering scheme based on logical time (Lamport 1978) is used. Logical clocks are structured as state vectors. In an N site system each site maintains a state vector s of size N by incrementing the i th component after execution of an operation from site i .

The following relations are defined for the state vectors:

$s_i = s_j$ if all the components have the same value as their counterparts

$s_i < s_j$ if each component of s_i is less than or equal to its counterpart in s_j , and at least one component in s_i is clearly less than its counterpart in s_j

(This relation is used to enforce precedence: if an operation is received at site i tagged with s_j and $s_i < s_j$ then it is not executed until $s_j \leq s_i$).

$s_i > s_j$ if at least one of the components of s_i is greater than its counterpart in s_j .

3.1.1.2 Operation requests

Each operation request is a tuple $\langle i, s_i, o, p \rangle$ containing a copy of the originators site id, its state vector, the specific operation and the *priority* of the operation. The priority can simply be the site id and is used for tie breaking in the case of two identical

operations with equivalent state vectors. It could be calculated on a fairer basis if necessary.

3.1.1.3 The transformation matrix T

T contains application specific rules which handle concurrent operations in such a manner that the effect of applying operation O_1 then operation O_2 at site i is the same as applying operation O_2 then operation O_1 at site j . A simple example: the rule dealing with the arrival of an insert operation "B:Ins x_2, y_2 val₂" on a spreadsheet cell " x, y " where insert operation "A:Ins x_1, y_1 val₁" is in the log:

```
if (A:sender < B:sender)
    return "B:Ins  $x_2, y_2$ " no change
else
    return "no-op" transformed
```

3.1.1.4 The core algorithm

Data structures:

Q_i the queue of operation requests

L_i the log of operations which have been executed

$\langle i, s, o, p \rangle$ the form of a request: i is the site id, s is the state vector, o is the operation and p is the priority.

Initialisation:

$Q_i \leftarrow$ empty the queue of operation requests

$L_i \leftarrow$ empty the log of executed operations

$s_i \leftarrow \langle 0, 0, \dots, 0 \rangle$ the state vector of site i

Generate Operations:

receive operation o from the user interface

calculate the priority p of o

$Q_i \leftarrow Q_i + \langle i, s_i, o_i, p_i \rangle$

broadcast $\langle i, s_i, o_i, p_i \rangle$ to all other sites

Receive Operations:

receive $\langle j, s_j, o_j, p_j \rangle$ from the network

$Q_i \leftarrow Q_i + \langle j, s_j, o_j, p_j \rangle$

Execute Operations:

for each $\langle j, s_j, o_j, p_j \rangle \in Q_i$ where $s_j \leq s_i$ begin

$Q_i \leftarrow Q_i - \langle j, s_j, o_j, p_j \rangle$

if $s_j < s_i$

$\langle k, s_k, o_k, p_k \rangle \leftarrow$ most recent entry from L_i where $s_k \leq s_j$ (\emptyset if none)

do while $\langle k, s_k, o_k, p_k \rangle \neq \emptyset$ and $o_j \neq \emptyset$

if the k 'th component of s_j is \leq the k 'th component of s_k

let u be the index of o_j

let v be the index of o_k

$o_j \leftarrow T_{uv}(o_j, o_k, p_j, p_k)$

fi

$\langle k, s_k, o_k, p_k \rangle \leftarrow$ next entry in L_i (or \emptyset if none)

od

fi

perform operation o_j on i 's site object

$L_i \leftarrow L_i + \langle j, s_i, o_j, p_j \rangle$

$s_i \leftarrow s_i$ with j th component incremented by 1

end

3.1.1.5 The partial concurrency problem in dOPT

In order to gain practical experience with dOPT a multi-user shared spreadsheet was implemented, which ran on a network of workstations. Progress at each site was recorded in an independent log (not L_i) and was used for post-mortem analysis. The spreadsheet initially appeared to work well, resolving occasional clashes correctly. Eventually however, the partial concurrency problem became manifest. Figure 3.2 illustrates a recorded instance of the problem in a time/event diagram where there are conflicting operations on spreadsheet cell C3.

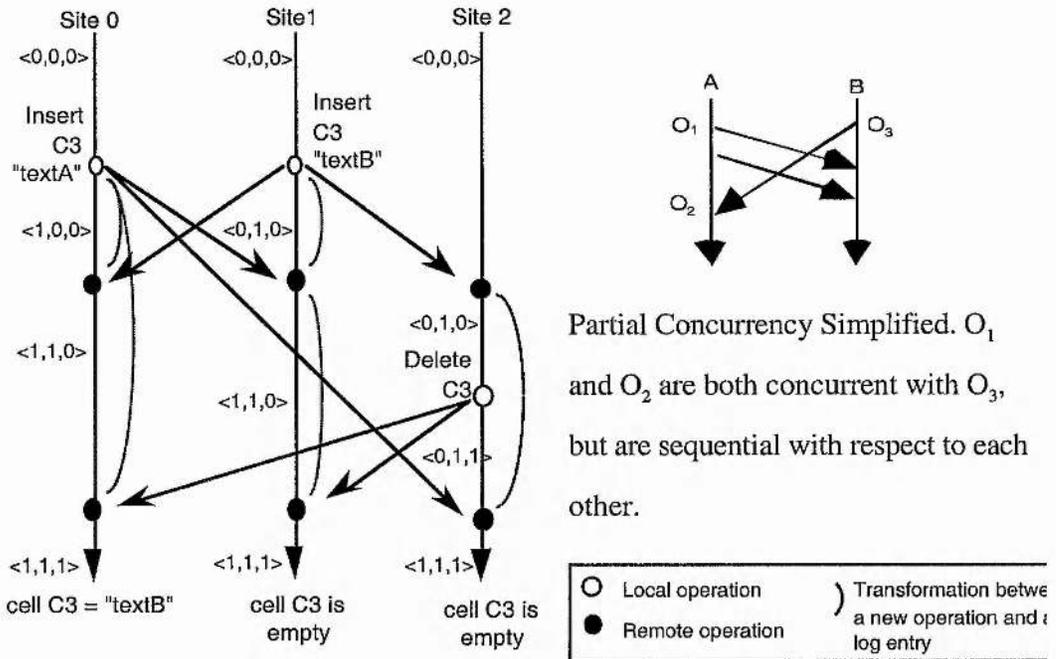


Figure 3.2: Partial Concurrency Example in a dOPT System

Site 0 inserts the string "textA" into cell C3 and increments its state vector. The broadcast message has state vector $\langle 0,0,0 \rangle$. It then receives the request $[1, \text{Insert C3 "textB"}, \langle 0,0,0 \rangle, 1]$ from site 1. $\langle 0,0,0 \rangle$ is less than the local state vector so the request is dOPTed and due to the priority $1 > 0$ (simple site id priority) the new string is inserted. Finally the delete request from site 2 arrives with $\langle 0,1,0 \rangle$ and is not transformed with respect to the most recent insert because its state vector is greater than that in the log entry (site 2 saw site 1's insertion before deleting it). It is then transformed against the initial insert to a no-op (insert is preferred to delete) and the insert from site 1 ("textB") remains.

Site 1 inserts the string "textB" into cell C3 and increments its state vector. The broadcast message has $\langle 0,0,0 \rangle$. It then receives the request $[0, \text{Insert C3 "textA"}, \langle 0,0,0 \rangle, 0]$ from site 0. $\langle 0,0,0 \rangle$ is less than the local state and must be dOPTed. According to site priority the request from site 0 is transformed into a no-op. Finally a delete request arrives from site 2 with state $\langle 0,1,0 \rangle$ which is less than the local state. This operation is dOPTed, but is not transformed against the most recent log entry because it is a no-op. In addition it is not transformed with respect to the oldest log entry

because its state vector is greater than that in the log entry (site 2 saw site 1's insertion before deleting it). The delete operation is carried out and the final result is an empty cell.

Site 2 receives an insert from site 1, increments its state vector, deletes the recent insert and broadcasts the delete operation with state $\langle 0,1,0 \rangle$. It then receives an insert from site 0 with clock $\langle 0,0,0 \rangle$ which is not transformed against the delete (and wins due to insert precedence). It is then transformed against the oldest log entry and due to site priority is transformed to a no-op. The final result is an empty cell.

Partial concurrency refers to any set of operations where at least two are sequential to each other but concurrent with a third. The problems in Figure 3.2 stem from the first two events at Site 2 being sequential with each other, but both being concurrent with the first event at Site 0. A simpler example is shown in the right hand side of Figure 3.2. The operations O_1 and O_2 are sequential with each other and concurrent with O_3 . Depending on which types of operations are involved the outcome from dOPT may or may not appear consistent. It may be possible to adapt the transformation rules to fix the problem for a particular set of operations but such a solution is lacking in generality, and is tantamount to sweeping the problem under the carpet as the application programmer must then grapple with coherence for each specific case.

3.1.2 Interpreting dOPT

Although the dOPT paper raises issues which are central to groupware implementation it also contains some ambiguities, and it is clear that at the time of writing the authors were not completely satisfied with the correctness of their algorithm. This section describes a resolution of some dOPT ambiguities.

Use of the empty set symbol " \emptyset " is not explained and is overloaded. When being compared with a request tuple "do while $\langle k, s, o, p \rangle \neq \emptyset$ " it appears to mean end-of-list. When used in comparison with a specific operation " $o_j \neq \emptyset$ " we assume it means "no-op". This seems a reasonable interpretation as a transformation can usefully return a no-op to invalidate an operation involved in a conflict, and when $o_j = \emptyset$ the search for

resolution stops. Part of the guard on the while loop however reads " $o_j \neq \emptyset$ " which implies that the transformation matrix should accommodate the possibility of a transformation of a new request against a log entry where $o_k = \emptyset$. Rather than extending T to hold $(O+1)^2$ rules we assume that the guard effectively means $o_k \neq \emptyset$ on the first pass, this being consistent with a no-op constituting a termination condition rather than a source of further transformation.

The use of the greater-than symbol ">" in the comparison of state vectors is at best confusing: " $s_i > s_j$ if at least one of the components of s_i is greater than its counterpart in s_j ". If the two vectors $\langle 0,1,0 \rangle$ and $\langle 1,0,0 \rangle$ are compared then they are both greater than each other. However, $\langle 1,0,0 \rangle > \langle 0,0,0 \rangle$ also holds. This is the negation of \leq and is not the definition of concurrency used in other work, that is $\neg(s_i \leq s_j) \wedge \neg(s_j \leq s_i)$. Although not explicitly used in the algorithm this does present the implementor with an unpleasant ambiguity regarding the intended meanings of comparisons between state vectors.

Relations defined on state vectors are "<", ">" and "=", but it is quite possible for two state vectors to meet none of these conditions when compared. Consider the two vectors $\langle 0,1,0 \rangle$ and $\langle 1,0,0 \rangle$. We resolve this idiosyncrasy by leaving a operation request, transformed or not, with its original state vector when it is logged. Thus instead of " $L_j \leftarrow L_i + \langle j, s_i, o_j, p_j \rangle$ " we use: " $L_j \leftarrow L_i + \langle j, s_j, o_j, p_j \rangle$ ".

3.1.3 Discussion: Serialisability in Groupware

Serialisability is an accepted means of maintaining consistency in the face of multiple concurrent modifications to an object. Although normally a well understood concept, it has been repeatedly rejected by leading figures in the groupware field, including the dOPT authors, as being an unsuitable synchronisation method. That rejection is refuted here.

Greenberg and Marwood (Greenberg & Marwood 1994) present two scenarios as serialisability counter-examples. In the first, two users of a shared drawing package get into a tug-of-war over the position of a circle. Such a scenario can certainly arise, but Greenberg and Marwood attribute it to the system's attempts to serialise the moves of the two users. But if this were the case, then *removing* serialisation should solve the problem, or at least alleviate it. But of course, it does not – it could even exacerbate it, by allowing the circle to adopt an arbitrary sequence of intermediate positions. In fact, serialisation is orthogonal to the problem, which arises through lack of co-operation at the user level.

Greenberg and Marwood's second scenario concerns simultaneous update of a sentence in a shared editor. User Saul attempts to delete the sentence, say: "Now is the time for all men to come to the aid of the party", while user Dave attempts to insert the word "good" before "men". However, three actions get serialised:

- the delete;
- typing "go"; and
- typing "od".

If these are the atomic actions that the system identifies, then it is hardly surprising that serialising them can lead to undesirable behaviour. In the example, they are serialised in the order (2), (1), (3), leaving the state of the sentence as "od". There are two distinct problems here. The first is that the non-existent word "od" can appear in a text; the second is that any insertion at all is allowed to take place in an already deleted sentence. As regards the first problem, *not* serialising the actions can only make matters worse, by allowing even finer interleaving and therefore a greater variety of unwanted effects. The problem is not a consequence of serialisation – it is logically prior to serialisation and results from an inappropriate choice of temporal granularity – the subdivision of an operation into the atomic actions that get serialised. In the example, there should be only two such actions: the delete, and typing "good".

In contrast, the second problem does directly concern serialisability. A variant of it is also described in the dOPT paper by Ellis and Gibbs. In their scenario, two users each

simultaneously issue serialisable transactions to "delete the third letter of a shared text string *S*". Here the unwanted effect is the clearly unintended deletion of the third and fourth letters of *S*. This apparent paradox arises not from *forcing* serialisation, but on the contrary from a *failure* of serialisation. This is because (in the context of the Ellis and Gibbs version) the second delete is actually not atomic – the state in which it is performed is different from that in which it was perceived to be necessary. The non-atomicity is caused by a misrepresentation of the semantic structure of the transactions involved. They are not of the form "delete the third letter of the *S* value read by the transaction", but "delete the third letter of *this S* value which I (the user) am looking at as I launch the transaction". We call such a transaction *dependent*, and it should do nothing to any other value of *S*.

The real source of the paradox is attempting to use pre-programmed transactions in an interactive system. The Warp solution, which is described in Chapter 4 is to provide *interactive atoms* which encapsulate not only the actions on the data but also the user's decisions and choices about those actions. In the example under discussion one of the deletions will be unsuccessful and get backtracked. Its user will see that the new situation renders the delete obsolete, and can proceed accordingly along a different course.

3.1.1.1 Pessimism vs. optimism

When Kung & Robinson (Kung & Robinson 1981) described their original optimistic transaction mechanism, they claimed that it would outperform locking under low conflict. This notion became folklore, but Agrawal (Agrawal & Dewitt 1985) has shown it to be largely a myth. Nevertheless, even if the choice between pessimism and optimism has no performance implications, it certainly has an impact on the ergonomics of a groupware system. Broadly, a pessimistic mechanism will give good progress but poor immediacy, because it must wait for every lock. For example, the Argus system (Liskov 1988) integrated database techniques into a programming language intended for use in application construction but an attempt to build a shared document editor, CES, frustrated researchers: "...in our initial experiments with the editor, response time was

slow enough for the editor to be unusable in realistic testing...."(Grief et al. 1986). Conversely, an optimistic mechanism will give poor progress (because uncommittable actions are allowed) but good immediacy. In an interactive system, particularly one liable to generate long transactions, it can be counter-productive if optimism frequently proves unfounded and large amounts of work are invalidated through conflict. The choice is a compromise. Warp, described in Chapter 4, attempts to facilitate informed optimism in groupware.

3.3 TimeWarp: Coherence in Distributed Systems

Coherence requires some form of distributed concurrency control. Concurrency control mechanisms can broadly be classified as pessimistic (using locking and blocking) and optimistic (using runahead and, if necessary, rollback). In a distributed system, where the global state is spread around several nodes, the problem arises of how to *undo* communications to other nodes which may have resulted in a change to their local states, contributing towards a change in the global state. In a centralised shared memory system this is less of a problem as an optimistic computation can work on a private copy of some data and be prepared to abandon its work if rollback is necessary. The particular problem in the distributed scenario is what to do about messages which have been sent, and whose very existence forms part of the process history at each node, and also the actual global state. TimeWarp (Jefferson 1985) solved this problem through the novel use of *anti-messages* and reversible virtual time.

Each process in a TimeWarp system executes without regard to whether there are synchronisation conflicts with other processes. Once a conflict is discovered the offending process is rolled back to some time before the conflict, from where it continues to execute along a revised path. Ideally, the detection of the synchronisation conflicts and the use of the rollback mechanism for resolving them should be transparent to the application. Jefferson envisaged Time Warp being employed in distributed discrete event simulation and distributed database applications. Most implementations to date have consisted of restricted special purpose versions designed for very specific

simulations. Exceptions include the Time Warp Operating System (Presley et al. 1989) and Twarp, described in Chapter 4.

TWOS was implemented by a team lead by Jefferson on a distributed memory multiprocessor, the CalTec hypercube. TWOS imposed significant restrictions on the structure of application processes. It included an alternative non-virtual time based messaging system which could be used to bypass the core rollback mechanism, and it made no provision for autonomous processes², except by the rather inconvenient and artificial device of having a process keep explicitly sending itself rescheduling messages.

3.3.1 Virtual Time

In a distributed simulation, virtual time is typically the simulated time. In a TimeWarp system each process advances along its own independent virtual time axis, called local virtual time (LVT), incrementing its local virtual clock. Any incoming messages must be handled in LVT order. However, the independence of the various LVTs means that messages will certainly not arrive in order of increasing LVT. Furthermore, a process cannot know at any point what the next highest incoming message timestamp will be. It is therefore inevitable that processes sometimes progress too eagerly (run ahead) and will be required to rollback. Specifically, rollback will be caused by the arrival of a message whose timestamp pre-dates the process's LVT. The process must be rolled-back to the earlier virtual time. This necessitates the storage of a process's earlier local states.

3.3.2 Local Virtual Time and Rollback

When a process rolls back, all its work since the rollback point is obsolete, including all the messages it has sent to other processes. The recipients of these messages may also have run ahead; if so they must be rolled back in turn. Rollback must therefore be propagated throughout the entire system. The basic device used by Time Warp to

² An autonomous process in this context refers to the case where a process does not receive messages from other processes.

impose the virtual time order is for every process to queue its incoming messages in their timestamp order, rather than their arrival order. In this way, a process can be thought of as working along its input, increasing its LVT to the timestamp of each message as it gets to it. When a message arrives whose time stamp is smaller than the process's LVT, the message lands in that part of the queue already processed, thereby causing the rollback (see Fig. 3.1).

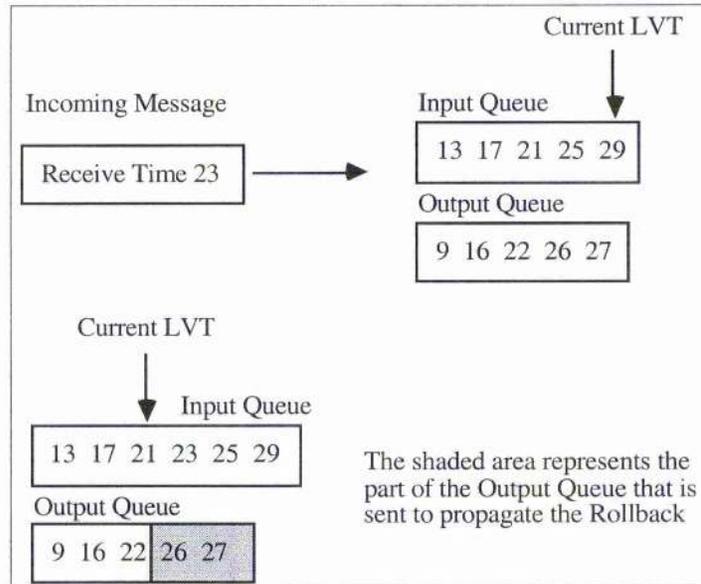


Figure 3.1: Propagating rollback

Time Warp achieves distributed rollback by *unsending* messages. Each normal message has a corresponding *anti-message* which, when sent to the same destination as the original positive message serves to annul it. The anti-message must carry the same timestamp as its positive and its arrival invalidates any work performed by the recipient from that timestamp, forcing the recipient to rollback to that point. When the recipient rolls back in response to the anti-message, it may need to send anti-messages to other processes. Thus rollback is propagated through the system. One consequence of using anti-messages is that a process must remember all its output messages in case they need to be unsent. To achieve this, each process also holds an output queue of the messages it sends, in their virtual *send-time* order. Should rollback occur, the messages in the output queue, with a send-time *higher* than the rollback time, are unsent. The queue can then be truncated at the rollback point.

3.3.3 Global Virtual Time and Commitment

Interactions with the outside world cannot be rolled back - it is difficult to unlaunch a missile or recall a travel ticket³. Such interactions cannot be committed until it is known that they can never again be subject to rollback. This corresponds to the system having made global progress relative to the outside world. Some means of assessing such global progress is therefore needed, to provide control over commitment. TimeWarp solves the problem of commitment via the notion of Global Virtual Time (GVT). A virtual time is *committed* if no process can be rolled back to or before it. GVT is the supremum of all the committed times; when an external interaction has a time stamp less than GVT it can be committed. GVT is defined as being less than or equal to the minimum of (a) all virtual times in all virtual clocks, and (b) all virtual send times of messages in input queues that have never been processed.

In summary, GVT is necessary to:

- (i) "Fossil collect". Any checkpointed states that have a timestamp less than GVT are fossils and may be deleted. This can take place at any time and must take place when the storage space is full.
- (ii) Allow irrevocable events to be processed. Whenever a server encounters an irrevocable event it must wait until the GVT has passed a certain time before it can perform that event.

Unfortunately the use and management of GVT turns out to be an Achilles heel in the suitability of TimeWarp as a general purpose coherence mechanism. This was learned through experimentation with the Twarp implementation that is described in Chapter 4.

³ Although many airlines have small print that allow them to cancel flights for which they have issued tickets!

3.4 BSCW: A Framework for Web-Based Groupware Environments

Basic Support for Co-operative Work on the world wide web was an early attempt to exploit the web for groupware purposes (Bentley et al. 1997). It has also been used for a variety of purposes in collaborative educational networked environments (Sikkel et al. 2001) (Klößner 2000) (Appelt & Mambrey 1999). It is described here as an exemplar of web-based groupware. At the core of BSCW is the concept of a *shared workspace*. A shared workspace is effectively a document repository. Documents can have a predefined type: article, image, note, or URL. The user interface is shown in Figure 3.3, which is copied from (Klößner 2000).

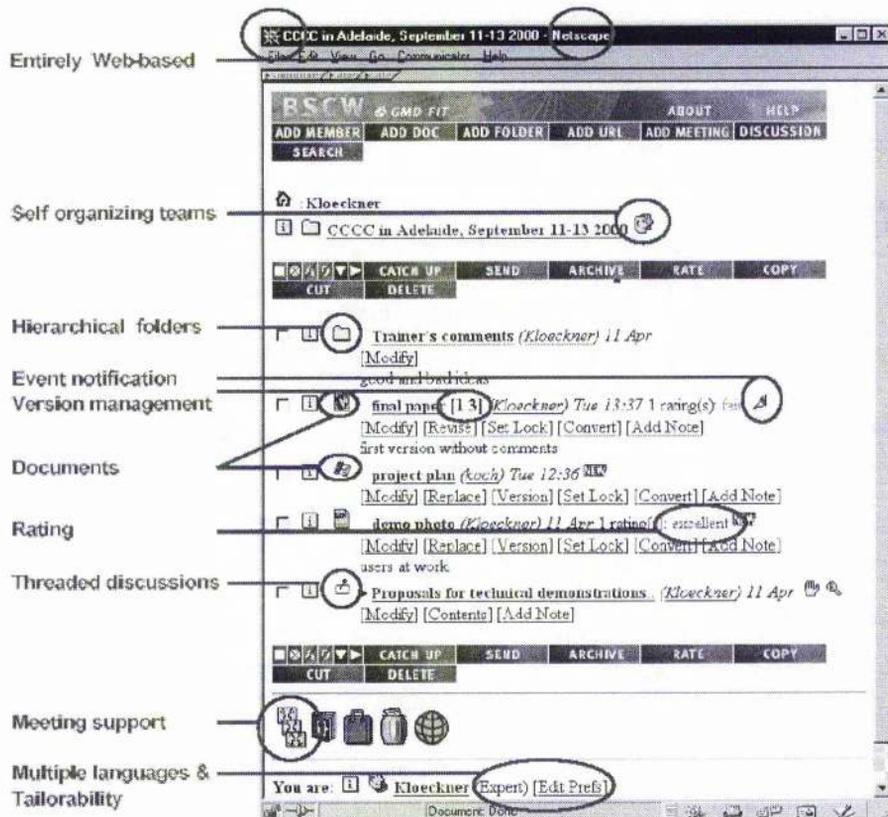


Figure 3.3.: The BSCW User Interface⁴

⁴ Annotated screenshot copied from Klößner, K. 2000. BSCW - Educational Servers and Services on the WWW. In: *International C4-ICDE Conf. on Distance Education and Open Learning*, Adelaide.

A shared workspace is shared by a group, and is accessible by members of that group from any web browser. Users belong to groups, they can create new groups, and can invite others to join groups. A user can therefore access many shared workspaces. The overall structure is hierarchical with respect to each user. Key features of BSCW include document locking, (to support collaborative authoring), group awareness (for teamwork) and 100% web-based (the access software client is a browser). The BSCW implementation makes relatively straightforward use of core web technologies: browsers, servers, HTML, CGI and JavaScript.

3.4.1 BSCW in Educational Contexts

A case study carried out by one of the BSCW authors compared ten features provided by BSCW in educational contexts. These are listed with brief descriptions in Table 3.3. TAGS (described in Chapter 5) evolved over a similar timeframe as BSCW but took quite a different approach to providing groupware functionality. The rightmost column in Table 3.3 lists TAGS resource types that provide the same functionality as BSCW.

One of the claims that the authors of BSCW make for their system is that it is integrated. The drawback is that it is integrated by employing one metaphor, the shared workspace, for all purposes. The metaphor has had to be continually evolved to accommodate a variety of new functional requirements. With respect to education it is hard to see how it can be stretched beyond generic document-centric groupware in order to integrate subject-specific resources. The TAGS approach would appear to be more useful, as the finer grained notion of a resource is used as the building block for a particular learning environment. Some resources provide groupware functions, other can provide subject-specific needs.

Use of BSCW	Description	TAGS Resource type(s) which provide(s) the same functionality
<i>Archiving</i>	A project group stores working documents on the Web	FileShare, DocShare
<i>Collaborative authoring</i>	Versioning – of a documents being updated	FileShare, DocShare
<i>Discussion</i>	Statement are proposed by the tutor for student reaction	Notebook, Q&A, Task List
<i>Reviewing</i>	Commenting on each others work	Assignment Tracking Tool, DocShare
<i>Monitoring</i>	Tutors can monitor students activities	Portfolio Management Facility, Assignment Tracking Tool,
<i>Communication</i>	Exchange of messages for team work	Notebook
<i>Using ICT</i>	How ICT can be used in education.	All TAGS resources
<i>Logistics</i>	Assignment management and hand-in	Assignment Tracking Tool, Task List
<i>Course info</i>	Dissemination of general information about a module	URL, URL Book, Notebook, Protected Page Set
<i>Access Control</i>	Fine grained access control over resource sharing	User, Groups, Domains and Resources Management Tool. Protected Page Set. (LDAP-based authentication,)

Table 3.3: Some Uses of BSCW in Educational Settings

3.4.2 Discussion: Web-based Groupware

The mode of sharing encouraged by the web is quite different from the sort envisaged by dOPT, and Warp (described in Chapter 4). BSCW leverages basic web facilities but fails to really escape far beyond a check-in, check-out model of sharing. This falls well short of the goals of realtime distributed groupware, as found in dOPT and WARP. However, if a measure of the success of a groupware system is its uptake, then the popularity and wide deployment of BSCW far outstrips more sophisticated systems, and exemplifies the success of the Web as a groupware platform.

3.5 PIPVIC: IP Multicast Groupware in Educational Contexts

The IPv4 multicast network capability was described in Chapter 2. Briefly, the experimental Internet Multicast Backbone – the Mbone – was developed to facilitate the transmission of multimedia traffic on the Internet (Deering 1996). Multicast traffic is efficiently supported by co-operating routers which dynamically maintain a spanning tree of point-to-point links in response to user-initiated joins and removes. The effect is like that of a spectrum of broadcast channels from which a user can select. Tools such as RAT (Perkins et al. 1998) and VIC (McCanne & Jacobson 1995) are used for audio and video. Joins and leaves are accomplished by the Internet Group Membership Protocol (IGMP). The motivation behind the PIPVIC project (1999) was to create a realistic mix of educational application loads to stress the new IP multicast capability and higher bandwidth made available by SuperJANET-4 (SJ4), a major upgrade to the UK's academic network.⁵ Prior to SJ4 the core Mbone in the UK had consisted of a network of Sun workstations running the *mrouterd* process. As such there was only limited throughput available, and in addition concern about the potential damage to ordinary data traffic that could be caused by video streams meant that these routers were obliged to politely limit their forwarding rate to 500Kb/s. With the advent of SJ4 the tunnel rate limits were removed. This was due to i) functionality that came bundled with the new core routers which supported multicast directly, and ii) a significant jump in the bandwidth.

The project was co-ordinated by UCL and involved twelve HE institutions making use of the Mbone tools mostly for teaching activities, but also for some administration and research. Table 3.1 lists the tools used and Table 3.2 summarises the activities.

⁵ The project was funded by UKERNA, a quango sub-contracted by the UK HE Funding Councils to provide network infrastructure for HE.

Tool	Protocols	Description
VIC	RTP/UDP/IP	VIC is a video conferencing tool. It offers a variety of encodings including motion-JPEG and H.261, and a controllable transmission rate in terms of frames and picture size. (McCanne & Jacobson 1995)
RAT	RTP/UDP/IP	RAT is the robust audio tool, designed to complement VIC, or run as a freestanding application. Like VIC it is possible to pick a preferred encoding which in turn affects quality and bandwidth parameters. RAT also attempts to perform silence suppression. (Perkins et al. 1998)
WB, WBD	Scalable Reliable Multicast/UDP/IP	These are shared Whiteboards. They have pixel paint semantics and each new user is allocated a new colour. They can import postscript.
NTE	Scalable Reliable Multicast/UDP/IP	Network Text Editor. This is a shared text editor for which the notion of coherence doesn't exist. All participants can read and write at the same time, and can try editing each others text. There is no notion of layout and results can be very messy. It's most useful when one person explicitly makes notes, and everyone else can see the notes being made. Any participant is free to save the document to local storage at any time.
SDR	UDP/IP	Session directory. This allows users to advertise and browse multicast sessions through a standard interface. It implements SDP, the Session Directory Protocol, and operates by transmitting on a well know multicast address.
SHRIMP	various	SHRIMP is a shrink wrapped version of the toolset for Windows. It was an attempt to make the tools more accessible to non-experts.
RELATE	various	A bespoke interface to a combination of the tools for language teaching.

Table 3.1: The Mbone Tools

Activity	Sites	Sessions	Applications and Platforms
EFL tutorials	UCL and Exeter.	4 participants 6 sessions	ReLaTe interface (students), separate RAT, VIC and NTE (tutor). 2 SGIs and 1 Win 95 at Exeter, Solaris workstation at UCL.
Italian for Engineers tutorials	Aberystwyth and Exeter.	6 participants 4 sessions	ReLaTe interface, NTE as shared workspace.
Russian for Beginners tutorials	UCL, Aberystwyth, Essex and SSEES.	5 participants 9 sessions	Shrimp ++ (NTE with Russian character support). All Windows platforms.
C363 tutorials	Westminster and UCL.	16 participants 8 sessions	Shrimp ++ tools and ReLaTe interface. WBD as shared workspace. 4 Windows platforms, 1 Solaris.
Distinguished Lectures in Software Engineering (Ian Sommerville)	St Andrews, UCL, Glasgow, Edinburgh, Aberystwyth, plus many nono-roject membets	25 Mbone participants (100 at venue). 3 sessions spread over 1 day.	2* VIC and 1 * RAT. Windows 98 400MHz PC and Sun Sparc 140MHz. Interactive Q&A with UCL Requirements Engineering Group, using RAT.
FINESSE sessions	Dundee and St. Andrews.	7 participants 2 sessions	Shrimp ++ tools. Windows NT workstations, VIC and RAT on Solaris Sparcs.
Business Game	Westminster, UCL and Glasgow.	7 participants 2 sessions	Shrimp ++ tools with ReLaTe interface at UCL and Westminster. Individual RAT, VIC and NTE at Glasgow. Windows platforms at UCL and Westminster, Suns at Glasgow.
Business Game seminar	Westminster, UCL and Glasgow.	10 participants 1 session	RAT, VIC and NTE.
PPNCG meeting	Glasgow, UCL, RAL, Oxford, Manchester, Imperial College.	7 participants 1 session	RAT, VIC and WBD on Windows 95 platform. Sites connected via CERN Virtual Room system.
Robot seminar	Essex, UCL and Aberystwyth, St Andrews	25 participants 1 session	Shrimp ++ tools, audio and three video streams only. Participants controlled robot through telnet window.
Strasbourg Oaths seminar	Aberystwyth and Exeter.	12 participants 1 session	Shrimp ++ VIC and RAT. Netscape Collaborative Web Browser for images.
History of Art seminars	Aberystwyth and Exeter.	8 participants 2 sessions	Shrimp ++ VIC and RAT. Netscape Collaborative Web Browser for images.
Privacy seminar	UCL and Glasgow.	14 participants 1 session	RAT and VIC on Unix platforms.
Sociology lectures	SSEES and Essex.	25 participants 20 teaching sessions	Shrimp ++ RAT and VIC, one-way only.
PIPVIC-2 weekly meetings	All project sites.	6-15 participants weekly	RAT, VIC and NTE on Unix and Windows platforms.

Table 3.2: PIPVIC Activities Summary

PIPVIC aimed to understand the issues involved in providing a large-scale videoconferencing service. The observations and measurements collected during the pilot were intended to provide the basis for a service specification for IP video conferencing across the UK academic networks. The objectives were:

- To determine the effectiveness of IP videoconferencing tools in a large-scale collaborative working environment.
- To provide a large-scale test of the concept of IP videoconferencing with a wide range of users in UK HEIs with different requirements.
- To determine a service specification for the academic networks in order that a successful IP videoconferencing service can be provided. The specification was to be made in terms of i) maximum packet loss rate; ii) end to end packet delay; iii) delay variation; and iv) any other parameters required for effective IP videoconferencing.
- To produce a model for the bandwidth utilised for different sizes of conferences involving different sites.
- To determine the scalability of IP videoconferencing on the academic networks, identifying issues that would limit the large scale deployment of an IP videoconferencing service.
- To assess the usability and interworking between desktop and room based facilities, and the integration of IP videoconferencing into current room based videoconferencing facilities.
- To determine the impact of IP videoconferencing traffic on Local Area Networks (LANs) at the pilot sites.

Note that the “academic networks” refers to all the networks used by end-users, including campus LANs, as well as SJ4. It can be seen from the above list that this was an ambitious project but that the activities were only “large-scale” in the sense that a wide geographical area was being covered, and that there was a variety of applications. There

were not actually very many people involved. Perhaps the most interesting and revealing activity was the series of weekly project management meetings that were held by using the Mbone tools VIC, RAT and NTE. The problems encountered in getting twelve sites, all manned by experts in the use of multicast technology, to function smoothly were quite substantial. It was rare for all sites to be satisfactorily connected to the session at the same time. As an active participant I was surprised at the low quality of audio and video that many participants seemed to be happy with. This was partly because the quality *expected* of the Mbone tools was not high and that desktop videoconferencing was a novelty in itself, regardless of quality. As these tools had been built to work in an earlier Internet environment where bandwidth was a scarce resource, their default settings were very conservative. Users, especially at UCL – the co-ordinating site – did not appear to expect a lot. This contrasted strongly with the relatively high quality video being used in Scotland at that time, based on the Scottish ATM infrastructure. The WARP project, described in Chapter 4, had used the Mbone tools with higher frame rates and bandwidth and had achieved much better quality, although still far short of that found in the dedicated ATM codecs used to transmit interactive audio and video in both room and desktop contexts.

The data that was gathered during the PIPVIC project included objective traffic records from VIC and RAT, paper questionnaires completed by participants after sessions, and informal e-mail. An interesting set of experiments was carried out during some of the weekly management meetings where both the subjective perception of quality of service and the objective network statistics were recorded for the same session. Participants recorded their subjective impressions of video and audio quality via a web form, three times during a session. All sites also ran a traffic statistics logging program and e-mailed the log to the co-ordinator at the end of the session (Watson & Sasse 2000a).

No problems were found with the new core network support for multicast. A major problem that did emerge was that the poor quality of some campus links, especially at Manchester and Imperial, precluded any benefit being seen from the new infrastructure (Watson & Sasse 2000b). This is analogous to the “last 100 meters” problem often

mentioned in telecommunications jargon, referring to the poor quality of the network connection between the local exchange and the household telephone.

3.5.1 The Mbone Tools as Groupware

The use of the Mbone tools for groupware provides a very different perspective from the sharing and coherence concerns of TimeWarp, dOPT and BSCW. Basically, the idea is that all applications are built around the real-time group communication facilities provided by the Mbone and the Real Time Protocol (RTP) (Schulzrinne & al. 1996). The groupware system model is peer-to-peer, and the pattern of communication is one-to-many. All applications follow the Mbone operational semantics, which is simply that participants may join and leave as they wish. All a participant needs to know is the IP address and port number. There is no admission control. All sites have equal rights to transmit to the group at any time, and can block out any selected participants at any time.

3.5.1.1 Usability

IP addresses fall into the IPv4 multicast range. An instance of an Mbone tool session, such as VIC, VAT or NTE, is uniquely identified by the combination of a multicast IP address plus a port number. The existence of a session can be notified by private communication (e-mail, phone call, etc.) or through the multicast Session Directory tool, SDR. Although accessing a session via SDR requires less expertise than starting a set of tools, one at a time, with combinations of IP addresses and port numbers as arguments, the successful creation and use of sessions still requires a degree of technical expertise that is well beyond the non-specialist. Not only is system support required in terms of hardware and software, but considerable user support is also required. Another consideration is that the Mbone is not ubiquitous. For example, very few ISPs will provide an Mbone feed to their customers.

3.5.1.2 (Lack of) Coherence

Mbone tools such as WBD (Highfield & Hasler 1999), WB⁶ (shared whiteboards) and NTE (Network Text Editor) are examples of real time distributed groupware which has little notion of coherence. NTE is a truly concurrent text editor that supports WYSIWIS in an uncontrolled fashion. It allows people to write to any part of a document at the same time and for all current users to eventually see the changes. There is no conflict avoidance or resolution – there is no notion of conflict. It is a free for all. Variance in network delay and reliability contributes towards the lack of coherence. New users joining a session see a version of the “current” state that is in reality a view unique to the local machine. There is no central storage. Any participant can save the local state of a shared document to their own computer at any time. Changes to a document propagate through the network to all users in real time(s), which means that they can arrive at different places at different times and in different orderings. The implementation can sometimes get a bit confused about what is being typed, leaving ghost text on the screen. Use of NTE has been described in (Sasse & Handley 1996). The software is available for download⁷. The brief description at the download site states: “Using NTE can be very interactive - unless you lock a block of text, anyone else in your session can edit that text or delete it. This is intentional. Many people can (if they wish) edit the same document simultaneously. Many people can even edit the same block of text simultaneously, but if more than one person tries to edit the same line at one time, a conflict will occur, which results in only one of the changes being preserved.”

Similarly, WBD and WB can be contrasted with a multi-user CAD/CAM system where system coherence is paramount to maintain the integrity of a shared model being developed by a design team - an aircraft for example. In the CAD/CAM situation there are typically strict rules regarding readers/writers and adaptive granularity of locking is used to allow concurrent updates. In WBD anyone can rub out anyone else's work, or

⁶ <http://www-nrg.ec.lbl.gov/wb/>

⁷ <http://www-mice.cs.ucl.ac.uk/multimedia/software/nte/>

draw over it, deliberately or by accident, because the events are not ordered. Two users can start drawing on the same white space at the same time and only see the consequences of each others actions seconds afterwards.

3.6 Internet2: A New Infrastructure for Education

The Internet2 consortium has grown since its inception in 1998 to include over 200 members, most of whom are higher education institutions in the US, but also including some prominent technology sector companies⁸ and members in Canada and Europe⁹. The primary source of its extensive funding is the US government. The primary focus is on developing the next generation of applications and networks in educational and research contexts, with a view to eventually transferring these advances into the commercial Internet environment.

“Internet2 is about everything we do in higher education. Therefore, we encourage and support applications development in all disciplines from the sciences through arts and humanities. Whether you're in the classroom, the laboratory, the library, or the dorm, you should be able to access Internet applications that provide benefit.”¹⁰

There are broadly three domains of activity in Internet2: applications, middleware and network infrastructure. Underlying the programme is the notion that the existing Internet has become too congested and unreliable for education and research, and that a new generation of applications based on high-bandwidth, low-delay networks and appropriate enabling middleware should be built to continue the advance of e-society and e-science. Table 3.5 summarises the programme from this perspective.

⁸ Technology sector companies listed as members of Internet2 include Cisco, Microsoft, BBN, Sprint etc. See <http://www.internet2.edu/members/html/corporate.html> for the full list.

⁹ Generally speaking, the national academic networking quangos of Western European countries are affiliated, thereby linking in the HE sectors. The UK HE sector is represented by UKERNA.

¹⁰ Quote from Ted Hanss, Internet2 Director for Applications Development, at <http://apps.internet2.edu/html/faq.html>

Category	Areas of Interest and Activity
<i>Application:</i> http://apps.internet2.edu/	Distributed Instruction, Interactive collaborative environments (inc. high quality desktop and seminar room video conferencing), Virtual presence at remote laboratories, Supercomputing on tap (the GRID), 3d immersive interactive virtual reality.
<i>Middleware:</i> http://middleware.internet2.edu	Distributed resource discovery, sharing and access control. Public Key Infrastructure (PKI), Directory based services, LDAP. Projects include Shibboleth, EduPerson, DoDHE (Directory of Directories for Higher Education), and the HEPKI (Higher Education PKI).
<i>Infrastructure:</i> www.internet2.edu/html/advancednets.html	High bandwidth, low latency, efficient multicast, effective delivery to the desktop, Quality of Service, DiffServ, distributed storage architecture, peer-to-peer architecture. Networks include Abilene, vBNS and ESnet. Projects include the Qbone.

Table 3.5: Internet2 Areas of Interest and Activity

3.6.1 Internet2 and Quality of Service: The Qbone

A major problem with the existing Internet is that there is only a best-effort packet delivery service, and that unpredictable peak loads at routers cause congestion and packet loss. There is no means for a distributed application to negotiate quality of service parameter values such as bandwidth, delay and jitter. The Qbone is a testbed for Differentiated Services (Teitelbaum et al. 1999). The Qbone architecture builds on emerging IETF standards for DiffServ forwarding by specifying mechanisms for achieving inter-domain end-to-end service quality. Its goal is to deliver the Qbone Premium Service, which will behave as a virtual leased line across wide area inter-domain routes. That is, it will offer network connections with virtually no packet loss, a known peak-limited bandwidth, and virtually no delay or jitter due to queuing effects. It is also planned to build in defences against Denial-of-Service attacks.

An alternative approach, such as is taken by TAGS, described in Chapter 5, is to be adaptive rather than proactive. That is to say, to actively monitor QoS and feed the information back into the system(s) concerned. QoS in TAGS (Allison et al. 2001), and other studies, have shown that the network is not necessarily a limiting factor in achieving satisfactory end-to-end QoS. For example, several of the TAGS QoS measurements (Allison et al. 1998) took place over a 155Mb/s IP/ATM connection with

virtually no packet loss, but it turned out that the end-user systems did not take advantage of the excellent network conditions. It is clear that this problem has also been met in Internet2. The Web100 project¹¹ acknowledges that existing applications may actually run worse over the Qbone than they did on the standard Internet. A set of tools is being made available for re-tuning applications to take advantage of the Qbone.

3.6.2 Distributed Access Control: Shibboleth

MACE is the Internet2 Middleware Architecture Committee for Education. The core concerns of middleware development in Internet2 are resource sharing, authentication and access control. The middleware layer is seen as essential for the realisation of “Internet2”, as is depicted in Figure 3.8.

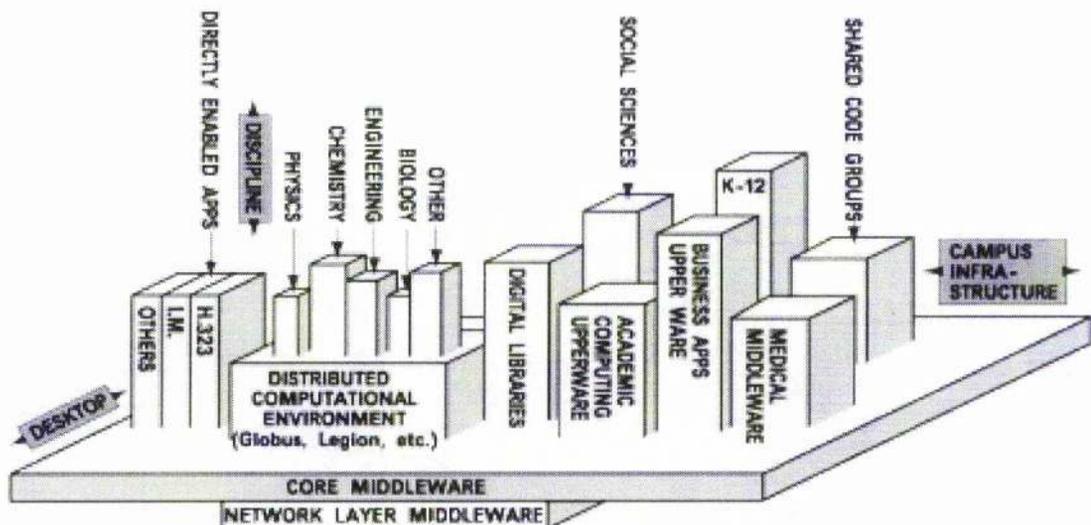


Figure 3.7: Internet2 Middleware Perspective, January 2002¹

In practice, regardless of the speed of the network, co-operating institutions want to start the controlled sharing of web-based education resources now, and this requires some working software. Shibboleth intended to address this pressing need. It is a joint project of MACE and IBM, and is “developing architectures, frameworks, and practical technologies to support inter-institutional sharing of resources that are subject to access

¹¹ <http://www.web100.org/>

controls". This is very similar to the access control functionality provided by TAGS, but is expected to work on a much larger scale. The "practical technologies" is meant to include, as a deliverable, a public domain, open source extension to the Apache web server which will allow for controlled sharing. The basic idea for Shibboleth is shown in Figure 3.9.

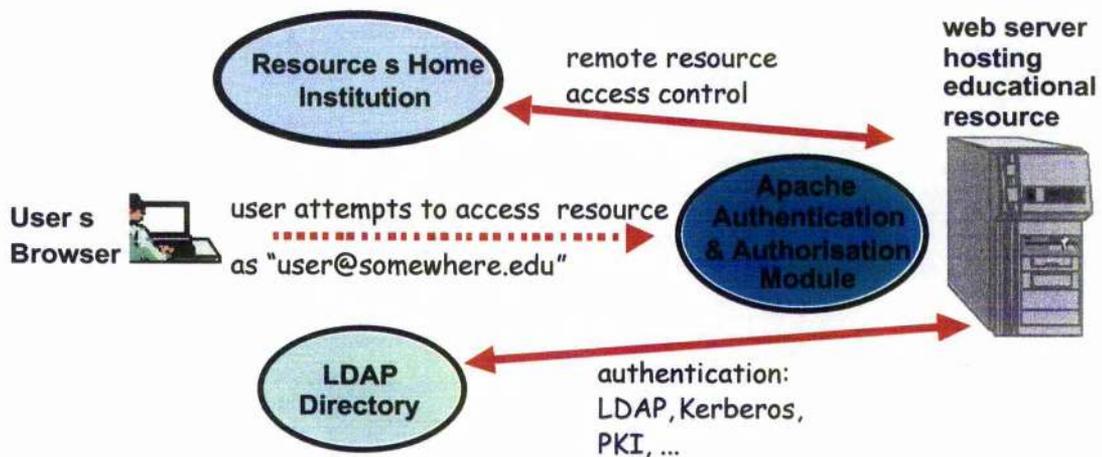


Figure 3.9: Shibboleth Access Control

Prior to addressing the distributed authentication issue as part of the TAGS project I contacted R.L. "Bob" Morgan, the Chairman of MACE, in July 2000, to ask how the work was progressing, and if the Apache module was ready. He replied that they were "still at the analysis-of-problem-space stage"¹. Accordingly, TAGS pushed on and implemented a scheme, based on LDAP (the Lightweight Directory Access Protocol) which extends the TAGS resource sharing and access control model to multiple authentication domains. The scheme covers both static HTML as well as the more easily regulated server-generated content. It is described in Chapter 5.

¹Copied from presentation by Ken Klingenstein, available at <http://middleware.internet2.edu/>.

¹ Personal communication from RL "Bob" Morgan, Chairman of MACE.

3.7 Summary

This chapter has presented work related to Coherence, QoS and Framework issues in educational and groupware environments. Timewarp and dOpT offer optimistic non-blocking approaches to maintaining the integrity of shared resources. In strong contrast multicast-based groupware tools such as WB, WBD and NTE are something of a free for all, their behaviour being heavily dependent on the underlying network QoS, and polite behaviour by the end users.

BCSW is possibly the best known of the early web-based groupware environments. dOPT, BSCW and Mbone groupware all have one thing in common: they attempt to use a single metaphor or mechanism for structuring all interaction. dOPT uses fine granularity, "fix-it-up-later" editing, BSCW uses the shared workspace and the Mbone tools use real-time multicast. Engelbart has stressed that in practice real work involves users moving between different modes, preferably seamlessly (Engelbart 1990). So a groupware environment should contain a variety of mechanisms and metaphors, and this is what the TAGS framework, described in Chapter 5, facilitates in its resource and group allocation model. The notion of a resource in TAGS is deliberately loose in order to accommodate different approaches, but the framework integrates them through a user-centric portal. The main point is that a DLE must accommodate a critical combination of QoS, Coherence and Framework components.

4. Addressing the Coherence Requirement

The need for shared object coherence is a key requirement of distributed groupware systems. Many applications resort to ad hoc techniques, typically not reusable, and often faulty. The search for a uniform coherence mechanism started with building a Timewarp programming kit, Twarp. The initial attraction of Timewarp, namely that it allowed for optimistic progress of concurrent operations proved to be less useful than initially thought. This was primarily because the logical time maintenance algorithm in Timewarp creates too tight a coupling between all nodes in a system, and results in a node being rolled back even when it had not been in conflict. The advantages of Timewarp as a coherence mechanism looked increasingly difficult to sustain outside of discrete event simulation applications. Perhaps the main insight gained from Twarp was that a mechanism was needed where time did not have to be explicitly managed by an application. However valuable experience was gained with state definition, checkpointing, and backtracking, and this was used in the design of Warp, the system which succeeded Twarp. Warp removed the need for programmers to explicitly attach timestamps to messages, and allowed for non-conflicting operations on shared objects to progress concurrently without risk of rollback. Both of these systems are described here, followed by a case study of a groupware application programmed built with Warp: a multi-user, distributed, shared spreadsheet. The spreadsheet in turn was used to develop a multi-user business game for management students, and represents the first step taken into the construction of a distributed learning environment.

4.1. Explicit Virtual Time Based Coherence

Twarp (Livesey & Allison 1992) investigated the use of the Timewarp model as a general-purpose optimistic coherence mechanism which could be used to provide coherence in distributed systems. The system was developed on a network of Sun workstations, running SunOS 3, connected by a 10Mb/s shared ethernet. This implementation of

Twarp on Sun's version of BSD Unix is briefly described under the headings: Process Model, Inter-process Communication, Node Configuration, Global Virtual Time, and Persistent Object Storage.

4.1.1. Twarp Process Model

Logical Twarp processes (TW nodes) have to be mapped onto the underlying Unix process model and a physical network of machines. Two types of mapping are considered, a single process model where each Twarp process is mapped onto a single Unix process, and a two process model where each Twarp process is mapped onto two Unix processes.

4.1.1.1 Single Process model

In the single process model (Fig. 4.1a) each TW node is a self-contained unit corresponding to one Unix process and is responsible for its own message passing, queue management, checkpointing and rollback. Twarp was implemented before threads or lightweight processes were available in Unix, and a drawback with the single process approach is that it provides no scope for concurrent execution within a TW node. For example, interrupt routines to handle external communications interrupt the progress of the application specific computation, even when the message being processed may have no direct consequences for that computation.

4.1.1.2 Two process model

In the two process model (Fig. 4.1b) each TW node consists of an applications program process (a Twarp client) and a Twarp server process running concurrently. The Twarp server contains reusable procedures and data structures common to all TW nodes and carries out all communications on behalf of the Twarp client. When a TW node wants to send a message to another TW node participating in a distributed computation the client tells its server to send the message to the remote node. Similarly, when a message arrives at a Twarp server, it is passed to the corresponding client, if necessary, when required. This system lends itself readily to distribution, and is not heavily reliant on

any particular component. The decoupling and parallelisation of communication and computation is a technique also used in algorithms such as the Baker's Algorithm (Lamport 1974), and the ISIS group communication variants (Schmuck 1993).

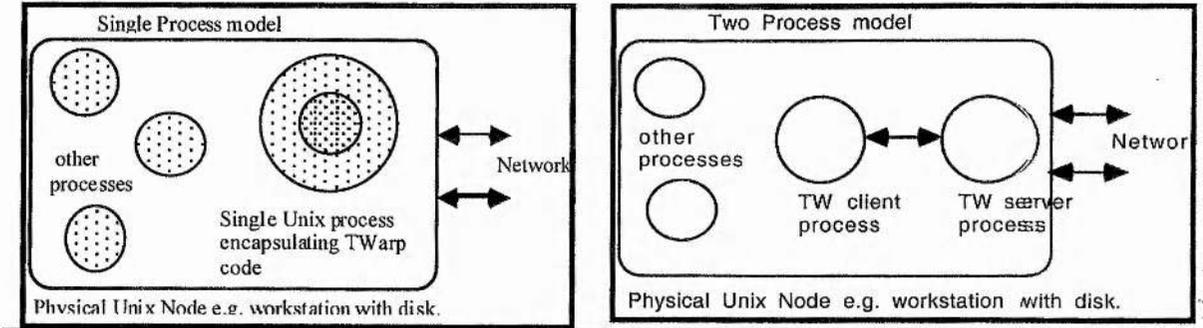


Figure 4.1a

Figure 4.1b

4.1.1.3 The Monitor Process

A monitor process is used to provide management facilities. It is responsible for starting a distributed application, for interaction with the outside world (excluding any actual Timewarp I/O) and for reporting or forcing termination. The monitor may also be used for statistics gathering and GVT calculation. In the same way that scaffolding can be removed once a development is complete an application may be run without the monitor. The design aims to keep any intrusive effects of the monitor to a minimum.

4.1.1.4 Inter Process Communication

The Twarp server is an interface between an application and the network. The two Unix processes which constitute a Twarp node do not have a typical Internet client-server relationship. In ISO OSI reference terms the Twarp server manages the session layer and interfaces with the transport layer. The view from the application (the Twarp clients) is that of a reliable message delivery service. (Figure 4.2).

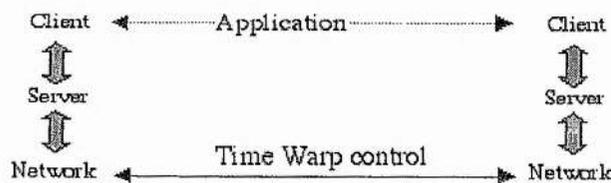


Figure 4.2

Timewarp as outlined by Jefferson(Jefferson 1985) requires a transport service that guarantees delivery of datagrams exactly once, although not necessarily in the order sent, because messages are ordered by virtual receive times (see section 3.3.2). In practice most transport services that guarantee delivery exactly once also guarantee FIFO semantics, so the relaxation of FIFO message delivery does not necessarily have any practical benefit.

Distribution requirements are as follows:

- i) Twarp servers must be capable of running on different hosts
- ii) Twarp servers must be capable of running on the same host
- iii) a Twarp client must be able to run on the same host as its server;
- iv) a Twarp client should be able to run on a different host from its server
- v) the non-volatile storage associated with a Twarp node should be able to reside at any host on the network

Although the target environment is one consisting of autonomous (disk-full) hosts, requirements (iv) and (v) mean that diskless, or storage free nodes, can also be accommodated.

4.1.2. Operational Communications Model

SunOs provides four network programming abstractions for the application programmer (Sun_Microsystems 1990a, b): BSD sockets, Sun Remote Procedure Call (RPC), Sys V streams, and Sys V Transport Level Interface(TLI). The latter two options are specific to AT&T versions of Unix, whereas Sockets and Sun RPC are de facto Internet standards. A drawback with Sun RPC is that the semantics vary with the underlying network service used and a clear high-level procedure call abstraction is not present. Sockets were selected as they provided a simple, clear abstraction.

The next choice was between UDP or TCP as a transport. TCP sockets offer reliable FIFO communication but have two drawbacks: i) maintaining channels between all

participating nodes requires n^2 connections, which is neither scalable nor efficient; and ii) set up and close down time can be relatively long so dynamically opening and closing a connection for each datagram can be expensive, and, in the wider Internet scenario, contribute towards network congestion¹. However, client-server connections that only have to be set up and shut down once in the course of an application's execution period can benefit from a TCP connection without incurring significant overhead.

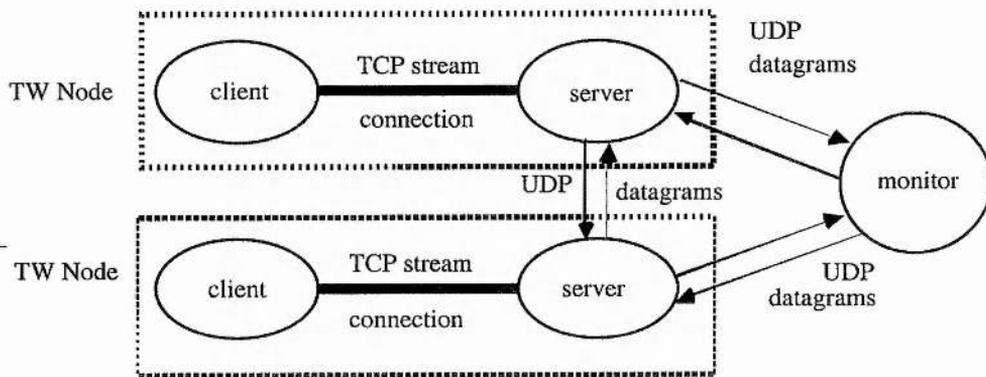


Figure 4.3: Internet transport level protocol selection

The approach taken is to use TCP and UDP where each is most appropriate (Fig.4.3). The monitor program is used to set up its own communications and also provides the arguments that all the Twarp servers and clients need. A pair of UDP sockets is set up between each server and the monitor to handle the sending and receiving of messages. The receive socket is associated with a SIGIO signal handler which is called when a signal is received indicating a message arrival. The signal handler reads the incoming message from the socket and places it in a buffer. This procedure is called whenever any data is waiting to be read, and, once the data has been placed in the buffer, the process continues. The mechanism is cheap, efficient and handles asynchronous message

¹ http v1.0 was heavily criticised by the Internet technical community for opening a new, separate TCP connection for every file transferred even when all communication was between the same two end points. http v1.1 attempted to rectify this problem.

arrivals, multiplexing if necessary, with a very low probability of being unable to service an interrupt and dropping a packet. The server-client pairs will frequently communicate and will last for the lifetime of the application's execution. They are connected by TCP.

4.1.3. Node Configuration

Twarp assumes a set of autonomous machines connected by a message handling system. Firstly, a separate source code file is created for each Twarp node. Secondly, a configuration file is created for the application, consisting of the filenames, their logical Twarp node names, and the machines they should run on. In this implementation the environment consists of homogenous workstations on a shared ethernet based IP subnet, which allows for simple relative host names. For example, the text in Fig. 4.4 indicates that the application consists of five Twarp nodes made up of five files. The first file is `carwash1.c`, which has a logical Twarp name `CW1`, and should be executed on Unix host `macallan`. The second Twarp node should be executed on `farclas`, the third on `davaar`, and so on. Finally, the configuration file is passed as a parameter to the monitor program. The monitor program combines the configuration file with a Makefile template and then runs the Unix `make` utility to compile and link the code wherever necessary before passing the details to the monitor programme, pauses and waits for a start command. On startup the monitor reports the progress of runtime initialisation. The monitor remains active during the distributed execution and exits after reporting natural termination. The monitor will accept an abort command to force termination. The monitor may be run on any network host.

<code>carwash1.c</code>	<code>CW1</code>
<code> macallan</code>	
<code>carwash2.c</code>	<code>CW2</code>
<code> farclas</code>	
<code>attendant.c</code>	<code>ATT</code>
<code> davaar</code>	
<code>source.c</code>	<code>SRC</code>
<code> atholl</code>	
<code>sink.c</code>	<code>SNK</code>
<code> laphroaig</code>	

Figure 4.4: A Twarp Configuration File

The configuration file approach is simple and extensible. For example, TW node requirements in terms of memory and processor type could be specified in the file and the decision left to the monitor to select appropriate machines from a database of available computers and their hardware configuration.

4.1.4. Global Virtual Time

GVT is needed to:

- **Fossil collect.** Any checkpointed states that have a timestamp less than GVT are fossils and may be deleted. This can take place at any time and must take place when storage space is an issue.
- **Allow irrevocable events to be processed.** Whenever a server encounters an irrevocable event it must wait until the GVT has passed the virtual time associated with the event before it can be acted upon. For example, it is not possible to unlaunch a missile or undo printer output.

GVT is defined as being less than or equal to the minimum of (i) all virtual times in all virtual clocks, and (ii) all virtual send times of messages in input queues that have not yet been processed. As these are local data, the monitor cannot directly access that information, but must instead ask each server for its data.

GVT can either be calculated by the monitor or by any server. If GVT is calculated by a server then a means of passing the value to all other servers is necessary as it would be inefficient for each server to calculate GVT every time it was required. In the case of a network that efficiently supports broadcast, or even better, multicast, servers can snoop on any GVT calculation. If, on the other hand, GVT is to be calculated by the monitor it may be calculated either on demand or at certain time intervals and stored.

When faced with processing irrevocable events the server needs to know if GVT has reached the time at which the event is to be processed. This can be achieved by the server repeatedly asking the monitor for GVT or alternatively the server could ask if it is

at the appropriate time. The monitor would then broadcast or multicast a query, such as "Does any node have a LVT less than 23?" Servers would then reply with their LVT only if it is less than 23. If a server's LVT is higher than the specified value it will not reply at all. This is equivalent to the server asking the monitor for the GVT, and the monitor calculating it, with the saving that not all the servers will have to reply. Setting a time-out for reply arrival would then be necessary. Later implementations have also sought to embellish Timewarp with broadcast (Bayerdorffer 1995).

The solution presented above is adequate for handling irrevocable events, but insufficient for fossil collection. Ideally, if the system is very busy, with lots of servers requesting the GVT, the monitor should calculate GVT every few requests and give the most recent calculation to the others. When the system is very quiet it should calculate GVT each time. This can be achieved by a mechanism which stores the time (real) elapsed since the last calculation of GVT. The programmer can specify a time interval which will decide whether GVT has to be recalculated or not, i.e. if a request comes in before the elapsed time register has reached 5, then GVT will not be recalculated, but if the time is greater than 5 the GVT will be calculated.

By combining the two solutions described above, we have an optimum method, involving two ways in which a server gains access to GVT: i) it *proposes* a new value for GVT; or ii) it *requests* the current value.

4.1.4.5 Autonomous Processes and LVT

The original Timewarp mechanism was concerned with an already-running application and no thought appears to have been given to the execution of an application from start to finish. Jefferson's model states that a TW node's LVT should be set to ∞ , denoting infinity, the highest possible LVT, whenever it has no messages in the input queue. Thus, when all LVTs are set to ∞ , i.e. when there are no more messages being sent, GVT will also become equal to ∞ . When GVT is equal to ∞ the application has terminated as no process can roll back. This is an unfortunate choice of mechanism for

termination. It means that the Timewarp mechanism cannot cope with *fully autonomous* processes where messages are sent, but none are received. If such a process were to be implemented the results would be disastrous. Taking a Car Wash simulation (Mistra 1986) as an example, there is a source process which generates messages (cars) at random times, but never receives any messages. As there are no messages in its input queue, the LVT of the car generation process is set to ∞ , denoting infinity, the highest possible LVT. The fact that ∞ will never change in an autonomous TW node violates the fundamental principle and intuition that the virtual send time of each message must be less than its virtual receive time. So the receiver should set its LVT to $\infty + 1$. Obviously it is impossible to have a time stamp greater than ∞ . Furthermore, all messages sent out from such a process would have the same send time. Having an LVT permanently equal to ∞ also creates problems when receive time is dependent on send time, for example, "when LVT = x send message", or "send message with receive time LVT + 3".

Rollback is not a problem in the case of a node that never receives any messages as it can never be sent any anti-messages. However, if a node receives only a few messages, rollback *will* be a problem. Initially a node will have an LVT of 0. If there are no messages sent to the process, this will change to ∞ . Then, as soon as a message arrives the TW node will have to roll back as the receive time will definitely be less than ∞ . Thus, *all* the messages sent so far must be re-sent as anti-messages because they all have a send time of ∞ . So, every time rollback occurs, the process effectively starts again from time 0.

One solution to this problem is to have two distinct classes of Twarp nodes, autonomous and non-autonomous, where the non-autonomous nodes act in the manner specified by Jefferson's model, and the autonomous nodes use some other mechanism. This option is discounted as it would add complexity both to the Twarp process model and to the task of programming an application. The following major change to Jefferson's model is used: instead of the server increasing its LVT only when a message is received, the

server will constantly update the LVT itself (autonomously). This LVT is then explicitly sent to the client even when there are no messages. Where, before, the client (the application program) only contacted its server when it wanted to send a message to another client, and the server only contacted the client when a message arrived, i.e. they were fairly independent, we now have the client and server very closely linked. In fact, we almost have a master-slave situation where the client is actually the slave. One can imagine the client as a Finite State Automaton where it is passed a state from the server, does some computation, and returns another state. This is all the client now does; it *reacts* to particular information. In practice, however, the client is not explicitly sent a state, rather it operates on a previously defined state.

4.1.1.6 Granularity and LVT

We now need to introduce a server-client message protocol to control LVT. This raises the question of how often the server should inform the client - the granularity of LVT advancement. Note that there are two general types of message, Timewarp system messages and application domain messages. Application messages are encapsulated in Timewarp system messages. We will refer to a system message as a TW protocol data unit (PDU). A simple interval mechanism is introduced to control granularity.

The server sends a TW PDU to the client:

t_0 (integer), **t_1** (integer), **n** (integer), **msgs** (application messages)

where **msgs** represents **n** application messages, each of the form:

receive time (integer), **sender** (integer), **length** (integer), **application data** (bytes)

This means that given the LVTs **t_0** and **t_1** , messages **msgs** and the state at time **t_0** , send all output for all LVTs in the interval [**t_0** , **t_1**] and the new state identifier at time **t_1** . If **$n = 0$** then there are no messages, and the server just wants to know if the client has any output during this time. Specifying a time *interval* in this manner allows control of the grain of operation. The system places no interpretation on the content of application messages.

The client will reply with a TW PDU:

flag (integer), n (integer), msgs (application messages)

where *msgs* represents *n* messages, each of the form:

receiver (integer), receive time (integer), length (integer) application data (bytes)

whereupon the server will send off the various messages (if any) and will possibly save the current state in the stable store. The flag **flag** indicates that the client is finished, waiting on input, or continuing.

4.1.1.7 Rollback and LVT

Having adopted the LVT interval mechanism, it now appears that rollback could be a problem in that a server incrementing its own LVT may run ahead too far. For example, say the time step is 3. We may have the case where there are no messages in the input queue, and the TW node runs ahead. The server questions the client at times 0, 3, 6, 9. Then a message arrives with receive time 4. The server must now rollback, and send all anti-messages from time 3 to time 9. It then carries on, times 4, 7, 10. Now a message arrives for time 6. Rollback again occurs. System behaviour could be unacceptable in the worst case scenario. As a solution to this we could design an *adaptive increment* which would depend on various parameters, such as the average number of rollbacks per message and vary its size accordingly. The opposite of this may also occur - while the server is questioning the client at times 0, 3, 6, 9, etc. a message arrives with receive time 23. Instead of increasing the increment, the next interval should be changed to (9, 24). In fact, whenever the input queue is not empty, the upper time limit should be set to the highest receive time of the queued messages.

4.1.5. Persistent Object Storage and State

Because Twarp nodes progress independently regardless of synchronisation implications, rollback may be required at any time. To facilitate this the state of the

Twarp client must be saved periodically. Before any snapshots are taken the subtle question of what constitutes the state of an active process must be addressed.

4.1.5.8 A Definition of State

By a state we mean all the information about a process at a particular time that must be recorded to enable that process to be reinstated to that (virtual) time at some point in the future (real time). If a process is to be checkpointed after an arbitrary interval much may have to be recorded, for instance the program counter, the current scope level and all the variables and structures. It is difficult to access such information in an active process.

So what do we need to define our notion of state? One may decide that all the variables in a process define its state. However, extracting even that information from active processes at different times is not trivial. By limiting the points in the client at which the state may be recorded to those in the outer level of scope the matter is simplified. We extend this restriction and permit states to be saved only after each LVT interval is processed by the client negating the need to record scope information and the program counter. Furthermore, it may not always be desirable for all the variables in a process to be part of the Timewarp mechanism. Those variables which are required to return to their stored values at a certain LVT given a rollback to that LVT, are said to be state variables. State variables in some systems are severely restricted, namely where state variables may not be heap items. The applications programmer defines the client's state by choosing which variables are state variables. We make the assumption that the programmer knows enough about the application to define a correct state for each client.

4.1.5.9 Recording and Accessing States

Given that a (client) state has been defined it must also be accessible from the server. The server requires access to the current client state to record a snapshot of that state. The server must also organise those dumped states in some way that will permit rollback and fossil collection. To allow both the client and server to access the state the following methods are possible:

Message Passing – A message containing the state is passed from the client to the server and back in turn-about fashion. This is wasteful as a potentially large state message would be passed frequently, which involves encoding the state into a message at the source and decoding a message as state at the destination. The encoding of arbitrary client defined states is non-trivial.

Shared Memory – The client and server could share memory, or both memory map the same file. A major drawback here is the work involved in representation of a heap and its objects on non-volatile storage media.

Persistent Object Store – The Persistent Object Store (POS) open architecture interface (Brown 1989) provides C programmers with a set of persistent stable heap management routines. By storing the state in a POS it may be accessed by both the client and server, with no overhead in message passing, a reduced complexity and adherence to the TW node client server model. By using the POS to store state variables, those variables may be manipulated in any way consistent with the POS interface. Any type of state variable may be modelled using the POS, including dynamically growing structures. To dump a state we make it an object, or objects, in a POS.

4.1.5.10 Integrating the POS

Several schemes for an interface between the POS and Twarp client and server processes are possible: i) one POS is use by all Twarp applications; ii) a POS is created per application; iii) every client-server pair within an application has its own POS.

The last option is the most distributed and versatile, and completes a functional definition of a TW node: a server, a client and a POS. To use a store it must be opened, operated on, stabilised and then closed. Only then may another process operate on that store. Stabilisation is necessary as modified cached data must be written back. As only one POS may be opened safely by one process at a time the client and server acting on the same store must operate on that store in a mutually exclusive manner. This is achieved by organising client/server pairs as co-routines.

4.1.5.11 Using a POS

States will typically be dumped in order of increasing LVT, except immediately after rollback. The client only needs access to that part of the POS which models its current state. Accordingly, the states are stored in a linked list ordered by LVT, with the current state at the head of the list. The current state is a transient object, subject to state variable updates. The burden for organising the POS and carrying out state saving, rollback and fossil collection lies with the server. Exactly when these actions should be carried out is determined by the applications LVT advancement policy.

4.1.5.12 State transition

The client accesses its current state by calling a procedure to return the first object of that state. On moving to a new state the current state is copied to a new object which becomes the stored (non-active) state with the highest LVT.

4.1.5.13 Rollback

When rollback to LVT t_n occurs the list of state objects must be traversed until one is found with some LVT t_k where $t_k \leq t_n$. Once found the object with LVT t_k is made the current client state and LVT is updated accordingly. Rollback to a point less than the first explicitly saved state is possible as the very first state is recorded at LVT 0 on initialisation.

4.1.1.14 Fossil Collection

Stored states with a LVT less than GVT are fossils and should be removed as they are an unnecessary waste of storage space. This involves breaking the link between the state with LVT immediately above GVT and the state with LVT immediately below GVT. The fossil collected states are now garbage in the POS and will be automatically garbage collected by the POS utility.

4.1.6. Lessons Learned from Twarp

Twarp offers a means of achieving coherence in distributed simulations. However, in common with other implementations of Timewarp, it deviates significantly from

Jefferson's design in order to obtain efficient operation. The model itself has the serious drawback that Global Virtual Time creates a barrier that can impede concurrency even where it is safe for a node to proceed. Similarly, rollback can be triggered at a particular node for no good reason. The fundamental problem is that the coupling of autonomous nodes is too tight for all applications. It makes sense in discrete event simulations, but not necessarily for other purposes. The impact on a groupware system could be drastic – a user could suffer having their work undone for no apparent reason. The Twarp project's main lesson was that the abstractions presented to an applications programmer required too much attention to the coherence mechanisms requirements, and not enough freedom to concentrate on the applications own substance. Twarp relies on explicit virtual time stamps being both generated and acted upon by the application. How successful would virtual memory be if a programmer had to explicitly tag program statements with page numbers and permissions?

4.2. WARP: A Coherence Mechanism for Distributed Groupware

The development of Warp was informed from the lessons learned from Twarp. Like Twarp, Warp sought to provide a single uniform coherence mechanism. The accompanying API aimed to simplify the programming of distributed systems, with a particular focus on realtime distributed groupware (RDG).

4.2.1. System Overview

A Warp system is made up of a group of logical nodes (Fig 4.5) consisting of the following components:

- an object manager, OM
- a conflict table visualiser, CTV
- a console
- applications
- the runtime library

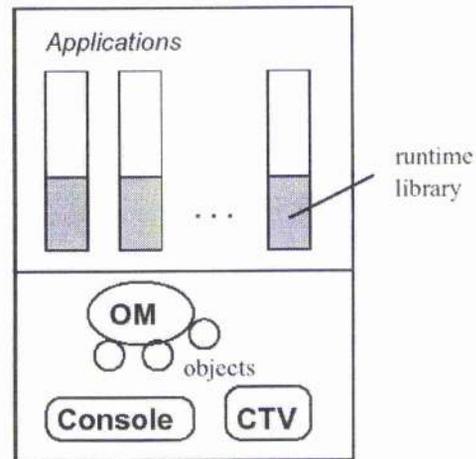


Figure. 4.5: Warp Node Components

The general approach in Warp is to augment the serialisability and atomicity properties of transactions with other features useful for groupware applications – automatic conflict resolution, fairness, and liveness. Augmented transactions are referred to as *atoms*. Atoms compute over *objects*, which are passive chunks of data. Both atoms and objects have globally unique identifiers.

The interactions between atoms and objects are conceptually represented in a global conflict table, CT, as shown in Fig. 4.6. The CT is a matrix with rows indexed by atoms, and columns by objects.

		Objects			
		Obj_1	Obj_2	Obj_3	Obj_n
Atoms	atom_1		OS, TS	OS, TS	
	atom_2	OS, TS		OS, TS	
	atom_n				

Figure 4.6: Warp Conflict Table (CT)

Each non-empty CT entry represents

- A shadow copy, or clone, of the object. atom_1, for example is operating on clones of objects Obj_2 and Obj_3.
- The current status of an atom with respect to an object: OS, TS.

OS refers to the ownership status and TS the timestamp. Only one ownership token exists for an object, so at most one status entry can indicate ownership in a column. The timestamp TS refers to the first access of an atom to an object. Time is global and logical, maintained by a variant of Lamport's clock algorithm. Each atom obtains its own shadow copy of each object it requires from an Object Manager.

Serialisability is the basis for correctness in the Warp mechanism. The first access to an object is referred to as the atom's *initial touch* of the object. The state of the host process immediately prior to the touch is checkpointed. Atoms proceed optimistically, but must ultimately acquire locks on their objects, called *owning* the objects in Warp, in order to commit. When an atom commits it may cause other atoms holding copies of objects to *backtrack*, if an object has been updated by the commit. The backtracked atom does not terminate, but obtains a new copy of the each object it needs. If an atom does not own all the locks it needs to progress it may be required to *backoff*, in order to allow for re-arbitration, which is performed by the Warp mechanism.

Table 4.1 shows how Warp operations (described in more detail later) are mapped on to the CT entries. Rows and columns in turn are mapped on to multicast group communication channels. Messages that are multicast to a row are seen by all Object Managers involved in an atom, messages multicast to a column are seen by all atoms involved with an object. When an atom completes its row disappears from the CT.

Operation	CT Dimension
initial touch	row
backtrack	column
backoff & commit	row

Table 4.1. Operations on CT by row and column

IP multicast is used to enhance scalability. A multicast-based variant of RPC-2 developed by Huang is used for reliable group communications. Scalable reliable multicast can make significant demands on network infrastructure. It is therefore important to note that the required *degree of scalability* is application dependent. For

example an RDG application may only need to support 20 concurrent participants at most.

4.2.2. Objects

An object is defined as a byte sequence with arbitrary length and sharable between all applications. Objects are stored and maintained by their OMs, and named by globally unique location independent object identifiers, OIDs. The format of an OID is shown in Figure 4.7.

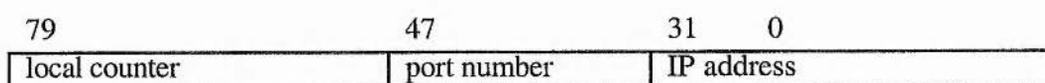


Figure 4.7. OID Format

An **OID** consists of a locally unique identifier based on a monotonically increased local counter, and the port number and the IP address of the OM at which the object is created. OIDs are mapped into multicast group addresses. The OM that manages an object has to join the multicast group associated with the object. When an atom tries to make an initial access to an object it multicasts to the group associated with the object and the OM in charge should replies. The Object Manager interface is shown in Table 4.2.

Name	Description
warpOM_CreateObject	Create a new object
warpOM_AccessObject	Read write or delete an object in or outside an atom
warpOM_BacktrackConfirm	When an atom is backtracked it has to multicast to all the OMs it has touched to clean up the CT
warpOM_Rearbitration	Rearbitrate all the ownerships the calling atom holds
warpOM_Validation	Check whether the calling atom owns all the objects it has touched to decide whether the atom can commit
warpOM_Commit	Commit the calling atom. Update all the objects modified by the atom and clean up the CT.

warpOM_Abort	Abort the calling atom. Give up all the ownerships the atom holds and clean up the CT
warpOM_ObjState	Get the conflict state of an object
warpOM_GetStats	Gets various statistic data

Table 4.2: The Object Manager Interface

4.2.3. Atoms

The fundamental unit of execution in Warp is the *atom*. A simple high level API is shown in Table 4.3. *atom_begin* and *atom_end* are macros. A fuller account of the runtime library is given later. Atoms are serialised atomic actions that operate on distributed objects, named by OIDs. Atoms always complete due to their *liveness* property. Abort is only ever invoked from an application – an explicit user choice to cancel a groupware operation, for example.

Call	Description
<i>atom_begin</i>	Start an atom
<i>atom_end</i>	Try to commit the current atom
<i>atom_abort</i>	Abort the current atom

Table 4.3. The high-level Warp API.

4.2.4. The Differences between Atoms and Transactions

The serialisability and atomicity properties of conventional transactions do not in themselves offer any guarantees against the potential problems of *deadlock*, *livelock*, or *starvation*. Atoms augment these conventional properties with *liveness* and *fairness*.

Atom ID Generation

Atom IDs (AIDs) are globally unique and are used to support liveness, and fairness. A variation of Lamport's Clock Algorithm is used to ensure that the distribution of AIDs is

fair. Each object is associated with a timestamp. Each client process has a local logical clock which imposes a total ordering on its internal events. The local clock always ticks between any two events which are atom operations, initial accesses to objects and the creation of child processes. When a client process accesses the OM it sends its local time as an argument to the OM. The affected objects timestamps in the OM are set to be the upper bound of the current objects timestamps and the client processes local time. The OM then passes the upper bound of the objects updated timestamps back to the client process and the client process (the root of the atom) sets its local time as the upper bound of the returned timestamp and the current local time. For example, an atom issued by process P at time T has an AID PT. It follows that the number of clocks in the system gives a static bound on the number of atoms that can overtake a given atom. This represents bounded fairness.

The format of the AID is shown in Figure 4.8. In a network environment, a process can be uniquely identified on a node by a UDP port number and its host IP address. AIDs are mapped into multicast group addresses. All the OMs touched by a atom join the multicast group associated with the atom. This is used to conduct the row communications (see Fig. 4.3).

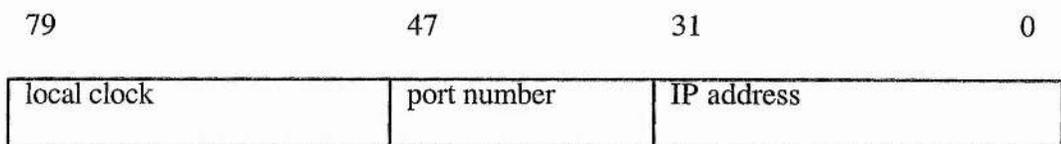


Figure 4.8: An Atom ID.

Liveness

Liveness means that provided an atom consists of code that would run to completion within itself, then it will run to completion even when in continual conflict with other concurrent atoms. In a concurrent execution environment it means that deadlock cannot occur and computational progress is guaranteed. Abort can only result from failure or an explicit invocation of `atom_abort`. In this setting, user abort appears as a backtrack to the start of the atom followed by immediate commitment. Atom identity ensures liveness.

When an atom releases a lock (object ownership token), it passes to the atom with the smallest AID amongst those waiting for that lock. Because an atom retains its AID until commitment, regardless of intervening backtracks, it is guaranteed commitment when it eventually becomes the oldest extant atom in the system. However, the AID ordering is not an enforced serialisation order, because an atom can commit whenever it has all its locks. This can easily occur before all older atoms have committed.

4.2.4.15 Fairness

Fairness refers to the absence of starvation. Even in a live system it is possible for slower processes to be starved by quicker, more aggressive ones. Warp gives preference to the oldest atom in cases of conflict. As atoms always complete and AIDs are totally ordered every atom will become the oldest at some point in its life and thereby acquire top priority in conflict situations. The older an AID gets, the greater its priority in conflict situations. An atom struggling to acquire enough resources to complete will eventually become the oldest in the system and win out in any conflict.

4.2.5. Resolving Conflict

Conflict between two atoms is resolved by backtracking one of the atoms to the last checkpoint preceding its touches of all the conflicted objects in combination with releasing locks. A curfew on lock holding by an atom that is unable to commit ensures freedom from deadlock. This is implemented as the *backoff* operation. Timeout values are set to reflect the type of application and its operating context. An RDG application will typically use values of less than one second. In contrast to conventional transaction systems, backtrack is graceful – an atom is backtracked just as far as is required to resolve the conflict. Moreover, even should an atom be backtracked to its start, this is not the same as conventional transaction restart. The atom retains the same identity, so its rerun is distinct from a new atom just entering the system.

A Simple Example

Figure 4.9a shows a simple example where two atoms, a_1 and a_2 , have been started at the same time by two independent, autonomous objects at Site 1 and Site 2. Initial ownership of an object, depicted by "*", goes to the first visitor. In this example object C is hosting both a_1 and a_2 . a_1 has reached object C before a_2 , so owns C, which has made a clone of itself for each atom. Figure 4.9b shows the corresponding conflict table.

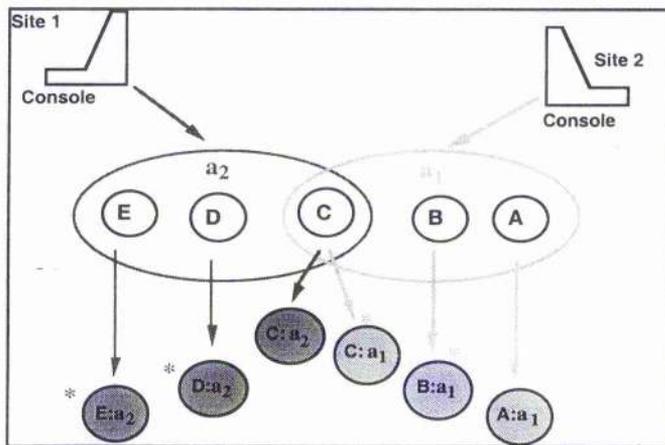


Figure 4.9a: Resolving a simple transaction conflict.

* indicates object ownership

	E	D	C	B	A
a_1			*	*	*
a_2	*	*			

Figure 4.9b: Conflict table for the simple example.

There are three possible outcomes in the illustrated scenario.

- a_1 completes before a_2 . Object A, the root of a_1 , knows this when it has obtained ownership of all the necessary objects. The clones $A:a_1$, $B:a_1$ and $C:a_1$ replace their masters. This invalidates clone $C:a_2$ and it must roll back to its start and be re-cloned from the updated C. Ownership of C then transfers to a_2 , which can run to completion. An atom cannot complete unless it owns all the objects it visits, but liveness ensures that all atoms complete eventually. Fig. 4.10. shows the progress of the conflict table when a new atom, a_3 , attempts to modify objects A, B, and C.

a_2 is now the oldest atom in the system, ordered according to AID, and wins out in the arbitration for ownership of C.

	E	D	C	B	A
a_2	*	*	*		
a_3				*	*

Figure 4.10: The conflict table when a new atom, a_3 enters the system

- a_1 backtracks to its start (i.e. aborts itself) and commits by not changing C. thus releasing ownership. In this case ownership transfers to a_2 and there is no need for the work done by C: a_2 to be invalidated because the state of C is unchanged. This shows that backtrack can be optimised in certain special cases.
- In a slight modification of the above example, imagine a_1 has visited, prior to C. an object F already owned by another atom that eventually commits. a_1 rolls back and ceases to be a visitor at C, whence ownership of C passes to a_2 . a_2 then owns a_1 its hosts and is able to commit. This outcome shows that even though a_2 is initially waiting on an older transaction at C, the eagerness of its clone may still be beneficial.

The Dining Philosophers

The Dining Philosophers problem (Dijkstra 1968) provides a good test of the liveness and fairness of concurrency control. Briefly, five philosophers spend their time thinking, getting hungry, eating, and then thinking again.

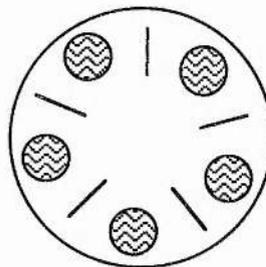


Figure 4.11: the philosophers table

The philosophers eat at a table laid out as shown in Figure 4.11. Each hungry philosopher requires two chopsticks in order to eat and they all need to finish their food before it gets cold. They never talk directly to each other. A good solution is live, fair, and makes good use of resources. The problem represents a situation where there are continual conflicting concurrent demands for multiple shared resources. It is a popular problem because real systems often have to function in environments where such conditions prevail. Further discussion of the dining philosophers and other concurrency control problems can be found in(Andrews 1991).

A well known solution (Tanenbaum 1987) using semaphores and shared memory is outlined in pseudocode in Fig.4.12. Each philosopher, *i*, executes a process *philosopher(i)*.

```
philosopher(i){
  do while (TRUE)
    think           // think for a random amount of time
    take_chopsticks(i) // get hungry, queue for chopsticks
    eat             // eat for a random amount of time
    put_chopsticks(i) // replace chopsticks, think again
  od
}
```

This can be programmed on Unix or Windows systems using the system calls provided to create and manipulate shared memory and semaphores, or with a high-level language such as SR or Ada. Shared memory is needed because all processes rely on access to their shared states. Semaphores are needed to enforce mutual exclusion to critical sections. There are two points to note about this type of solution: it is not easy to understand or program correctly, and it cannot easily be distributed using standard workstation software and languages because semaphores and shared memory constructs are not network objects.

<pre> main loop philosopher(i){ do while (TRUE) think take_chopsticks(i) eat put_chopsticks(i) od } </pre>	<pre> declarations semaphore condition[N] semaphore mutex := 1 int state[N] define EATING 2 define HUNGRY 1 define THINKING 0 define RIGHT (i+1)mod N define LEFT (i-1)mod N define N 5 </pre>
<pre> take_chopsticks(i){ down(mutex) state[i] := HUNGRY test(i) up(mutex) down(condition[i]) } </pre>	<pre> put_chopsticks(i){ down(mutex) state[i] := THINKING test(LEFT) test(RIGHT) up(mutex) } </pre>
<pre> test(i){ if (state[i] = HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) { state[i] := EATING - up(condition[i]) } fi } </pre>	

Figure 4.12: Dining Philosophers Solution using semaphores and shared memory

An application was produced using Warp atoms as follows:

```

philosopher(i){
  do true ->
    think()
    atom_begin()      // get hungry, get a new atom id
    get_chopsticks    // queue for chopsticks
    eat               // have chopsticks
    replace_chopsticks // replace chopsticks
    atom_end()        // end atom, go to back of queue
  od
}

```

This solution is easier to express than the one in Fig.4.12. The programmer does not have to know anything about the complexities of the coherence mechanism but should be familiar with the concepts of atomicity and serialisability. A screenshot of the application is shown in Fig. 4.13.

Why does the Warp version work? Each time a chopstick is requested by an atom a clone is generated to participate in that atom (Fig.4.14). There are potentially two conflicting atoms for each chopstick and therefore at most two clones operating on behalf of each chopstick. At commitment time, when the chopsticks are replaced and the atom terminates the clones associated with it also terminate. The solution is live and fair because only one atom can be the owner of a chopstick at a time (represented by "*" in Fig.4.14), and in cases of conflict the oldest atom involved is always preferred but the other atoms do not automatically abort – they know that they will eventually acquire ownership through the ageing process and wait.

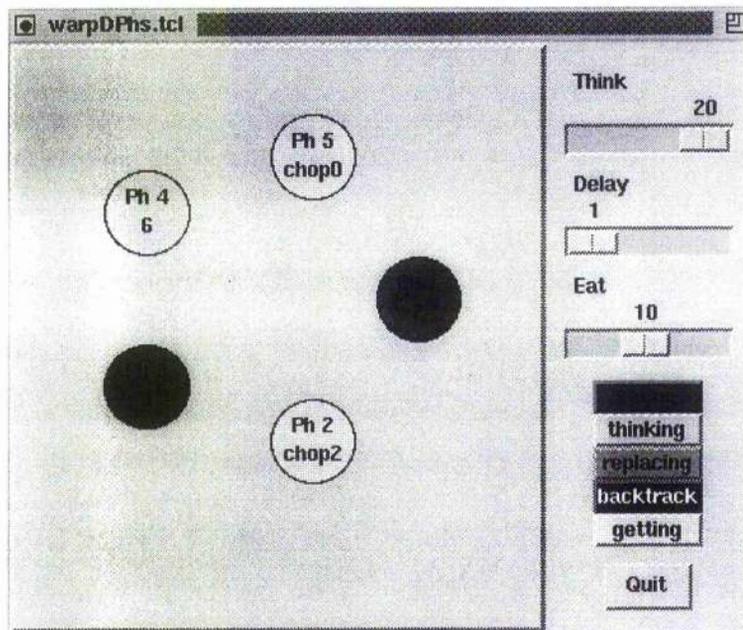


Figure 4.13: Screen shot of the philosophers

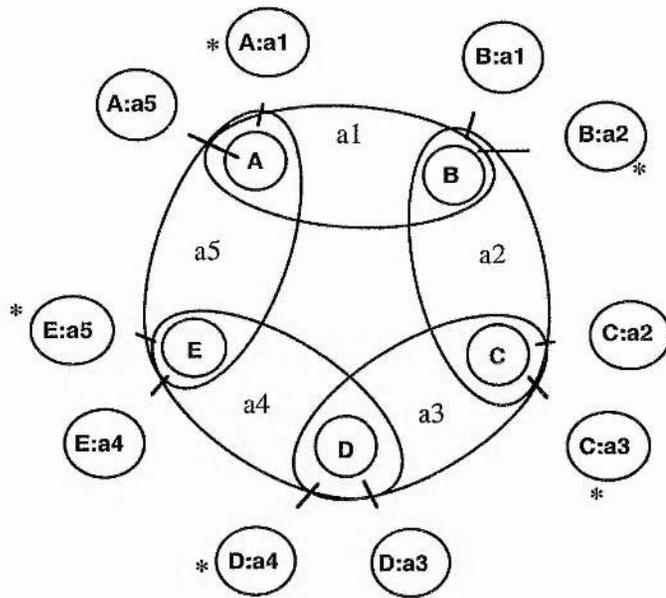


Figure 4.14

Assuming the starting pattern of ownerships depicted in Fig. 4.14 and an (unrealistic) lockstep series of state changes, Figure 4.15 illustrates the first few potential conflict table states in one run. The best performance that can be achieved is represented by always having two philosophers eating concurrently. This is indicated in the Fig. 4.15 by having two rows each with two asterisks.

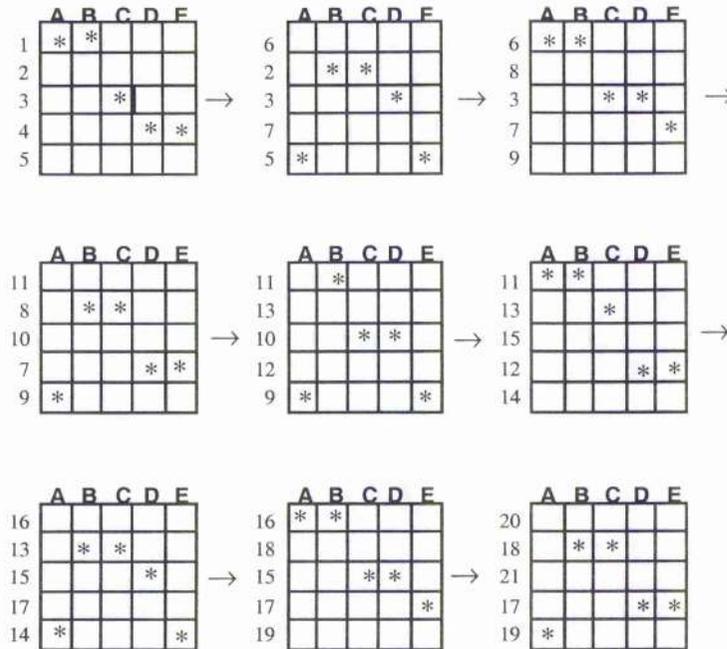


Fig. 4.15

A conflict table visualiser (CTV) was developed to actively monitor the progress of any application using Warp. Figure 4.16 shows a screen shot of the CTV observing the Dining Philosophers. The rows are indexed by AIDs and the columns by Objects.

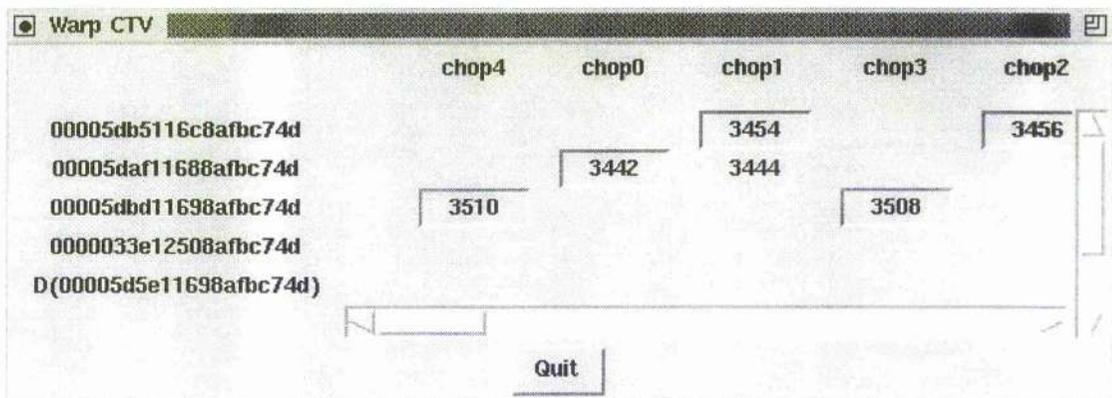


Figure 4.16: The Conflict Table Visualiser

4.2.6. The Warp Runtime Library

The runtime library consists of following modules: initialisation, checkpointing, object management, atom management and upcall.

Initialisation

The initialisation module provides applications with facilities to initialise the runtime library, to access the console and to clean up side effects on termination. The API for initialisation is shown in table 4.4.

Name	Description
warp_init	Initialise the runtime library
warp_exit	Clean up side effects and quit
warp_console_put	Display a string on the Warp console

Table 4.4. Warp Initialisation API

Checkpointing

The checkpointing module provides facilities to save consistent states of atoms. Atoms are checkpointed at each initial access to objects. This is essential to allow for backtracking. Checkpoints are carried out using the libckpt library (Plank et al. 1995). Checkpoints are associated with timestamps (based on local clocks which are synchronised using the clock algorithm). Backtracks involve reinstating a checkpoint associate with a particular timestamp.

Object Management

The object management module provides applications with facilities to create, read, write or delete objects. The API for object management is shown in table 4.5.

Name	Description
gcreate	create an object
mgcreate	create a group of objects
gset	read or write an object
mgset	read or write a group of objects
gunset	delete an object
mgunset	delete a group of objects
gstate	get the conflict state of an object

Table 4.5 Object Management API

Atoms are checkpointed at each initial access to an object. The last committed value of the object is fetched from the OM and a shadow object is created in the atom's address space. Subsequent accesses to the object are carried out on the shadow object. As OIDs are mapped into multicast group addresses atoms locate objects using multicast.

Atom Management

The atom management module provides applications with facilities to start, commit, abort and detect *anchoring* of atoms. Anchoring refers to the phase in an atom's progress where it has obtained copies of all the objects and ownership tokens it requires and then computes over them prior to commitment. Warp provides user control over when a transaction anchors. If successful it is then immune from further backtrack. The protocol for anchoring is the same as the commitment validation. An anchored transaction must not touch any new objects and an attempt to do so will raise an exception.

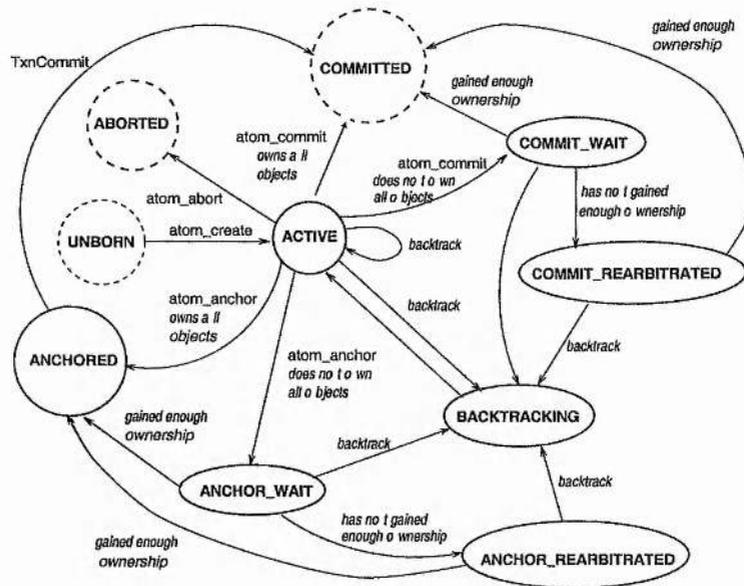


Figure 4.17: Atom State Transition Diagram

After being created an atom is in one of the following states: ACTIVE, BACKTRACKING, ANCHOR_WAIT, ANCHOR_REARBITRATED, ANCHORED, COMMIT_WAIT, COMMIT_REARBITRATED, COMMITTED, ABORTED. The state transition diagram is shown in Figure 4.17.

Commitment

When an atom runs to the end it tries eagerly to commit. The commit protocol (see Figure 4.18) is implemented in two phases: validation and commitment. In the first phase the atom checks with the OMs it has contacted to see whether it can commit i.e. whether it owns all the objects it has touched. If this is confirmed the commitment is validated and the atom initiates the second phase which causes the OMs to update all the objects modified by the transaction to backtrack all the losing transactions and clean up the CT. If the commitment is not validated the atom will sleep for a certain amount of time and retry. If it still cannot commit it re-arbitrates all the ownerships it holds then sleeps and retries again. This will repeat until the atom eventually commits or is backtracked. No atom is aborted because of conflicts. All of them will commit eventually (given that they would commit in the absence of conflicts). Atoms may wish

to abort themselves in certain circumstances however. For example, if an atom is executing on behalf of a user who changes their mind and wishes to cancel. When an atom aborts it gives up all the ownership tokens it holds and all the CT entries of the atom are destroyed. On its return the applications local state will be restored to the beginning of the atom and the atom terminates.

```

Xi = { m | m is an object that atom i has touched }

Si = { m | m ∈ Xi and shared with other atoms }

Oi = { m | m is an objects owned by atom i }

Omodi = { m | m ∈ Oi and modified by atom i }

If Xi ⊆ Oi Then

    atomState(i) := COMMITTED;

    ∀o ∈ Omodi, replace o with its shadow object;

    backtrack all other transactions which share Oi;
ELSE
    If atomState(i) = ACTIVE Then

        atomState(i) := COMMIT_WAIT;

    ElseIf atomState(i) = Commit_WAIT Then

        atomState(i) := COMMIT_REARBITRATED;

        ∀o ∈ Si ∩ Oi, rearbitrate ownership;

    End_If
End_If

```

Figure 4.18: The Commit Protocol

Upcalls

The upcall mechanism, `warp_set_probe (command)`, provides applications with facilities to monitor and change their runtime behaviour. The commands that can be probed are listed in table 4.6.

Command	Upcall Events	Command	Upcall Events	Command	Upcall Events
<code>atom_begin</code>	<code>atom_begin enter</code> <code>atom_begin aborted</code> <code>atom_begin leave</code>	<code>gcreate</code>	<code>gcreate leave</code> <code>gcreate recovered</code>	<code>backtrack</code>	<code>backtrack</code>
<code>atom_end</code>	<code>atom_end enter</code> <code>atom_end leave</code>	<code>mgcreate</code>	<code>mgcreate enter</code> <code>mgcreate leave</code> <code>mgcreate recovered</code>	<code>gunset</code>	<code>gunset enter</code> <code>gunset leave</code> <code>gunset recovered</code>
<code>atom_anchor</code>	<code>atom_anchor enter</code> <code>atom_anchor leave</code>	<code>gset</code>	<code>gset enter</code> <code>gset leave</code> <code>gset recovered</code>	<code>mgunset</code>	<code>mgunset enter</code> <code>mgunset leave</code> <code>mgunset recovered</code>
<code>atom_abort</code>	<code>atom_abort enter</code> <code>atom_abort leave</code> <code>atom_abort restore</code>	<code>mgset</code>	<code>mgset enter</code> <code>mgset leave</code> <code>mgset recovered</code>	<code>gstate</code>	<code>gstate enter</code> <code>gstate leave</code>
				<code>checkpoint</code>	<code>checkpoint enter</code> <code>checkpoint leave</code> <code>checkpoint recovered</code>

Table 4.6: Upcall Events

4.3. A Groupware Case Study: The Warp-based Shared Spreadsheet

Figure 4.18 shows the spreadsheet it in use between participants at five locations on the Fife and Tayside Metropolitan Area Network, in conjunction with Mbone audio-visual conferencing tools VIC and VAT. Multicast IP is used for both shared object synchronisation and continuous media streams. During these sessions the spreadsheet was used to develop an educational groupware application.

Each spreadsheet is identified by a user defined *sheet name* which is used as the prefix of the names of files that store the cell data and the OID of the root object. Each cell of the spreadsheet is represented by a Warp object. The root object stores the number of rows and the number of columns; the OIDs for objects hold information of the version

number, the list of names of columns and the list of widths of columns; and a matrix of OIDs for cell objects. All these objects form the *shared context* of the spreadsheet. Insertion or deletion of a row or column involves creating or removing multiple cell objects and modifying all objects related to the sheet structure – e.g. the root object, the list of column names and the list of column widths. In these cases all objects are bundled together for a single checkpoint.

4.3.1. Group awareness features

Group awareness is supported by both “out-of-band” communication channels, namely audio/video conferencing tools, and also by features of the application. The out-of-band channels started by application start-up script. These are tools which use the Real Time Protocol multicast over UDP/IP. It would have been possible to introduce other out-of-band channels such as chat rooms if required. It is also possible to envisage participants using a telephone conference facility from BT while running the core application. The desktop display each user sees is similar to Figure 4.18.

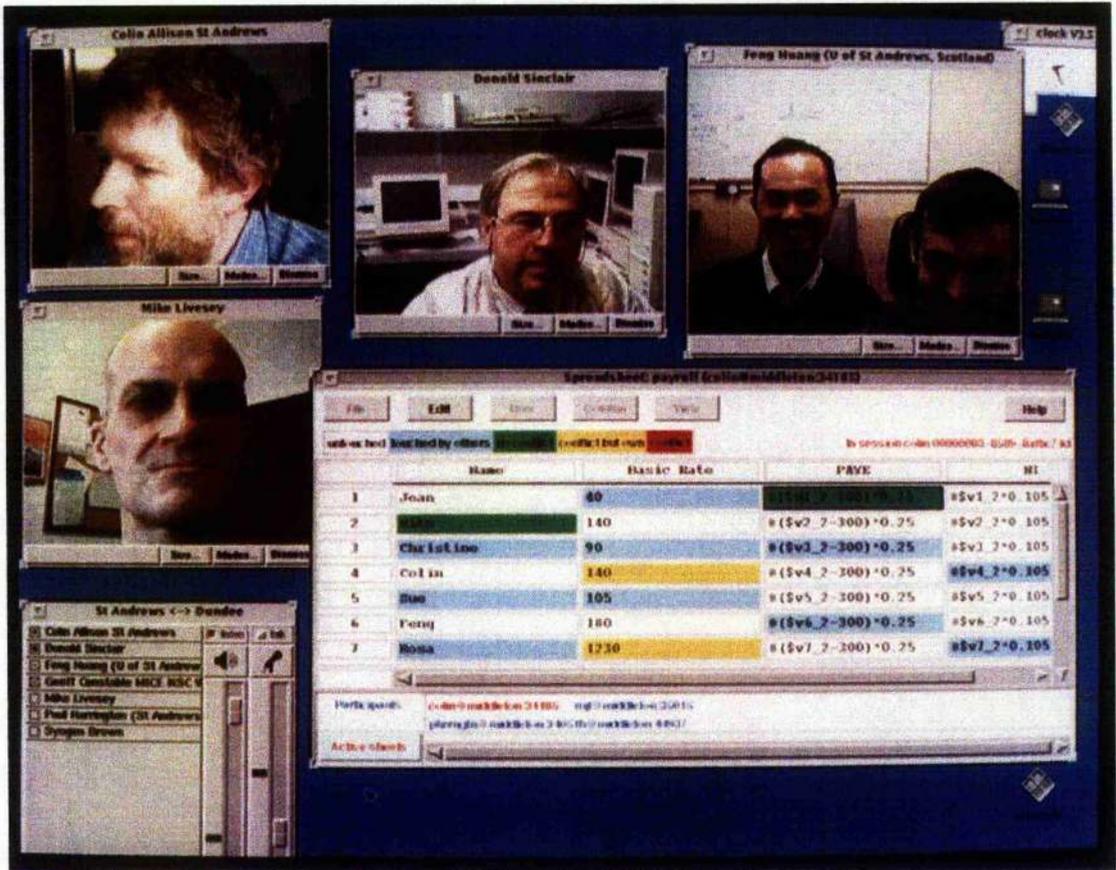


Figure 4.18: A shared spreadsheet development session

Each execution of the shared spreadsheet program is identified by a unique EID consisting of the user name, the host name and the port number. The title bar shows the name of the currently loaded spreadsheet and the EID. The graphic interface starts with a menu bar that enables the user to control the program. It is followed by a description of the colour code: *white* indicates that the object is untouched; *blue* shows that the object has been touched by someone else; *green* indicates that the user is the only one touching the object; *yellow* means that the user is the owner of the object but there are other users touching the object; and *red* shows that the user is in conflict with the owner of the object. Clicking on a cell/region that is potentially being modified by more than one user will display details of the other participants involved. It is then possible to use audio-video channels to discuss the situation with these other participants.

	Name	Basic Rate	PAYE	Net Pay
1	Tom Franklin	405	@\$v1_2*0.25	@\$v1_2-\$v1_3
2	Mike Livesey	700	@\$v2_2*0.25	@\$v2_2-\$v2_3
3	Paul Harrington	3050	@\$v3_2*0.25	@\$v3_2-\$v3_3
4	Colin Allison	4500	@\$v4_2*0.25	@\$v4_2-\$v4_3
5	Sue Wilson	1050	@\$v5_2*0.25	@\$v5_2-\$v5_3

The overall status of an editing session is displayed when a user starts editing a spreadsheet. A list of the participants in the collaboration is displayed below the spreadsheet. Clicking on a participant will reveal their personal information. Clicking on the "Active Sheets" button will expand the window to show all currently active spreadsheets. Clicking on a spreadsheet name will give a list of users who are currently accessing that spreadsheet.

4.3.2. The shared context

A shared context is a set of objects where the objects and the actions performed on them are visible to a set of users. In the shared spreadsheet case, the shared context consists of cell objects and a root object. All these objects are managed by the Warp system. A cell object represents a cell in the spreadsheet. The root object holds the following information:

- the number of rows
- the number of columns
- the OID of the version number object
- the OID of the object storing the column names
- the OID of the object storing the column widths
- the OIDs of cell objects

Insertion or deletion of a row or column involves creating or removing multiple cell objects and modifying all objects related to the sheet structure e.g. the root object, the list of column names and the list of column widths. In these cases all objects are bundled together as a single checkpoint to improve performance.

When refreshing the display, the Warp OMs are accessed to get an up-to-date copy of the shared context. This would normally involve a lot of context switches and data copying. In order to reduce these overheads, a version number is associated with each spreadsheet. When an atom commits, the version number of the spreadsheet is incremented. Each instance of the application caches a version number. When it tries to refresh its display, it compares the cached version number with the version number in the OM. The application gets the up-to-date shared context from the OMs when its cached version number is smaller.

4.1.3. WYSIWIS issues in the shared spreadsheet

Group interfaces differ from single-user interfaces in that they depict group activity and are controlled by multiple users rather than a single user. What You See Is What I See, *WYSIWIS*, is a well known approach. In strict *WYSIWIS* everyone sees exactly the same image of the shared context but in practice this can prove too inflexible (Stefik et al. 1987). Alternative user-centric approaches have been made such as flexible user interface coupling (Dewan & Choudhary 1995) and window-level sharing (Prakash & Shim 1994). The shared spreadsheet also opts for a more user-centric display and relaxes strict *WYSIWIS* in the following ways:

- Only the local cursor is displayed as multiple cursors are distracting.
- Uncommitted modifications are not automatically visible to other participants. This relaxation is quite natural in that a user would not normally want to make a modification visible until it was complete and committed, in a similar fashion to pressing the enter key to send a command from a terminal
- Committed modifications are propagated to other participants lazily. It is easy to identify key points where a display update is essential: firstly when a user starts a modification, and secondly when a user requests an updated view. When a user commits an atom the committed results are not propagated to another user until that

user next accesses the Warp coherence mechanism or initiates an operation that involves a display refresh. In addition, the display is automatically refreshed every ten seconds.

- User-centric colour codes are used to communicate the relative status of regions of potential conflict.

4.1.4. Editing Sessions

An *editing session* is the interactive view of an atom provided by the user interface. A user must explicitly start, end or abort an editing session. During an editing session, a click on a unit of the shared context, results in a *touch* on the object representing that unit. All of the objects touched in an atom form the *touch set* of that atom. They are guarded by the atom and are subject to backtrack. The interface allows a user to anchor an editing session explicitly, on the understanding that if the session subsequently touches any new objects the anchor is forfeit. It is reasonable to assume that, in the absence of backtrack, a non-interactive atom will terminate. In contrast, an interactive application comes with no such guarantees. A user might start an atom and acquire the ownership of some objects, then go for lunch or go on holiday! This would block other user atoms and prevent them from committing. To avoid such blocking, a *time-out abort* policy is adopted – if an atom receives no user input for more than a certain time, it is automatically aborted. At present no warning is given to the user, but a simple enhancement could consist of a countdown display.

4.1.5. Quality of Service Issues in the Spreadsheet

Interactive responsiveness is a key requirement for groupware systems. The performance of the spreadsheet was eventually tuned to acceptable levels for small group working. A particularly disappointing cause for concern was the failure of the fast (155Mb/s) ATM network to properly support IP multicast. Although ATM does support a multipoint mode, it transpired that IP multicast was being handled by a single processor on only one of the ATM switches involved in setting up virtual circuits to

carry IP traffic. This bottleneck occasionally lead to a complete system failure when the processor became overloaded with interrupt handling. Various schemes have been proposed since then for better handling of multicast IP on ATM networks.

Four other aspects of the Warp system were identified as contributing towards poor performance: checkpointing, context switching, display refreshing and the use of an interpreted language.

Checkpointing. libckpt (Plank et al. 1995) was used, which in turn uses the standard `write()` Unix system call. The initial implementation of the shared spreadsheet was slow when checkpointing to a network file system but by ensuring that checkpointing was done to a local system the response time was made satisfactory. Tests were carried out on an 350K process image. Checkpoints stored to a local disk took 0.3 seconds on a Sun Sparc 5. If they are stored on a remote disk via NFS it could add another 3 seconds or more. The best performance is obtained by checkpointing to local memory using the `tmpfs` virtual file system. This took under 0.1 seconds on the same systems. In practice either of the local options was acceptable.

Context switches. Each initial access to a Warp object takes two RPCs, one from the `front_end` to the `back_end` and the other from the `back_end` to the Warp kernel. The need to split the application into two processes comes from the X windows event model.

Display refreshing. For each display refresh, all objects in the shared context have to be copied to the text variables associated with Tk cells, one at a time.

Finally, the implementation of the Warp mechanism and the shared spreadsheet in Tcl/Tk and interpreted languages are generally slower than compiled native code.

4.1.6. The Shared Spreadsheet in an Educational Context

The Department of Accountancy and Business Finance at the University of Durdee teaches courses in Business Computing. These courses include content which places a significant emphasis on both computing and IT skills. The computing courses are taught

in tandem with mathematical methods for finance and accounting. A feature in the development of business computing courses is the move to integrate the accounting and mathematical methods courses with the computing courses. By the end of their second year students have had extensive practice in the use of spreadsheets to solve accounting and finance problems, and are familiar with techniques such as basic simulation. However, a genuine interactive business game cannot be developed using conventional spreadsheets because they do not support multi-user realtime concurrent access. This type of application is therefore very appropriate for evaluating the usability of the WARP-based shared spreadsheet. The specific requirement from an RDG spreadsheet in this context is to support composite, real-time simulations that amalgamate results from large numbers of individuals and show the time-dimensions of such amalgamations. This requirement suggested the development of an Interactive Business Game with multiple players. This would exploit the obvious benefits of viewing real interaction of inputs to such a game.

This project was an early example of co-operation between subject-specialists (Business Computing) and distributed systems builders (the Warp group at St Andrews). The Warp RDG environment was used for two purposes here: as a multi-user application development environment, and as the target system for the business game.

A business game was developed in which all participants ran their own businesses and were able to vary operational parameters for the duration of fixed trading periods to see how they performed, both with respect to their own variations, and with respect to each others attempts to win business. Each column in the spreadsheet represented a business and each row a specific operation parameter. Business owners could vary their own parameters but only see updates of others new values on fixed period calculations. At the end of each trading period (5 minutes) a variation on the weight of parameters was made across the board. These variations were calculated according to a formula

provided by the course lecturer, based on the theory of business model simulation that was taught in the course.

4.1.7. Lessons Learned

Trial sessions of the business game were successfully piloted by the developers, using the same wide-area multimedia environment that had been used to originally develop the spreadsheet. The Warp mechanism worked correctly throughout. There is still, to the best of the authors knowledge, no other product, commercial or otherwise, which provides such a high degree of support for multiple concurrent writers and readers. The major problem encountered when attempting to deploy the game in the teaching laboratories was that of *software infrastructure*. Dundee's student teaching environment was based on a mixture of NetWare and Windows NT, which did not support IP multicast, and also proved quite resistant to X-windows. Although these are problems that could be overcome given enough resources, it highlights a fundamental issue in educational groupware, namely that deployment is an issue, and by implication, a DLE must understand the infrastructure on which it runs. All students have access to machines that have a working browser simply because Web access, like the public telephone network, has become a "must". There is therefore a strong requirement on educational groupware to be web-friendly. X-windows and IP multicast are excellent systems, but they have not found the widespread acceptance that the Web has, and are accordingly not nearly as suitable for the deployment of educational software.

In summary, the Warp system made considerable progress towards demonstrating the potential of groupware in tackling education objectives, and the business game has the potential to meet several requirements of distributed learning environments: group-oriented, engaging and interactive, real world input through multi-user interaction driving the simulation.

5 Addressing the Framework Requirement

The Web has had a dramatic impact on the way we think about distributed systems, and this has been especially true in the use of groupware environments for teaching and learning. This chapter focuses on the design and implementation issues that arise in Web-based multi-user educational environments. In particular, it describes work carried out collaboratively between subject-specialists and systems builders as part of the Finesse and TAGS projects (Allison et al. 2000a, Allison et al. 1999, Allison et al. 2000b, Allison et al. 2001, Power et al. 1998).

It would have been possible to write this chapter slightly differently, as if the TAGS framework was the original focus of the work, and Finesse a DLE that was implemented under it. Indeed, Finesse is now a DLE which executes under TAGS. However, the historical order of events has been maintained. Finesse came first, and showed the need for a generic framework. There is an analogy to TCP/IP. Many people assume that IP, the core Internet Protocol, was developed before TCP and used as the basis for TCP's development. In fact, TCP was developed first, and showed the need for the underlying framework of addressing and routing provided by IP.

Finesse can be seen as a case study of a DLE featuring a subject-specific educational resource, so, after describing TAGS, a complementary case study is presented, featuring a generic resource for assignment tracking using group-based project management. Finally, the chapter concludes with a summary of issues encountered in the design and production of a framework for DLE construction and maintenance.

From WARP to Web

Warp produced an exemplar real-time distributed groupware application in the form of a multi-user, distributed, shared spreadsheet. However, as noted in the previous chapter, a new important requirement must be added to the original requirements list for this type of groupware, namely that it should be *usable on the Web*. The Web was quickly embraced as a new environment for groupware applications. Their growth began as soon as the pervasiveness of the Web made it clear that it could be a valuable tool to

facilitate the sharing of information among members of any group connected to any network running the Internet protocols. Of course, the main original purpose of the Web was the sharing of research results and documents among researchers and scientists (Berners Lee 2000). The growth of the Web as the preferred medium for group work has meant that, while the underlying principles of distributed groupware remain, there is now a clearly preferred platform for their implementation and deployment. Sikkel, Neumann and Sachwech summarise the advantages the Web offers to an application (Sikkel et al. 1998):

- Users throughout the Internet can share the application. They can use the same program in different places at the same or different time.
- The Web offers a truly cross-platform environment. This is important in distributed, remote co-operative work. The Web browser provides a common interface that keeps its appearance if the user changes platform. This implies a better acceptance of the tool by the user and a homogeneous interface the members of a group.
- The application is embedded in the usual working environment. Assuming that users are at a certain level of familiarity with the internet – Web access is available to every member on the group in their everyday working place – the Web-based application is managed and displayed by the browser, which is now a default piece of software on every computer. This results in a considerable shortening of the learning curve. Users do not see the application as "another" new, different piece of software that they must learn to use, but as part of the Web, which is an environment they are familiar with.

The design and use of groupware environments is radically impacted by both the explicit incorporation of pedagogic goals in system design and the nature of the Web. For example, the Warp version of the multi-user real-time business game was followed up by writing a Web-based version. This met the same functional requirements from the viewpoint of the subject-specialist, but was much easier to deploy. Although the Web-based version was an ad-hoc application it implemented many features that are common

to all Web-based educational groupware. These include maintaining student identity via authentication, providing global monitoring and configuration privileges for staff, and delivering appropriately tailored views to each participant in the multi-user environment. For example, students could see the results of the other participants' business plans, but not the parameter values that they had used in their business strategies. The general potential demonstrated by the Web-based version of the real-time business simulation inspired the development of a more substantial multi-user educational environment to meet a clearly perceived need in finance and business education programmes, namely the teaching of *fund management*.

5.1 Finesse: Finance Education in a Scalable Software Environment

The shared spreadsheet based business simulation demonstrated the potential of multi-user educational resources. Lessons learned from that project were used to inform the development of a shared, multi-user stock portfolio management facility for fund management education. Finance, Accounting and Management degree programmes typically include courses on fund management. In professional practice a fund is likely to have a spread of over 1000 stocks and be managed by a team. Team members typically specialise in certain market sectors. Finesse (Allison et al. 1999, Power et al. 1998) is an early example of a DLE (1996-) which caters for this subject area. Its design and construction brought to light core issues in the process of developing a Web-based DLE, such as role-based access control and view provision, and the management of users, groups and learning resources. Key features of Finesse include:

- a form of computer-based learning that could not function in a non-networked environment
- support for teamwork through various forms of conferencing, shared tasks and shared objects
- the introduction of real world input into the study environment

- exploitation of online working to monitor usage and then feed the information back into the educational process, on an individual basis.

5.1.1 Origins of Finesse

Finance lecturers at the University of Dundee had developed a portfolio management game for students based on a Lotus spreadsheet. The purpose of the game was to encourage students to use techniques and theories that they were being taught in their finance courses to create a virtual portfolio of shares. At the same time, the Glasgow Caledonian University also developed a simple paper-based portfolio modeling game for finance courses with the same objectives as the staff at Dundee. The specific pedagogical aims of these separate projects were:

- to force students to consider the trade-off between the risk and the return from investing in a portfolio of equities
- to demonstrate to students the difficulties of investing in equities by looking at issues such as the impact of transaction costs on portfolio profits
- to show students the profitability and risk of various investment strategies for a UK investor (such as opting for a portfolio of small company shares or for a fully diversified portfolio of shares);
- to enable students to investigate the benefit of including securities with low correlation in their portfolios and, in particular, to consider the benefits of emerging market investments.

Drawbacks identified in the way the original game was administered included the large cost in lecturers' time as each group's portfolio was created and updated manually. This method also severely limited the range of securities available for inclusion. The students' interest in the game was not sustained as portfolios were only updated once a month. Thus it was difficult for groups to respond to significant daily changes in the market. Dividends were not included, transaction costs were not calculated in a consistent or realistic fashion and bank interest on any funds not spent was not taken

into account. Lecturers found it difficult to evaluate the group decision making process, and assess individual contributions. Although the game was seen as “better than nothing”, it was not deemed to be reliable or transparent enough to be assessed and credited. Finesse was designed to overcome these limitations and improve the educational effectiveness and realistic nature of the portfolio game.

5.1.2 The Finesse Portfolio Management Facility (PMF)

The Portfolio Management Facility (PMF) is the core Finesse learning resource. It allows groups of students to manage a portfolio by buying and selling shares chosen from a database of live data¹ for approximately 1100 companies quoted on the London Stock Exchange (LSE). Monthly summary information dating back to 1990 is also provided for sector analysis. Groups are allocated a Portfolio with a starting balance of £100 million. The main Finesse menu is shown in Figure 5.1.

Groups are allocated with a *Notebook* resource, described later in this chapter, to communicate with each other and their tutor concerning their investment decisions. A view of a Notebook is available to tutors which allows queries on the usage of any particular instance. As the environment is Web-based it is useable on an “anytime/anywhere” basis, and teams can be formed from students who are geographically widespread. Students can only modify their own portfolios, and see their contents in detail. The overall performance of other portfolios can be seen, but not how it was achieved. In contrast, tutors can inspect any portfolio in detail and even modify their contents if necessary.

Tutors can :

- pick any portfolio to examine and apply any standard command
- credit and debit portfolios when events such as mergers happen and a stock disappears

¹ “Realtime” here refers to 20-minute old data picked up every 5 minutes.

- set which selection of shares and stocks are available for trading
- set what the initial capital available is
- set how transaction costs are calculated

Students can

- inspect changes and historic data in a range of companies
- buy and sell shares
- look at their portfolio profit and loss
- view audit trails of their portfolio by date, bank balance or security
- compare their portfolio performance with other groups (view rankings)
- be notified if other members of their team are currently working on the same portfolio
- follow links to other sources of market information

Money left unused is credited with interest based on rates that are downloaded daily. Transaction costs and share price-spread are explicitly set by the tutors, based on their expert knowledge of stock market operations. The formulae used in all calculations are maintained on a Web page available to all developers, providing the rationale for the values as they should appear to users. Fig. 5.1 shows some application screenshots.

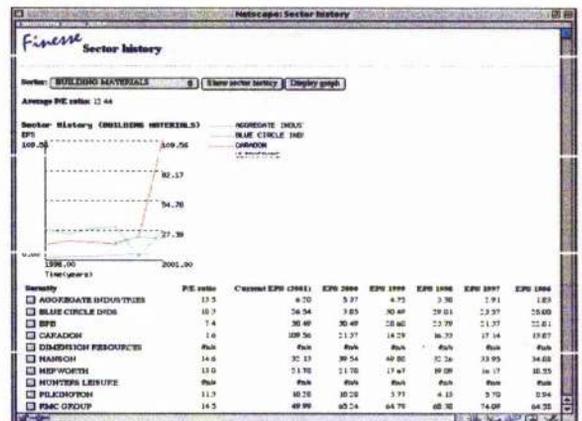
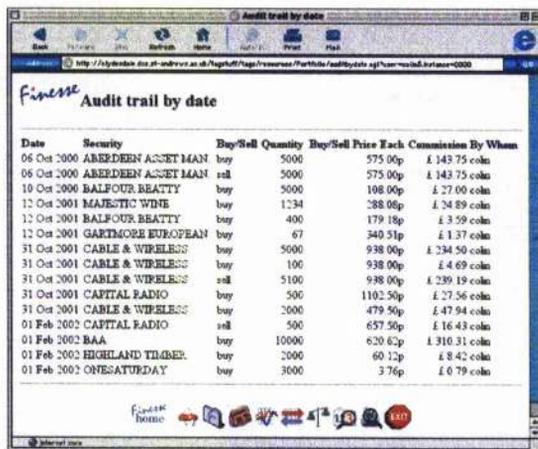
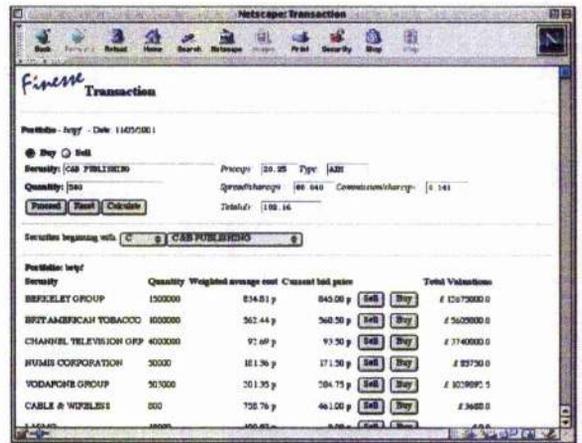


Figure 5.1 Finesse Screenshots: the Main Menu page, an Audit Trail by date, the Sector History page and the Transaction page.

5.1.3 Finesse Application Overview

There are broadly three ways of providing content on the Web: *static* pages, *dynamic* pages and *active* pages (Comer 1999). Static refers to HTML files which are simply downloaded from server to browser, dynamic refers to server-generated HTML, via CGI or Servlet generated for example, and active refers to executable code which is downloaded to the browser, such as Javascript and Java applets. It was decided not to use Java applets at the outset of the project in 1996 as there was no way of guaranteeing that the client machines, which could be any public access computer in a Scottish University, would be able to run them. Indeed, a survey carried out of known users

confirmed this. Similarly, it was decided to defer adoption of Java as a server-side language due to the poor performance associated with the early implementations. Accordingly, the design focused on a thin-client, heavy-server approach where most HTML was dynamically generated by CGI scripts on the server. The HTML often contained embedded JavaScript code, of a fairly lightweight nature. This approach had several advantages:

- As the target population was split across an unknown number of public access PC laboratories at three Scottish Universities, equipped and configured in an unknown manner, absolutely minimal assumptions were necessary regarding these client machines - simply that they had a working Web browser e.g. Netscape 2, IE 3 or more recent.
- All client/server interactions involved a CGI script on the server, providing an excellent basis for usage recording and detailed monitoring.
- As all interactions were serialised by a single Web server concurrency conflicts could be handled relatively easily.
- As all interactions were on the same server, the traditional ease of information sharing associated with single computers was possible. The multi-user nature of the system was emphasised to users by having displays of who was currently active.

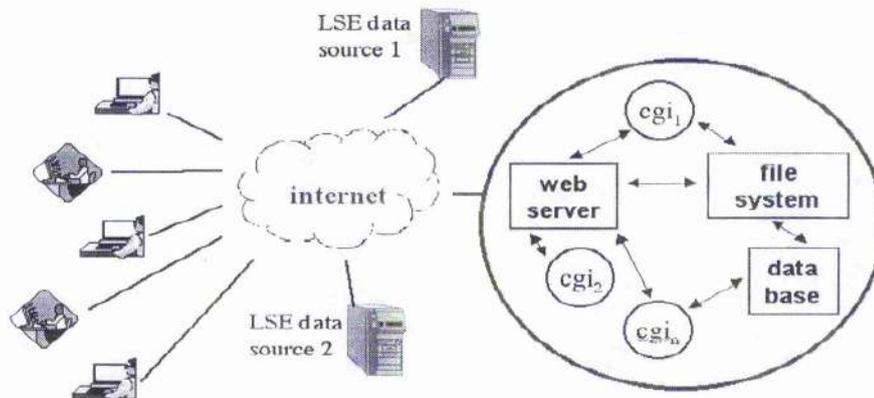


Figure 5.2: Finesse Application structure

Fig. 5.2. provides an overview of the structure of the Finesse DLE. The application was initially developed as a set of server-side CGI programs, written in a high-level

interpreted language, Tcl/Tk (Ousterhout 1994) and put into a shared library, summarised in Table 5.1. Some actions are limited to tutors. Early versions of routines to manage users and groups, and allocate portfolios, are omitted here, as they are later described in their more mature form, as part of TAGS.

Library Interface	Purpose	Tutor Only?
adjust_portfolio_values.cgi	used by tutors to adjust student portfolios in unusual situations	*
admin.cgi	general admin interface for tutors	*
aportfolio.cgi	displays a portfolio	
auditbybankbalance.cgi	summarises trades associated with a portfolio in terms of bank transactions	
auditbydate.cgi	displays trades within a portfolio, ordered by date	
auditbysec.cgi	displays trades within a portfolio, ordered by security	
create.cgi	creates a new portfolio	*
delete.cgi	deletes a portfolio	*
listlocks.cgi	lists any active locks on a shared portfolio	*
log_view.cgi	displays a users activity record	*
menu.cgi	builds a menu of options	
navigation_bar.cgi	builds a navigation bar	
noaccess.cgi	informs a user that they cannot access a resource due to lack of privilege	
performance.cgi	displays portfolio summary	
portlisting.cgi	displays portfolio listing by profit and loss	
pricelist.cgi	invokes scripts to display page allowing views of current stock prices	
sectorhist.cgi	allows user to select and display the history of particular stocks from particular sectors	
setdate.cgi	start accelerated historical mode from this date	*
transaction.cgi	Perform a transaction on the current portfolio	

Table 5.1 Finesse Library Interfaces

Two sources of stock information are imported regularly. The current share price information is obtained from Udata (UPDATA), a small commercial share price provider, every five minutes. This information is twenty minutes old, and is free. Every twenty four hours a set of statistics such as Price/Earnings Ratios are downloaded from Datastream (Bradstreet), a major financial data provider, and added to the share descriptions. Data is kept separate from code and was originally organised using the Unix hierarchical file system. The main structure is summarised in Table 5.2. The outline state of a portfolio is held in the accounts directory as three files: <portfolio

name>.log, <portfolio name>.por and <portfolio name>.quo. The complete state of a portfolio including stock values can vary with real time due to changes in the LSE data. User activity records are also held in /accounts, as <username>.log.

Data directory	Contains
accounts/	Portfolios, accounts and log files
chartgen/	files for sector history chart generation
datastream/	data received from Datastream
downloads/	.bat files to control downloads from Datastream and Updata
misc/	Miscellaneous
timing/	Specifies data pick-up times for Datastream and Updata
updata/	Contains data received from Updata

Table 5.2: PMF Data files

Data was later moved into a relational database. The library interface remained constant when the implementation was modified to access the database using SQL.

5.1.3.1 Group Awareness

Several features of Finesse contributed toward group awareness.

- Finesse created a multi-user environment somewhat reminiscent of a timesharing computer system. This was carried over to TAGS (See Fig. 5.3). Users can find out who else is currently using the system, and are automatically warned if anyone in their own portfolio group is active, as concurrent portfolio modification can lead to unintentional conflict. Finesse achieved this by the use of *cookies* (St. Laurent 1998). The protocol underlying Web traffic (HTTP) is stateless, so some work-around is necessary to maintain state across multiple independent http transactions. Each transaction between browser and server contains a cookie with a timestamp. This is noted and renewed by the server. If a specified period has passed since a cookie was received from the browser then the server may revoke authorisation and demand to see credentials again. Finesse was configured so that a default timeout of ten minutes inactivity would cause the student to be “logged out”.

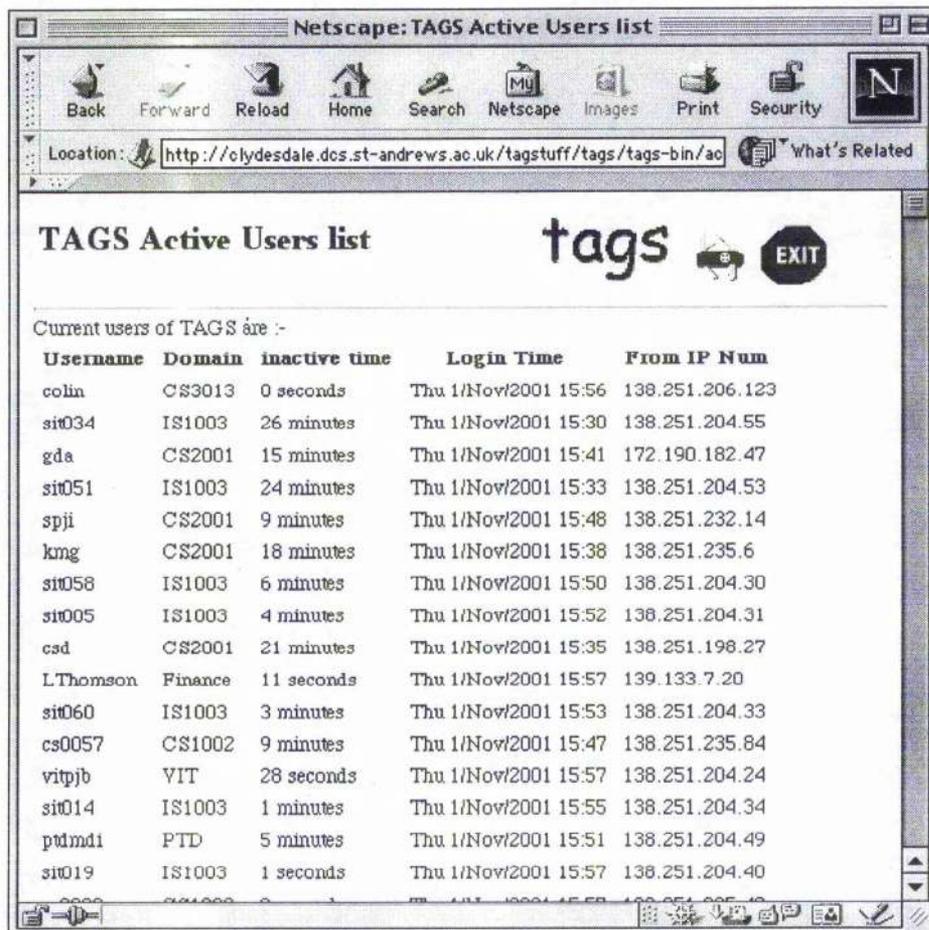


Figure 5.3: Finesse/TAGS Active User Display

- It is possible for a user to compare the performance of their own groups portfolio with all other portfolios in the system, including those belonging to staff and students from other institutions. See Fig. 5.4.

Portfolio	Interest/Dividends Received	Total Commission	Realised p/l	Unrealised p/l	Total Profit/Loss	Account Balance
tracker_flemings	£ 148,256.02	£ 587,091.04	£ -1,438,473.52	£ -6,584,768.8	£ -7,823,241.32	£ 214,489.95
xxxxx	£ 37,188	£ 0	£ 0	£ 0	£ 0	£ 1,037,188
tracker_blackrock	£ 209,467.38	£ 393,556.17	£ 0	£ -1,767,459.78	£ -1,767,459.78	£ 1,426,900.66
p_nicola	£ 135,447.46	£ 879,720.01	£ 4,024,387	£ -6,282,589.08	£ -2,258,202.08	£ 1,944,661.04
tracker_nicola	£ 143,004.91	£ 418,929.35	£ -7,382.3	£ 298,353.77	£ 290,971.47	£ 2,314,694.58
MARKET_MARVELSP	£ 342,952.43	£ 590,017.64	£ 92,940.96	£ -4,491,836.82	£ -4,398,895.86	£ 2,334,744.74
tracker_barings	£ 155,231.52	£ 367,740.59	£ 0	£ -5,389,241.55	£ -5,389,241.55	£ 7,852,401.44
tracker_alliance	£ 316,075.82	£ 603,065.85	£ -60,678.8	£ -1,197,001.21	£ -1,257,680.01	£ 8,569,124.71
Chancerpf	£ 475,626.26	£ 497,435.07	£ -135,765.29	£ -4,554,802.1	£ -4,690,567.39	£ 10,623,732.67
p_flemings	£ 240,926.99	£ 673,951.56	£ 1,630,944.86	£ -7,848,601.71	£ -6,217,656.85	£ 15,722,381.33
p_alliance	£ 268,089.14	£ 321,306.38	£ 0	£ -4,090,599.7	£ -4,090,599.7	£ 20,843,526.55
Port10	£ 809,290.91	£ 393,551.45	£ 0	£ -19,763,147.89	£ -19,763,147.89	£ 2,170,467.97
tracker_jupiter	£ 241,866.64	£ 311,730.33	£ 0	£ -4,638,520.51	£ -4,638,520.51	£ 2,199,593.43
Indifferentpf	£ 884,655.25	£ 1,074,873.56	£ 2,387,398.88	£ -4,883,458.91	£ -2,496,060.03	£ 22,586,183.41
p_blackrock	£ 429,634.56	£ 402,085.84	£ -125,534.99	£ -3,113,922.41	£ -3,239,457.4	£ 23,535,317.45
Port11	£ 944,069.84	£ 1,136,941.98	£ 2,700,350.59	£ -18,403,409.75	£ -15,703,059.16	£ 25,492,111.86
p_soros	£ 1,790,379.88	£ 1,212,086.28	£ 2,790,452	£ -29,415,280.79	£ -26,624,828.79	£ 25,554,104.66
Port8	£ 915,719	£ 2,109,384.79	£ 4,902,044.59	£ -7,720,069.77	£ -2,818,025.18	£ 26,068,363.52

Figure 5.4: One of the View Rankings display options

- The Notebook Tool, described in the next section was used as an asynchronous group awareness facility.

5.1.4 The Finesse Development Process

The move to the Web meant that the development process itself became open to all team members, both the software developers and the subject-specialists. In the early days of the project face-to-face meetings were held which usually ended up with a group of people gathered around a computer screen, looking and commenting. These meetings were supplemented by phone calls, e-mail, faxes and snail mail. The systems builders were keen to introduce multicast screen sharing sessions, e-mail distribution lists and a local UseNet newsgroup – but the educators were worried about the learning curves of these types of computer-mediated communication traditionally used by computing specialists.² Accordingly, a simple Web-based notebook tool was implemented to facilitate communication inside the development group.

² Although this was before Netscape and Explorer offered integrated GUI interfaces to News Groups, it is doubtful that the subject-specialists would have been motivated to adopt these facilities had they been available.

5.1.4.1 The Notebook Tool

The Notebook is illustrated in Fig. 5.4. It is a lightweight groupware tool providing some of the functionality of e-mail and a bulletin board. It interfaces to conventional e-mail if required. The Notebook application works as a CGI program. It manages a collection of messages sent to it by members of the group it belongs to. There are both "push" and "pull" options with respect to its interface to e-mail. That is, for any given note the submitter can choose to alert the entire group by e-mail (push). Reciprocally, an interested party may choose to be notified by e-mail of all new notes posted, regardless of whether they have been explicitly pushed (pull).

The design strategy for the Notebook was that new features would be added only if requested and they did not involve an increase in the complexity of the user interface. The Notebook has some similarities to e-mail and bulletin boards but minimises application context switching. To this end every page in the prototype Finesse DLE had a link to the developers notebook. It made commenting easy and intuitive. The notion of unlabelled notes seemed to strike the right chord as people did not feel they had to think of a Subject or Thread or Category before communicating. The shared nature also helped to create a sense of online community - something quite familiar to software developers with access to News groups and e-mail lists, but quite foreign to the subject-specialists from Finance and Accounting.

The Notebook tool revealed itself as a very useful alternative to e-mail and other conventional ways of communication when it came to matters related directly to the use of the Finesse as groupware environment. In this case, the critical mass described in (Grudin 1997) was clearly reached. The success of this tool is largely based on the simplicity of its presentation to the user, giving more importance to the information shared than to its structure or classification.

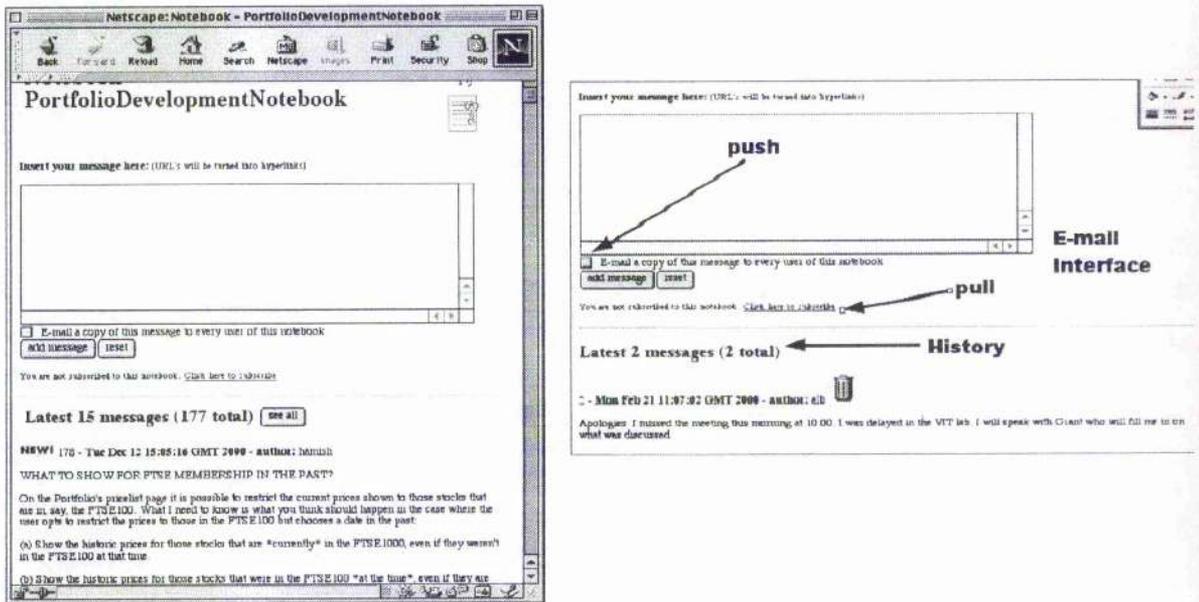


Figure 5.4: The Notebook Tool

A Notebook is accessed through a browser by accessing the URL of the Notebook application. Once accessed, the application displays the contents of the Notebook, as shown in Fig 5.4. The HTML page returned contains an empty text area for users to type new comments and send them to the notebook. The messages are free format. There is no requirement for a subject or any other information. A reply-to address field is supplied but it is optional for the user to fill it. When the message is submitted the system gives a reference to it so that users can refer to it when communicating with other members of the group. The messages are shown with its reference, author, reply-to address, date, time and an automatic label identifying those messages that have arrived recently to the system. Messages can be deleted by its sender or by the administrator of the system. The list of messages is shown in inverse chronological order so that most recent messages appear on the top of the list. When submitting a new message, the application gives users the option of sending an e-mail notification of the message to all the members of the group. Additionally, members of the group can "subscribe" to the notebook, in such a way that they receive an e-mail copy of every message sent to the subscribed notebook. To study the use of this tool, the application keeps a shadow list of the comments with information about the operations done to them: creation, deletion and so on.

The Notebook was sufficiently popular that it was adopted as a tool for student/tutor, and student/student communication. It was accordingly added to the Finesse DLE along with the Portfolio Management Facility. A Note Admin facility was created at a later date. This allows tutors to extract information about Notebook usage to aid assessment in the groupwork environment. Queries such as who wrote notes, how often, when, how long etc. are supported.

5.1.4.2 Discussion: Web-based CSCW

Computer networks in general are a primitive form of support for Computer Supported Co-operative Work (CSCW), as they facilitate work between people who are using different computers. CSCW has grown in use and sophistication with the Internet. The goal of CSCW is to provide computer-based means of communication and interaction to working teams. Nevertheless, CSCW software has turned out to be sometimes an obstacle, rather than an improvement, in the real, everyday working life of the end users. This has resulted in some cases in ambitious projects that are only used sparingly, if at all, by the expected users. Similarly, relatively few features of a system are perceived as helpful and are adopted into the environment of the user. The remaining features remain unused. This failure of CSCW software to meet users' requirements has been (and still is being) studied by CSCW designers and researchers. Research strongly suggests that designing for groups is much more difficult than designing for individual users. Group requirements are harder to predict, and groups change with time. Further discussion about features of design for groups support can be found in Mandviwala and Olfman (M. Mandviwala 1994) and Grudin (Grudin 1997).

There are a large number of CSCW tools available for use on the Web, and these tools take advantage of the features the Web provides: cross-platform; easy, known user interface; no need for new software in the user machine in most of the cases. All these characteristics have an undeniable attraction for CSCW designers. The challenge for these new Web-based CSCW tools is coping with heterogeneous groups of users with different interests and backgrounds and widely differing degrees of computer literacy. The problem is that a large number of new users of these tools have discovered the

Internet before any other CSCW environment. These users find traditional Web-based CSCW tools (HyperNews, Notes) difficult to use, as they need to learn extra skills to use them effectively. Moreover, most of them are reluctant to change their usual way of working, as all the CSCW software (as any other software) impose its own way of approaching tasks, and creates a new regime of action. An analysis of this imposition of work practice by CSCW software designers is found in (Hales 1994, Suchman 1993). Experience with Finesse has shown that an approach based on "weak" software such as the Notebook described in the previous section is more likely to be successful.

5.1.5 Evaluation of Finesse

The evaluation of Finesse focused on three main areas: the software, the educational content and the integration and use of the software in the teaching process. These areas were selected because (i) it was felt that they were the main focus of interest for those involved in the project including the finance lecturers, the systems builders and the students; and (ii) no one evaluation method would be sufficient for a project with a variety of different stakeholder groups. Each of these groups had their own set of objectives for the project, and to try and evaluate Finesse against these different objectives using a single approach would be difficult, if not impossible. Therefore a variety of evaluation methods were chosen; both qualitative and quantitative approaches were adopted for completeness. Teaching staff identified the educational goals and drew up the functional specifications; external experts in both the computing and finance fields monitored quality of content; dissemination via reports for fund-holders, papers and presentations to different sectors of higher education provided peer response and feedback; pre and post questionnaires were employed for student learning evaluation; lab-based sessions provided useful tests of software robustness and illuminated better interface design. The evaluation process and results is reported in detail in (Helliard et al. 2000).

5.1.5.1 The Formal (Questionnaire-based) Evaluation

A two-questionnaire strategy was used to evaluate Finesse in 1999/2000. A pre-questionnaire was distributed at the start of the course before students were introduced to Finesse and a post-questionnaire was employed at the end of the course once the game was finished. Each questionnaire contained a number of statements about which the students were asked their views; a 5-point Likert scale was employed where a 1 indicated "total disagreement" with the statement while a 5 suggested "complete agreement". The statements were grouped into four main areas, two of which were common to both questionnaires. The initial section was different between the two questionnaires.

In the pre-questionnaire, the initial section solicited students' views on working in groups, using computers and surfing the net, in order to gauge whether these essential pre-requisites for the game were in place. In the post-questionnaire, the initial section investigated whether students had enjoyed Finesse, found it easy to use, benefited from its notebook and e-mail features, and grasped theoretical concepts which had been discussed in lectures more quickly because of taking part in the game.

The second section sought information on students' familiarity with certain topics that were covered in the course and employed in Finesse. For example, the difference between dividend income and capital gains, the nature of investment trusts, familiarity with market indices and an understanding of the distinction between AIM stocks and Main Market shares were included.

The third section of the questionnaire focused on students' views about the benefits of group work and asked respondents if such work improved writing, presentation, communication and recognition skills. It also investigated whether students believed that collaborating in groups would improve their marks in general.

The fourth section was only included in the post-questionnaire and asked the students for general comments on the game. Finally, both questionnaires were piloted on staff and reworked several times as a result of the feedback provided. However, it was decided not to test them on students because such a strategy would have reduced the

already small sample size available. In addition, the questionnaires were completed anonymously to allow students to express frank and honest views without fear that any adverse opinions might effect their overall mark for the course. Of course, the authors are aware that, despite these precautions, the questionnaires were subject to all the usual limitations of such research instruments. Nevertheless, it was felt that the views of the students would provide useful feedback on the operations of Finesse in 1999/2000.

5.1.5.2 Results

The pre-questionnaire was administered to the 4PM class in October 1999. A number of points emerge from an analysis of this Table. First, the students appear to have the necessary background to undertake the course; the average scores in Section A of the questionnaire indicate that respondents did not like to work alone, enjoyed using computers and did not see any difficulty with using the Web. In addition, they regularly read the financial press and believed that other course material would be useful in studying for the 4PM course. Second, students indicated that they were familiar with different share selection strategies, aware of transaction costs, knew the difference between dividend income and capital gains, understood what was meant by investment trusts and aware of the stock market indices; in each instance, the mean score was greater than 3.000. However, with the exception of familiarity with transaction costs, the p-values were all greater than 0.050 indicating that none of these mean scores were significantly different from the neutral value of 3.000 on the Likert scale.

Third, the responding students seemed positive about the potential of group work to improve their skills. The mean scores for improvements in presentation, communication and negotiation skills associated with group work were 3.65, 4.273 and 4.35 respectively and all the p-values were less than the critical value of 0.050. The average rating for the other three statements in this section of the pre-questionnaire were all close to 3.000. The high standard deviation in response meant that the null hypothesis that the average was equal to the neutral score of 3.0 could not be rejected at conventional levels of significance.

The post-questionnaire was distributed to the class at the end of the course in April 2000 and indicated that students benefited from using Finesse. Statements about enjoying the portfolio game, finding Finesse easy to use and improving a student's ability to work as part of a group all had high average scores. Features of the game such as the Notebook were also seen as helpful. Surprisingly, face-to-face meetings had a higher average score than an e-mail facility in terms of helping students with group work. However, because students met each other every day in lectures and tutorials, a reliance on electronic communication may not have been vital. Also, there was a very high standard deviation of responses for both of these statements indicating that there was a range of opinion on this issue among the students questioned.

An analysis of the second section in the post-questionnaire reveals that student familiarity with financial concepts and financial terms had improved after the course and once the game was finished. With the exception of capital gains and AIM stock, there was a general agreement that students were more familiar with the main aspects of portfolio management. Perceptions about the advantages of group work also remained strong according to the post-questionnaire results. In the third section of the questionnaire, students' agreement with statements that group work improved presentation, communication and negotiation skills remained strong.

5.1.5.3 Qualitative Responses

Comments were invited in the fourth section of the post-questionnaire on any features of Finesse which the students found helpful and on how the computer-based system or game could be improved. The students who responded to this section liked the ease with which the game could be used, the "ability to try strategies without risk", the buying and selling at up-to-date prices and the realism of Finesse. One respondent argued that the game demonstrated "that it isn't easy to make good profits every time you invest". Some liked the practical aspects of Finesse such as "putting theory into practice" and enabling them "to use the information that we have learnt during the past 4 years into practice". A small number of negative comments were received. Responses to other questions concerning the management of group activities underline some of the findings

above; students organised regular and frequent face-to-face meetings in general, though one commented that meetings were "Ad lib, usually through notebook". Overall, the views in this section were supportive of Finesse and indicated that the educational goals of the course had largely been achieved.

5.1.6 Abstracting Generic DLE Features from Finesse

The construction and deployment of Finesse as a working DLE brought to light key potentials and functional requirements in Web-based DLEs. These included:

- The educational potential for network-based learning. The Portfolio Management Facility illustrates a type of educational resource which could not exist in a non-networked environment
- Inclusive Software Development. A development process which featured subject-specialists and software developers working together through the same collaborative medium as the deployment target, which in itself was collaborative.
- Collaborative Working. A collaborative environment for group-based learning was produced. This supported student team-work and group-based management of the learning activities by tutors.
- Cross institutional sharing of online teaching and learning resources
- The constitution of a DLE: subject-specific and generic types of shared, multi-user, interactive educational resources complementing each other
- Roles. Clearly different roles of *tutor* and *student* necessitating different access privileges and functional interfaces to the same resources
- Monitoring. Techniques for monitoring students' use of resources
- Responsiveness. In the course of developing and deploying Finesse the key QoS issue of delay was encountered. A simple but useful analytical model for measuring Web delays as experienced by users was devised which enabled productive decisions to be made about reducing the delay (see Chapter 7).

- Security. Several aspects of providing an online environment necessitate security policy and implementation. Role-based access and the privacy and integrity of students work and records stored online. Assessment of online work requires a guarantee of identity, hence the need for authentication.
- The need for management facilities to organise users into groups and allocate them educational resources.

The next section describes how these features were used as the basis for building a generic framework for constructing Web-based DLEs.

5.2 The TAGS Framework for DLEs

Finesse highlighted key issues in DLE design and construction. TAGS (the Tutor and Group Support scheme) used the experience gained from Finesse to inform the design of a *framework* for the research, development and deployment of DLEs. TAGS progressed by maintaining two versions: a production service (which is in daily use at time of writing) and a research version where new ideas can be implemented, shared and tested.

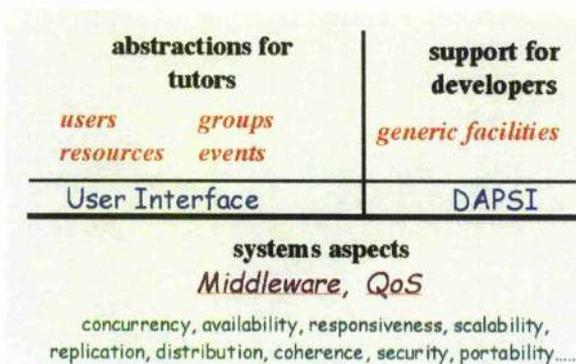


Figure 5.5a:
Overview of TAGS Framework

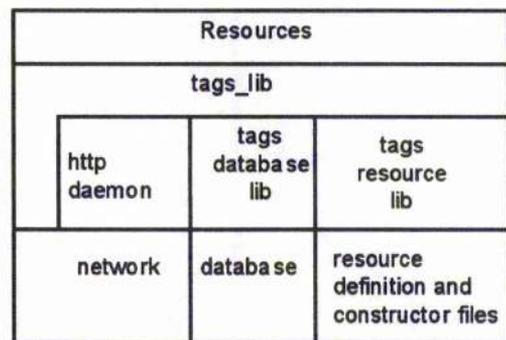


Figure 5.5b: Implementation Structure

The TAGS production service has been used in accredited degree programmes in eight subject areas at six Scottish Universities. The experience gained from running a real service has proved invaluable in that the project has identified and addressed issues in usability, security, responsiveness, concurrency control, availability and overall Quality

of Service (QoS) that would not have arisen in a pure research environment. Figure 5.5a shows an overview of the TAGS framework, figure 5.5b shows the components used in implementation and how they are organised relative to each other. Like Finesse, core TAGS functionality uses a thin-client, heavy server, structure based around a set of CGI programs which provide a runtime library for services and resources. TAGS_Lib, summarised in Table 5.3, provides the core API and services for TAGS resources and the framework. It in turn makes use of two private libraries providing interfaces to (i) a relational database, ii) files associated with resource management; (iii) the Apache Web server (http) daemon, and (iv) the network directly (for tasks associated with remote resources and authentication authorities).

5.2.1 Users, Roles and Groups

Much of the utility of TAGS comes from its strong support for group-based working, which has in turn resulted in the group ethos permeating the system. The concept of the *group* is central to TAGS, where it is used to support many functions, including:

- role definition
- privileges and access control
- information dissemination
- event awareness
- teamwork involving shared, multi-user educational resources
- the management of online collaborative learning
- user-centric portal generation
- collaborative resource development

The key roles supported by TAGS are tutors and students. The framework also offers support for the more specialised roles of resource developers, course administrators, system administrators, and service providers. A particular resource may export different interfaces to arbitrary roles, via the group mechanism. An educational resource, such as

the Finesse PMF for example, may distinguish the role of “lender” or “banker”. However, all users must belong to one (or more) of three base groups: student, tutor or system administrator. In practice most resources will distinguish between the views and interfaces they provide to students and tutors.

“Tutor” refers to anyone involved with the management of educational processes or the delivery of educational content. A key role for TAGS is to provide useful abstractions for tutors to work with. The basic abstractions provided for tutors are *users*, *groups*, *resources* and *events*. One could imagine providing tutors with further abstractions such as “tutorial group”, “course”, “class”, “assignment” and so on, but this may unreasonably pre-empt decisions on the structure of a DLE that should be made by tutors for themselves. There is considerable diversity in higher education and such preemption is likely to be seen as a limiting imposition rather than as an enabling technology. The notion of an *assignment* for example varies enormously across subject areas and institutional boundaries. It is therefore arguably more productive to identify and provide basic building blocks with which tutors can construct their own DLEs to suit their preferred styles of administration and meet their own pedagogical goals.

In practical terms, tutors construct and maintain a DLE by using the Users, Groups and Resources (UGR) management tool. Tutors may create users who are either students or tutors, and resource instances of any type available in a particular domain. The UGR tool allows the creation of arbitrary relationships between users and resources, using groups as the basis for the mapping. Users and groups are unique by name; resources are unique by name and type. Access rights can be specified when a resource is allocated to a group. A resource may simply be distinguished as Read-only or Read-Write, or it may export a more subtle set of access methods. Figure 5.6 illustrates an example set of relationships between users, groups and resources.

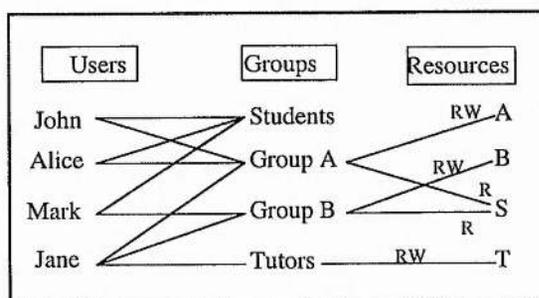


Figure 5.6: Basic Relations

John, Alice and Mark are members of the Students group. There are no resources shown allocated to the Students group. Jane is a Tutor and has created Groups A and B. John, Alice and Jane are in Group A and have Resources A and S allocated. S is allocated on a Read-only basis, whereas A is Read-Write. Mark and Jane are in Group B, which allows resource B as Read-Write and S as Read-only. When a user is a member of multiple groups who have different access privileges to the same resource then they are credited with the highest level from their set of privileges.

Tutors are free to create students, groups and resource instances, as described above, but normally can only be created by *system* or *domain* administrators. Domains are described later in the Chapter, as a feature to support service provision. Administrators privileges are similar to that of a super user in a timesharing operating system such as Unix, or a database owner in a database system such as Ingres.

5.2.2 Events

The *event* abstraction allows tutors to review the use of the DLE through two methods: the direct inspection of log files, and by setting predicates and actions that the system will manage. Actions are usually e-mail notifications of events such as "user U has not used the system for over a week" or "tutor T has not completed the marking of assignment 3 by the due date".

The system tracks general information such as login times, source addresses and resources accessed. Resources do not have to issue resource-specific events, although

they are encouraged to do so³. Each resource can generate an arbitrary number of different types of events. Event type is relative to resource type. Examples of event types are: logging in, posting a message to a notebook, buying shares, downloading a file from a shared workspace. Each time a user generates an event, an event instance is stored.

Event types are defined by the following fields:

- **name** - name of the event (unique for the resource type)
- **resource** - name of the resource type that generates this event (if the event is a system-wide event, the 'resource' is "system")
- **default_ttl** - default time to live of this type of event (to aid garbage collection)
- **extra_fields** - list of pairs (name, type) describing the extra information contained in the event. This information is resource and event dependent.

An event instance is stored in a table unique to a resource instance. One or more tables may then be queried by search scripts. Rules and searches are stored in a table with the following fields:

- **rule** - boolean: yes -> rule / no -> search
- **creator** - user that sets the rule
- **user** - user to be monitored / any
- **group** - group to be monitored / any
- **resourcetype** - resource type generating event to be searched / system / any
- **eventname** - event name to be searched / any
- **instance** - resource instance that must be monitored / any

³ One of the most widely used resources, an assignment.tracking tool, provides its own event and action subsystem.

- **predicate** - string in the form: name operator value, where name is a name of the resource specific fields of the event type and operator is one of == != < > <= >=

Rule-only fields

- **threshold** - number of times the event must happen to activate the rule
- **time_period** - Time period during the threshold number of occurrences must happen. Its value is <n><units>, where n is an integer > 0, and units is one of, "day", "week", or "month".

Search-only field

- **next_evaluation** - time (number of seconds) when the search will be repeated.

Rules and search scripts are generated from Web-based forms, intended to make the process of setting up event monitoring user friendly for tutors.

5.2.3 Support for Developers

In contrast to learners and tutors, developers are computer literate to some extent. They are the people who develop and maintain resources. It is important to let people play to their strengths, and in the case of developers this means allowing them to focus on providing core functionality for their educational resources without worrying about deployment, distribution, and management of a resource. These types of needs are generic and a collaborative learning environment can provide a useful set of services and interfaces for developers. Generic models for distribution, for instance, allow a developer to choose the conditions under which their resource can be made available, replicated for performance or fault tolerance, copied, re-used, and accounted.

The developers and application programmers service interface provides educational resource developers with commonly needed system services thereby allowing them to concentrate on content. A developer can use any of the following services for their resource:

- access control (by role)

- view (by role)
- generic user event logging
- resource-specific event logging
- obtain the groups and membership lists for this instance of a resource
- notify the group or groups associated with this instance of the resource

These facilities are implemented by TAGS_Lib.

5.2.3.1 TAGS_Lib

The main TAGS library is implemented as a TCL package consisting of several other TCL packages. A resource can also present its own set of access methods. Table 5.3 shows some of the component packages. The domain package is described later in this chapter.

Package	Methods
resources.tcl	CreateResource {resname} {restype} {creator} {data} DeleteResource {resname } AddResourceToGroup {res grp } {privs } RemoveResourceFromGroup {res grp}
groups.tcl	CreateGroup {grp} DeleteGroup {grp}
users.tcl	AddUserToGroup {usr grp} RemoveUserFromGroup {usr grp } CreateUser {usr password realname timeout email groups creator} DeleteUser {usr force } ChangeUserDetails {usr password realname email timeout} GetUsersGroups {usr} ExistsUser {usr } GetUsersDetails {usr} GetUsersAccess {usr} <i>Returns a list of resource names available to the user</i> ListUsers {} <i>Returns all the users in the system</i> BelongsToGroup {usr grp} IsStudent {usr} IsTutor {usr} IsDomainController {usr} IsSysAdmin {usr}
objects.tcl	CanDeleteObj {obj typ usr} <i>Determines if a user is allowed to delete a object of type user/group/res/domain</i> GetObjectOwner {obj typ} ListEditableObjects {}

Table 5.3 TAGS Runtime Library

5.2.4 Resources

The concept of a resource in TAGS is deliberately loose. It can be a simple timetable, an automated assessment exercise or an interactive multi-user simulation. Educational resources can be generic or subject-specific. For example a customisable timetable or marksheet could be termed generic whereas a continuously updated French current affairs digest is subject-specific to language teaching. A useful side effect of the shared environment is that developers can see what others are producing, and tutors can see types of resource that have been developed for contexts other than their own, but may nevertheless prove useful in some aspect.

5.2.4.1 Registering a New Resource Type

A Web-based form is provided to upload the files and other information necessary for the registration of a new resource type. Information about the existing registry is optionally displayed to avoid failure through a name clash. The following information is required:

name	The name of the resource type, e.g. notebook, portfolio, fileshare, ecrire, DAT, MyResource,
iconfile	Name of the file containing the icon for this type. e.g. MyResource.gif.
urls	The base URLs for the resource on each of the machines where a resource may be present. (e.g. http://dalwhinnie.dcs.st-and.ac.uk/tags/misc-bin/notebook/tags). In a single server scenario only one URL is used, but the scheme allows for replicated resources.
entrypoint	The name of the script/page/servlet that handles the resource e.g. notebookclient.cgi.

- createinstance** The name for the script that must be invoked by TAGS when we want to create a new instance of this resource. (e.g. `createnotebook.cgi`).
- deleteinstance** The script to be used for removing instances. (e.g. `deletenotebook.cgi`).
- eventdesc** The descriptions of the events sent by the resource, or a file containing the list. Names of events are unique within a Resource type.
- description** The URL of a HTML page containing a brief description of the resource and how to use it from an end-user point of view.
- replication** This refers to the coherence policy for maintaining replicated instances. The coherence models are described in Chapter 7.

The following example shows the resource definition file for the Notebook in a single server scenario:

```
name=notebook|iconfile=myspad66.gif|
urls=http://dalwhinnie.dcs.st-
and.ac.uk/tags/bin/notebook/cs3013|entrypoint=notebookclient.cgi|
urls=http://tullamore.accountancy.ac.uk/tags/bin/notebook/cs3013
createinstance=createnotebook.cgi|deleteinstance=deletenotebook.cgi|
eventdesc = "notebook_events" |
description=http://tags.ac.uk/tags/bin/notebook/tags/notebook_help.html
```

5.2.4.2 Example Resources

Once a resource type is registered instances of it can be created by tutors and assigned to groups. Resources created to date fall into four broad categories:

- management resource types for creating and maintaining a DLE – See Table 5.4a
- generic resources for managing online learning such as assignment hand-in and tracking – See Table 5.4b

- generic resource types for online learning tasks such as group communication – See Table 5.4c
- subject-specific resource types – See Table 5.4d

Some example resource types are summarised in Tables 5.4 a,b,c and d.

Resource Type	Purpose
<i>Domain Management</i>	The facility which allows service providers to create new administrative domains, and domain administrators.
<i>Resource type registration, editing and deletion</i>	Allows system administrators, service providers and developers add new resource types to the system
<i>Users, Groups and Resources Management Tools</i>	A management facility primarily for tutors. Provides the facilities to manually manage arbitrary groups of users and resources. These facilities can also be driven by importing a spreadsheet. May also be used by domain administrators.

Table 5.4a: Management Resource Types for Creating and Maintaining DLEs

Resource Type	Purpose
<i>Document Approval Tool</i>	Collaborative Tool for project and assignment management. Also supports pull and push options with respect to e-mail notification. Used by students and tutors.
<i>Reflective Marksheet</i>	How-am-I-doing tool for students, with space for self-reflection and incorporating tutors feedback for each assignment.
<i>Tasklist</i>	An assignment Management tool which supports self-selecting groups
<i>Protected Page Set</i>	Maps static HTML trees into TAGS access control space. TAGS resources are usually CGI-based, meaning that they can't be accessed without an authorisation. This resource type extends that control to static html that would otherwise be book-markable.
<i>Tutorial Attendance and Performance Register</i>	Allows students to sign up for tutorial groups and lab. sessions. They can check their own attendance and performance record. A tutor can see and modify records for his group, and a course co-ordinator can see and modify any record.
<i>Archival Tool</i>	A management tool which can create a static html archives of arbitrary groups of users and resources instances including their states. An archive can be downloaded to a local disk or kept online. All the components of an archive, including user ids and resource instances, can be deleted from the live system without losing the details of a particular class and their work.

Table 5.4b: Generic Resource Types for Managing Online Learning

Resource Type	Purpose
<i>Questions and Answers</i>	Collaborative working tool. Similar to Notebook, but the tutor has extra controls e.g. which messages to answer privately, which to anonymise and are broadcast to the entire group with comments, etc.
<i>Creditor</i>	A Web-based Collaborative Report Writing Editor, which allows a tutor to review the composition process from various perspectives to aid assessment of group-produced reports.
<i>URL</i>	A standard Web link.
<i>URL Book</i>	A page of standard Web links.
<i>Notebook</i>	Tool for collaborative working. An easy-to-use non-threaded lightweight piece of groupware which provides both the functionality of an e-mail list and a bulletin board.
<i>File share</i>	Tool for collaborative working. A shared repository for arbitrary documents. The creator can set up access permissions for different types of user in the group, which it is allocated to. For example, students may have write-only access.

Table 5.4c Generic Resource Types for Collaborative Working

Resource Type	Purpose
<i>Stock portfolio</i>	Teams of students at multiple institutions compete to manage fund portfolios. Real world input from London Stock Exchange, share prices updated every five minutes. (See Chapter
<i>Model Patients for Clinician Training</i>	Adaptations from Aberdeen MediCAL project, involves downloadable MPEG clips of real patients to illustrate symptoms.
<i>Ecrire</i>	Revision of an application originally scripted in HyperCard, for use via the Web, by groups
<i>French Current Affairs Digest</i>	Introducing real world input into language teaching, with support for group work.
<i>Genetics Case Study</i>	Groupwork for teams of medical students given real case histories (anonymised) and have to formulate relevance to patient, diagnosis and team presentation. Split campus Aberdeen/Inverness Medical Education

Table 5.4d: Subject-specific Resource types

5.2.4.3 Views into Resources

The splitting of views and capabilities distinguishes the specific requirements of a DLE from the peer-to-peer WYSIWIS approach taken in the Warp spreadsheet previously

described in Chapter 4. The view of a resource depends on the role(s) a user has. Roles are implemented as group memberships, as described in Section 5.1. There are three basic roles which are chosen from on the creation of a new user: *tutor* or *student*. Resources typically offer different capabilities to these two fundamental types of user, and this is reflected in the user documentation. This is also reflected in the user documentation where there are tutor and student versions of the user guides for each resource type. Most resources will use these basic divisions to determine the view into a resource instance. Some resources go further and offer multiple views depending on the role of the user. For example, the Document Approval Tool resource provides three views: student, tutor and course co-ordinator.

5.2.4.4 User-Centric Portal Generation

TAGS uses the resource set allocated to each user to maintain a home page, or *portal*. The purpose of a Web portal in general is to provide a single, initial point of contact for a range of services. It is a technique widely used by Web-based service providers. It can reduce the time that individuals spend searching, although it only promotes options that are commercially sponsored. Portals sometimes offer facilities for personal customization; for example, an interest in specific sporting events or the local weather forecast. However, these options are very limited. They require identity and registration, which in the context of general Web services must be user-driven as the service provider has little idea of who is using a portal. Any user may use a search engine portal for example. By and large, Web portals are public and cater for anonymous users.

User-centric portals differ from public portals in that they are built entirely around the identity of the user. Their utility in a DLE is that they can be generated for individuals and dynamically maintained using information that is already known about an individual's roles and responsibilities. For example, if a lecturer is (i) a 1st Year Advisor of Studies for the Science Faculty, (ii) teaching modules CS3013 and CS2001 and (iii) responsible for tutorial group CS1001/5, all the links to the relevant resources associated with these responsibilities can be aggregated onto a single page. Similarly, a

student who is on courses CS3020, IS1902 and EC4221 will find all their learning resources clearly presented on their home page. A users home page is updated dynamically with links to all the resources they have been allocated via their group memberships.

5.2.4.5 A resource-based approach to home page view management

To date, most TAGS users have a relatively uncluttered home page. There is a potential however for home pages to get cluttered when users such as course co-ordinators or system administrators are members of many groups. This problem can be addressed either by introducing resource variants that are specifically intended to provide overviews, or by implementing a general purpose view manager. The Document Approval Tool (DAT) and Document Tool Approval Set are examples of the former approach. The DAT is described more fully later in this Chapter. It distinguishes three roles: student, tutor and course co-ordinator.

5.2.5 Support for Service Providers

Service providers, for example IT Services or independent online learning environment providers, are supported by the TAGS framework by the following:

- registration and removal of resource types available – discussed in the Resources section in this chapter
- authentication and management of the user base – this section
- use of remote trusted authentication authorities – this section
- administrative domains – this section
- management of distribution, in the multiple server scenario – Chapter 8
- QoS monitoring – see Chapter 7
- replication – Chapter 8

5.2.5.1 Authentication and Authorisation

It is hard to overstate the importance of access control and security in online environments. Privacy is essential to ensure that a student's online work is not corrupted or plagiarised. It is also possible to try and detect evidence of plagiarism by comparing different student's submissions to the server for the same assignment, using techniques such as those described in (Finkel et al. 2002).

Authentication is essential to the maintenance of *identity*, establishing that a user is who they say they are. Once authentication is completed, authorisation needs to be performed each time a resource is accessed, to establish whether the user has the right to perform the requested operation on that resource. The TAGS scheme for authentication and authorisation is shown in Fig. 5.7. This allows for the secure remote access of distributed resources. When a user logs on to the system they are authenticated against a user base directory. This may be internal to TAGS or belong to some trusted one or more directories (which may be local or remote). Once a user is authenticated they may attempt to use a number of resources. Each access attempt is then authorised against the user's set of group memberships. The UGR tool allows the allocation and revocation of access rights to resources.

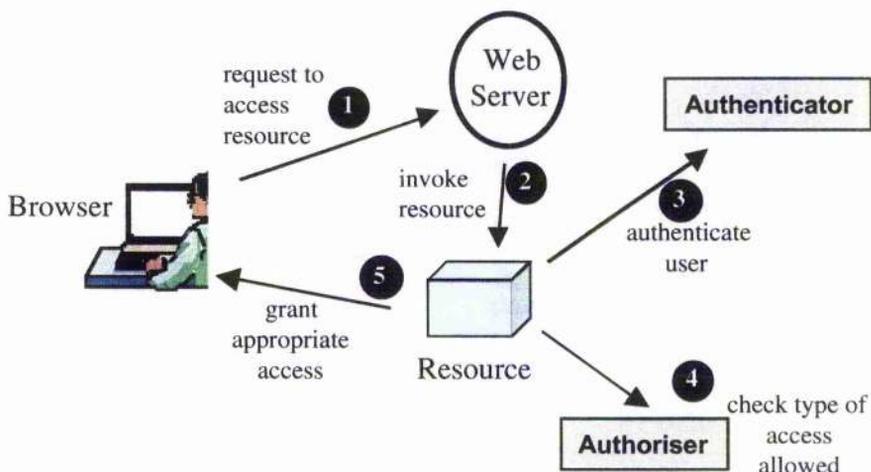


Figure 5.7 TAGS authentication and authorisation

Note that a group does not explicitly have to be an access control group, or an information dissemination group, or any other type of group – it is the responsibility of the resource to use the groupings as it sees fit.

5.2.5.2 Cookies, Session Ids and Timeouts

TAGS provides conventional user/password style authentication for its own internal database of users. The content and length of a password are enforced in such a way to make the password reasonably difficult to crack. After initial authentication, continued access to the server-based resource depends on continual automatic re-authentication. The default access control mechanism provided by Web servers such as Apache involves the use of a *.htaccess* file on the server. If this is present, and configured appropriately, it will cause the user to be challenged for id and password. The authorisation remains in place for the duration of the browser's session. This was a cause for concern as students who forgot to "logout" would be leaving access open to their private workspace, including coursework. Accordingly, in TAGS continual re-authentication is carried out by the use of tokens, encapsulated by cookies, which are transmitted between the browser and the Web server on each interaction. The tokens are time-stamped. If a user is inactive for a specific period of time (set by the tutor), the next automatic re-authentication attempt will fail. This reduces the risk of a student leaving their work exposed via a public access terminal browser session. There is no attempt to control access by IP address, and such a scheme would not be any use as many accesses are from ISPs.

5.2.5.3 Interoperability with Remote Authentication Authorities

If a new user is added to a DLE it is highly likely that they will already have one or more user ids and passwords issued by their host Institution or Department. So, although a user must explicitly be added to TAGS it is highly desirable to let them use an existing id and password. The advantages are i) the user does not require another id and password to remember— this is particularly useful in cases where students are issued with an id and password on matriculation ii) if their institutional id is terminated then they will not be able to access TAGS without making special arrangement; iii) problems such as forgetting a password are handled by the remote authority and iv) there is no need to store sensitive data such as passwords.

The Lightweight Directory Access Protocol (LDAP) (Yeong et al. 1995) has gained acceptance as a means of deploying directory-based services in an Internet environment. The adjective "Lightweight" is intended to reassure the potential user that this is not a heavyweight cumbersome standard like X.500, the ISO standard for directory services. LDAP is increasingly associated with providing a standard network interface to authentication directories, thereby abstracting over services such as Sun's NIS, Novell's NDS and Microsoft's Domain system. The TAGS authentication mechanism allows for the use of LDAP. This currently provides a choice of authentication domains, including the University's Information Technology Service and the School of Computer Science (ITS, DCS). In other words, TAGS will allow bona fide students or staff to use their normal Institutional identity credentials for authentication.

A user must still be registered in the TAGS internal user database, and their roles and privileges are dependent on what groups they have been allocated to, and what resources have been allocated to these groups.

5.2.5.4 Administrative Domains of Protection

The role of *system administrator* is supported in TAGS. It is somewhat akin to that of *root* in Unix systems. Course co-ordinators were originally created belonging to the *sys-admin* group, thereby enabling them to create tutors as well as students. As use of the TAGS system grew it became clear that course co-ordinators were not happy with having to shoulder such system wide responsibility. On the one hand they did want to have full powers with regard to their own creations, but on the other they did not want the capability to accidentally damage users, groups or resources created elsewhere. The problem here is the tension between sharing and protection - how could the collaborative nature of the environment be maintained while introducing strong protection barriers?

The TAGS architecture resolved this by constraining the rights of a system administrator within the boundaries of an Administrative Domain of Protection. The possibility of a user being associated with more than one domain, must be taken into account. A single

system-wide identifier and password should allow access to all of the domains that a user has the right to access.

The problem of protection is one that has been explored in the context of operating system design (Silbershatz & Galvin 1994).

- A computer system can be thought of as a collection of processes and objects. The operations that are possible may depend upon the object.
- A process should be allowed to access only those resources it has been authorised to access. Furthermore, at any time it should be able to access only those resources that it requires in order to complete its task. This “need-to-know” principle limits the damage that can be done by a faulty process.

In the TAGS system a user plays a role analogous to that of a process in an operating system and Users, Groups and Resources that of operating system objects.

A domain is an area of protection, which maps to an administrative area of control. A user, resource or group may belong to one or many domains of protection. Normally a user will only be active in one domain at a time. If they wish to manage objects in another domain to which they have management privileges, they must explicitly change the domain in which they are active. This helps support a weak version of the need-to-know principle. The user only has access to those objects they need for the tasks they are undertaking at a particular point in time.

5.2.5.5 Groups and Domains

Group membership is used to structure collaborative tasks, so in order to retain this benefit TAGS treats group membership as a tighter binding than domain membership. Consequently, assigning members of different domains to the same group will facilitate inter-domain communication. The group has to belong to multiple domains but the users may be in separate domains. For example, a student may be in the Physics and Comp. Sci domains. Groups may also be used as a mechanism for allowing resources in one domain to be made available to users in other domains. In this way domains provide the administrative protection that is required without unnecessarily constraining the freedom

of communication that collaborative group work requires. Entities which did not have an obvious domain to be moved into were left in the "open" domain. The domain manipulation routines in the TAGS library are listed in Table 5.5.

<pre>AddResourceToDomain {res domain} AddUserToDomain {usr domain} RemoveResourceFromDomain {res domain} RemoveUserFromDomain {user domain} CreateDomain {domain} DeleteDomain {domain} SetGroupDomain {grp domain} CanRemoveFromDomain {obj typ domain user} Does the same as CanDeleteObj, but inside a Domain GetAllDomains {} Returns: list of all domains in the system GetUserGroupsInDomain {user domain} GetUserDomains {user} GetResourceDomains {instance} GetGroupDomain {group} ExistsDomain {domain} GetDomainUsers {domain} GetDomainGroups {domain} GetDomainResources {domain}</pre>
--

Table 5.5: Domain Management

5.2.5.6 How Domains are Used

Domains are used in a variety of ways, reflecting the range of scenarios that are encountered in higher education. Finesse users across several HE institutions all share the same "Finance" domain, as a deliberate choice. In other cases a domain is mapped on to a single module, such as "BED3" or "CS2001" in the active user display in Figure 5.8. Further variations may include a whole degree programme involving several modules being mapped into a single domain – VIT for example.

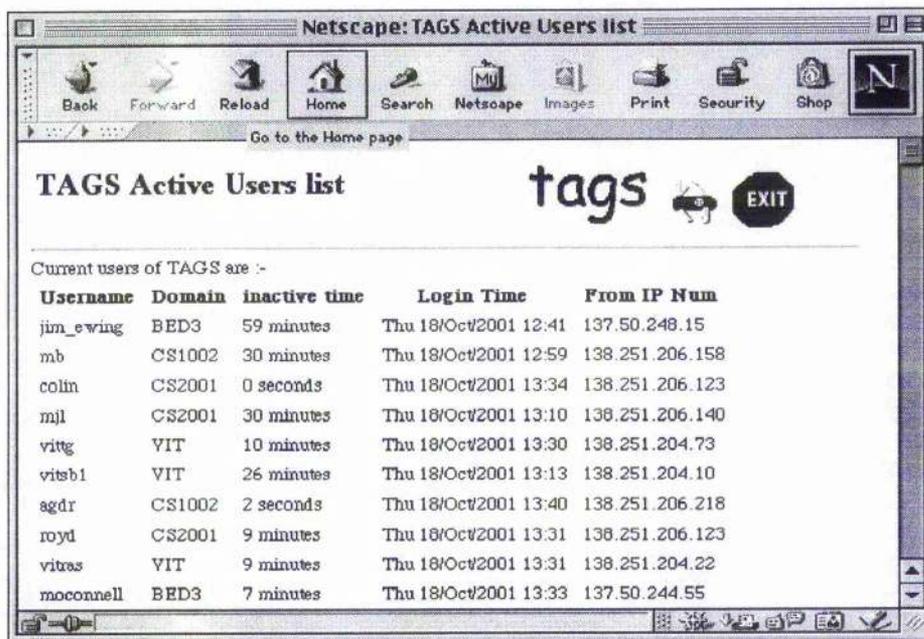


Figure 5.8: Domains are used in a range of scenarios

5.3 Case Study: Group-Based Management of Distributed IT Work Placements

Finesse can be seen as a case study of a DLE featuring a subject-specific resource, the PMF. In this section a complementary case study is presented, featuring a generic resource for assignment tracking using group-based project management. When combined with the description of Finesse this gives a flavour of the range of resource types educators require for pedagogical reasons. The Document Approval Tool (DAT), described here, also shows the growing use of web-based groupware for managed learning environments (MLEs). MLEs are a subset of DLEs that are specifically concerned with the management and business processes involved in providing education.

The School of Computer Science at St Andrews runs a postgraduate IT course. The course includes a twelve-week work placement project. The grouping associated with each placement consists of an external project provider, one or more students, a local supervisor who is a member of the academic staff and a project assistant, typically a tutor. Placements are mostly spread over Fife and Tayside, but some have been located

further afield – Israel and South Africa for example – and even in Edinburgh and Glasgow. So, there are at least four people associated with each project: the student(s), the local supervisor, the project assistant and the remote supervisor.

The traditional project management process involved a substantial amount of paperwork. This included a project specification, a final report, software documentation, and weekly reports. All of these had to be signed by the student, the local and remote supervisors and the project assistant, as evidence that the project was progressing satisfactorily. This substantial paper trail was used for post-course auditing and was presented to the external examiner, but was not easy to use for individual student feedback. Supervisors tended to batch reports for signing for example, so they failed to provide timely warnings of problems developing within a particular project. A further consideration was the frequent but hidden use of e-mail for discussing project work.

The aim of using TAGS to produce a DLE for distributed IT project management was to:

- eliminate the paper trail, whilst maintaining accountability
- to improve monitoring so that problems could be identified in a timely manner
- to provide a means of online communication for project groups, which provided a coherent record of the project dynamics
- to pilot the use of electronic signatures

Three TAGS resources were used, two of which already existed, and one which was developed. The Users, Groups and Resources Management Tool was used to generate user accounts, groups, and allocate resources to everyone associated with the course. This included eighty students, twelve academic staff, six tutorial assistants and thirty placement providers. Fewer than 8% of these users could not use TAGS directly due to lack of routine Internet access. The Notebook Tool was used for intra-group communications, and a new resource, the Document Approval Tool, was developed to specifically address the monitoring needs of the course.

5.3.1 The Document Approval Tool (DAT)

The DAT was developed to speed up and simplify the handing in of all documents associated with a work placement – the project specification, the weekly reports, the final report, and auxiliary documents. The DAT allows these documents to be uploaded to a protected space on a Web server where all members of a group can access the document, read the document and leave comments.

A DAT consists of zero or more *slots*. Each slot has a name and can accommodate a single document. If more than one document is uploaded to it a version number will be appended. Only tutors can delete documents from slots. There are two dates associated with a hand-in slot: the *due date* and the *acknowledge-by date*. The DAT can automatically notify supervisors and remind students when a deadline has been missed. It can aid overall course management by alerting the course co-ordinator when an acknowledgement deadline has been missed, for any reason. Thus the system reminds tutors, mentors and administrators when dates of respective interest have been missed.



Figure 5.9: The Slot Definition Editor

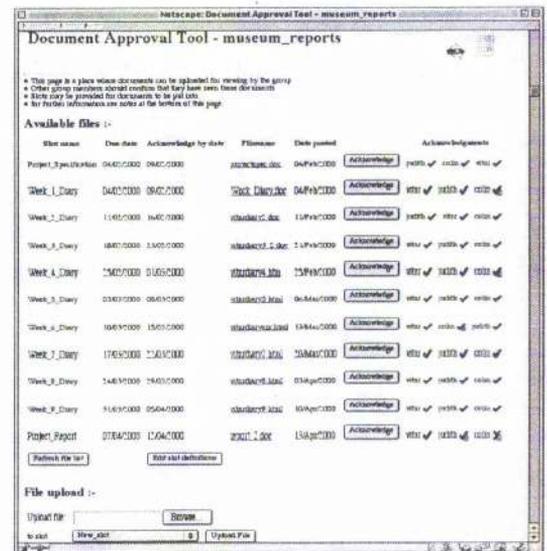


Figure 5.10: The main DAT window

On initial creation, a DAT is empty with no slots defined. In this state it can be used to upload files and allow other users to comment on them. To access the monitoring and reporting features, slots must be defined. The page for defining slots is shown in Fig. 5.9. The example shown in Fig.5.10 already has a slot defined at the top. The lower

buttons are for creating new slots. Slots can also be copied as templates from any other DAT instance in the TAGS system. The users associated with a slot are the members forming the group to which the DAT has been allocated.

There are three classes of operation defined by the tick boxes for each member of the group that has been allocated a DAT. In order to understand the last of these, it is necessary to know that a document can either be accepted or rejected when it is acknowledged. In the case of a rejection, the message entered is also e-mailed to the user who uploaded the file.

- E-mail on upload & must acknowledge - Users will receive an e-mail any time a document is put into that slot requesting that they acknowledge it by the acknowledge-by date. E-mail will also be generated every day after this date if they have not acknowledged the document.
- Alert if not uploaded by due-date - Users will receive e-mail every day after the due-date if there is not a document in the slot.
- Alert if not accepted by acknowledge-by date - Users will receive e-mail every day after the acknowledge-by date if the most recent acknowledgement from any user in the group is a rejection.

Users need not be selected for all (or indeed any) of these alert conditions. The ticks associated with a particular slot can be reproduced across all slots in a given DAT by a single click.

Once slots are defined, users can start to upload documents. These are presented in a table along with some information about the file; green ticks indicate acceptance, red crosses indicate rejection. Either of these may be annotated in which case clicking on the icon will bring up the comment in an alert box. The view of DAT for a given group is shown in Figure 5.10. This view, listing the status of each slot, is visible to both tutors and students. Students can upload documents to particular slot, and also to the same workspace, to an unnamed slot.

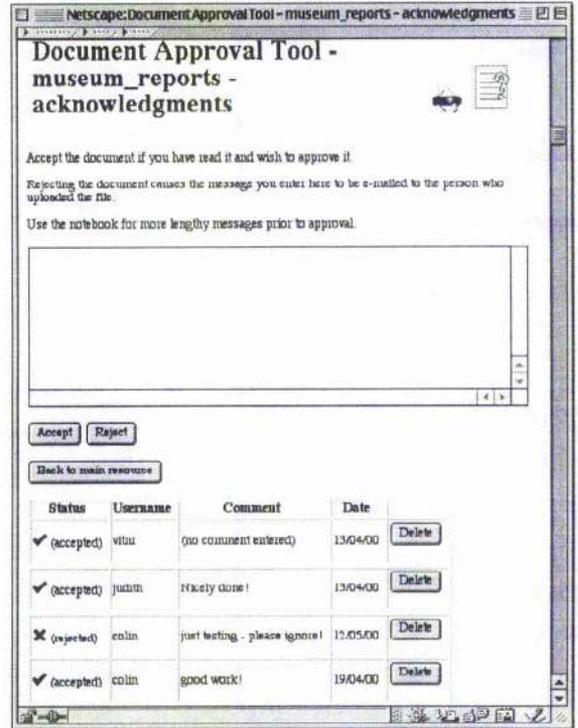


Figure 5.11: The Acknowledgement and Comment Window

Only tutors can delete documents. When a tutor is notified of a submission they are e-mailed the URL which directly links to the DAT instance. They are challenged for authentication if not already logged in to TAGS. A screen shot of an acknowledgement window is show in Figure 5.11. The most recent comment entered in a DAT slot is also visible as a small sheet-of-paper icon attached to a tick (accepted) or cross (reject). If clicked the comment is displayed as a pop-up alert box.

5.3.2 Evaluation of the DAT

Informal formative evaluation took place during the placement period with feedback from tutors and students to the software team via the developer group Notebook. The DAT tool proved popular with the tutors, supervisors and students involved with the course. A more formal evaluation based on a short questionnaire of eighty students and all supervisors was also undertaken and reported in (Allison et al. 2000a).

- Informally, the DAT appeared to have met its targets and also provided extra benefits:

- A supervisor who had to avoid small groups for medical reasons was keen and able to continue supervision solely through the use of the Notebook and DAT resources.
- A complete record of all work is now online enabling good examples of specifications and reports to be selected for showing to new students
- An archiving facility was developed which gathers all CGI output from all DAT instances associated with the course and creates a static HTML, self-contained Web site suitable for copying and moving.
- The external examiner expressed satisfaction in being given a URL and a password instead of several kilograms of paper.
- The use of the TAGS authentication mechanism to replace written signatures in this context has been accepted. Although the use of an asymmetric public key system was considered, it was felt that it offered no advantage in this situation, and would probably not be achievable without a major installation and support exercise for computers not based at the University.

Other comments from staff and students highlighted the need for timely responses to student reports. The use of the DAT, with its e-mail reminders, has shown local supervisors the work required in supervising project placements.

5.3.3 Evolution of the DAT

After the success of the use of the DAT in the course described above, it was selected for use in other modules, and has become the single most heavily used resource running under TAGS. In order to support the requirements of modules in general the following features have been added:

- A marks facility - this can be made visible to or concealed from the student
- A facility to explicitly e-mail a comment to a student has been added

- A view by slot facility which allows a tutor to see all their hand-ins and marks for a particular assignment. This provides simple averages and a facility to download the marks to a spreadsheet for more general manipulation.
- A view by tutor facility - this provides a tutor with an overview of all their students and their progress in all assignments.
- A view by module facility. This provides an overview of the entire class and every assignment, and is implemented as separate resource, a DAT set (See Fig. 5.12). Although primarily intended for course co-ordinators, this has also proved popular with tutors wishing to see how well a course is running at a glance. Student names have been hidden in the screenshot.
- A download facility. As spreadsheets such as Excel are well suited to manipulation and analyses and presentation of marks it was deemed to be more useful to download a copy of the course or tutor view in a spreadsheet format rather than trying to duplicate the functionality of such desktop tools.
- An auto-archive facility. This saves an administrator time in running the TAGS archiver explicitly. A single click will create a static html archive of all material and marks associated with a particular DAT set.

Netscape: Document Approval Set page

Back Forward Reload Home Search Netscape Images Print Security Shop Stop

cs0080 Group DAT	✓ 20	✓ 17	✓ 20	✓ 19	95.00	[Redacted]
cs0081 Group DAT	✓ 18		✓ 12		75.00	[Redacted]
cs0082 Group DAT	✓ 20	✓ 12	✓ 14	✗	76.67	[Redacted]
cs0083 Group DAT	✓ 20	✓ 20	✓ 8		80.00	[Redacted]
cs0084 Group DAT	✓ 8				40.00	[Redacted]
cs0085 Group DAT	✓ 20	✓ 19	✓ 20	✓ 20	98.75	[Redacted]
cs0087 Group DAT	✓ 19	✓ 18	✓ 20	✓ 20	! 96.25	[Redacted]
cs0088 Group DAT	✓ 18	✓ 18	✓ 17	✗	88.33	[Redacted]
cs0089 Group DAT	✓ 14				70.00	[Redacted]
cs0092 Group DAT	✓ 20	✓ 16	✓ 12	✓ 10	! 72.50	[Redacted]
cs0099 Group DAT	✓ 20	✓ 14	✓ 20	✓ 18	! 90.00	[Redacted]
cs0100 Group DAT						[Redacted]
cs0101 Group DAT	✓ 19	✓ 13	✓ 19	✓ 16	83.75	[Redacted]
cs0102 Group DAT	✓ 7	✓ 0			17.50	[Redacted]
eca Group DAT	✓ 20	✓ 19	✓ 17	✗	93.33	[Redacted]
jmb Group DAT	✓ 20	✓ 17	✓ 19	✗	93.33	[Redacted]
rd1 Group DAT	✓ 20				100.00	[Redacted]
ruc Group DAT	✓ 20	✓ 6	✓ 10		60.00	[Redacted]
rst Group DAT	✓ 18	✓ 18	✓ 20	✓ 13	86.25	[Redacted]
Average / Max	17.97 / 20	14.27 / 20	16.53 / 20	17.70 / 20	/ 20	

File present and all acknowledgments are acceptances
 File present but one or more required acknowledgments are missing or are rejections
 File present but acknowledge date has not passed

Figure 5.12: Course Co-ordinator's View

5.4 Summary

The Web has become the de facto platform for groupware. It provides an ease of deployment and very low cost which is particularly attractive to the education sector. There is a strong requirement for a framework to research, develop and deploy DLEs on the Web. This comes from both the pedagogical needs of subject-specialists and the needs of those involved in the management and delivery of courses. There are many common needs which a framework can provide for a wide range of DLEs: security, role-based access, user-centric portals, monitoring, abstraction over technicalities, sharing and group working. At the same time a framework must be flexible to allow for exploration and customisation. In addition to educators and learners a framework must also support resource developers and service providers. The primitives provided in TAGS – users, groups, resources, events and domains – have been shown to meet the requirements from all those categories of users. Table 5.6 below summarises usage of the TAGS production system by Institution, Discipline and Resource type in academic year 2000/2001. The range of the users and resources hosted by TAGS validate its success as a Framework.

Resource Type	Students	Disciplines	Institution
Portfolio Management Facility	250+	Accounting, Finance and Management (AFM)	Aberdeen, Dundee, Glasgow Caledonian, Glasgow, Strathclyde
écrire	200+	Languages	St Andrews, Dundee
Register	160	Economics, Computer Science	St Andrews
DAT	400+	Computer Science	St Andrews, Dundee
MarkSheet	60	Physics	St Andrews
Notebook:	200+	Computer Science, Physics, AFM, Education Studies, <i>Developers</i>	St Andrews, Dundee
Q&A	110	Computer Science, Physics	St Andrews
TaskList	60+	Education Studies	Northern College
PageSet	100+	Computer Science, Physics	St Andrews
FileShare	20	Computer Science, <i>Developers</i>	St Andrews, Durham, Dundee
Archiver	250+	Computer Science, Physics, Education Studies	St Andrews, Northern College
Users Groups and Resources Management Tool	2000+	<i>All listed above.</i>	<i>All listed above.</i>

Table 5.6: Summary of Usage in Academic Year 2000/2001.

The TAGS framework has also provided a focus and development environment for QoS-monitoring, distribution, replication and video conferencing. These features are described in Chapters 6 and 7.

6 Understanding the Quality of Service Requirements for Interactive Responsiveness

Distributed Learning Environments, such as those deployed under the TAGS framework, consist almost entirely of server-based interactive learning resources. In an holistic sense of QoS we can say that both *coherence* and *responsiveness* are requirements for such resources. This chapter however concentrates on understanding the responsiveness issue and describes work carried out to identify the sources of delay in distributed learning environments.

The work was carried out as two investigations, the second (Allison et al. 2001) building on experience obtained from the first (Allison et al. 1998). The approach differs from most QoS analyses in two respects: (i) it takes the end user's experience into account; and (ii) it provides a basis for feeding back the results of QoS evaluation into the DLE. If a DLE aspires to be QoS-aware these two features are essential. They are indicative of a particular concern with the *interactive responsiveness* of a DLE, monitoring it as a key QoS parameter, and providing the possibility of taking actions to improve it when necessary, where possible. Such analyses can also feed into longer term development and adaptation strategies. This approach contrasts strongly with the type of QoS measurements made by telecommunications companies which do not take the end-user's experience into account and are primarily concerned with providing evidence in disputes over Service Level Agreements.

Figure 6.1 shows the typical HTTP / TCP / IP protocol stack that is employed by the bulk of interactive web transactions between browser and server. Other protocols such as DNS could also contribute to delay, but in practice browsers cache DNS mappings for the duration of an invocation, so the impact is negligible where the same sites are repeatedly contacted.

Distributed Learning Environment		
<i>Multiple Interactive Server-Based Shared Learning Resources</i>		
Hyper Text Transfer Protocol (HTTP)	other application level protocols	Domain Name Service (DNS)
Transmission Control Protocol (TCP) <i>two-party reliable data exchange, no time guarantees</i>		UDP
Internet Protocol (IP) <i>unreliable, best effort, no time guarantees</i>		

Figure 6.1: Network Protocols Involved in Typical Web Interactions

The first QoS investigation, described in section 6.1, was prompted by problems encountered in the Finesse Portfolio Management Facility, and identified basic problems subsequently found in many web-based services. However, that investigation did not provide a detailed analysis of network behaviour or suggest a means of automatically detecting changes in delay that were likely to affect the user. Both of these issues were addressed in the second investigation which uses a revised model that places a stronger focus on the information that can be gleaned from the transport level (TCP) packet headers.

The revised model, described in section 6.2, digs beneath the application-level HTTP and distinguishes transport protocol delay from network delay and attempts to identify the interplay between network, client and server components connected by TCP. As passive monitoring can be carried out on network paths the ability to infer an applications performance from network behaviour provides a basis for developing QoS-aware feedback systems for applications and servers.

Finally, section 6.3 applies the revised model to the analysis of TAGS traffic and reports the results.

6.1 Interactive Responsiveness in Finesse

The Finesse DLE was originally designed to be used by individuals and small groups, either remotely at the same time, remotely at different times or as a physically co-located

team sitting round one machine making decisions. It was not envisaged as being used by more than a handful of students concurrently. However, the need to support a much higher degree of concurrent usage emerged as a requirement from teaching staff, who wished to hold introductory induction sessions with whole classes at a specific time¹. In terms of system load this meant that instead of having to cope with a worst case of five or six concurrent users, the system could be used intensively by up to eighty students at the same time².

A major problem was encountered in the first Finesse induction sessions, namely that response time was too poor to hold the student's attention. Staff involved with running the sessions quickly assumed this to be a network problem. In reality the total delay experienced by an end-user may be due to a wide range of causes including network delay, browser design, server overload, poor HTML page design, or even bad HTML produced by commercial authoring tools. However, because analysing this delay and breaking it down into its constituent parts is not trivial, people often make decisions about how to improve a web service based on little more than guess work. Although there has been much work focused on i) improving web server performance (Arlitt & Williamson 1997, Cardellini et al. 1999, Schroeder et al. 2000); ii) web content caching (Rodriguez et al. 2001); and iii) web communication protocol efficiency (Mogul 1995, Spero 1998), there has been considerably less regard for incorporating the users perspective in detailed technical analyses.

The QoS analysis carried out to investigate these early problems with Finesse performance made three contributions: it presented a simple structured timing model to guide analysis of total delay; it described a number of lightweight methods for measuring

¹ For induction and training purposes Finesse was enhanced so that the progression of stock market data changes could be drawn from a historical period and accelerated for the duration of these introductory sessions. For example, share prices could be set to change every two minutes based on monthly progressions from some point in time set several years ago.

² Numbers could be higher if larger Labs were used or if two induction sessions were scheduled to overlap from separate large Labs.

the constituent parts of the total delay; and it listed a set of recommendations for substantially reducing delay in specific situations.

6.1.1 Basic Web Operations

Figure 6.2 summarises typical basic web operations possible in 1998. Figure 6.2a shows a basic web interaction involving a client and a server. The server response may be a simple file retrieval and transmission, or the invocation of a program via the Common Gateway Interface (CGI), which generates output that is returned to the client, 6.2b and 6.2c show the potential for concurrency in client and server. Figure 6.2d shows a typical scenario where a browser aggressively requests many remote file-based objects at the same time to try and assemble a page more quickly for display.

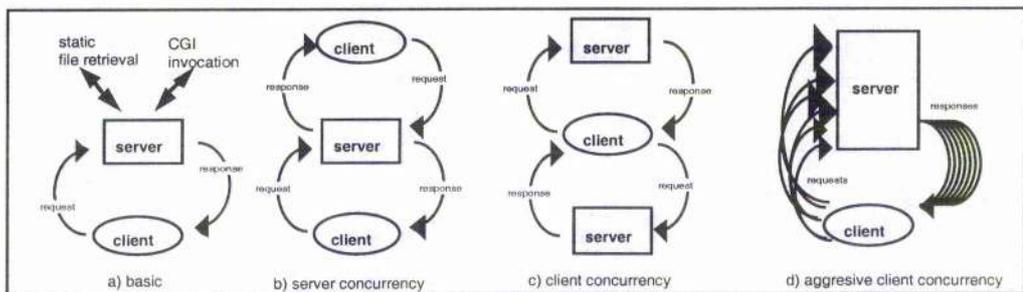


Figure 6.2: Web Client-Server Interactions

HTTP 1.0 (Berners-Lee et al. 1996) has four methods of communication between the client browser and the web server. These are GET, POST, HEAD and PUT. The PUT method is used to upload HTML forms from the client to the server. The HEAD method is used to request meta-information (the header) about the requested file object e.g. in order to create an entry in a search engines database. The POST method is used in HTML forms to send structured information entered via the web browser back to the web server. The GET method is the most frequently used method, which requests an entire document from the server. It is the GET method we are most interested in as it is typically a user triggered event which features the user waiting for data to be returned and rendered on their display.

There are broadly three types of data returned: static, dynamic and active [Comer]. Static refers to HTML files, dynamic refers to server-generated HTML, and active refers to executable code which is downloaded to the browser, such as Javascript and Java applets.

6.1.2 Proxies and caching

There are two widespread optimisation strategies used in the web architecture to minimise delay. *Client-side caching* works by keeping copies of recently accessed files on local storage. An HTTP request-response period can be shortened: a check is made on the header of the remote file (using HEAD), a comparison is made with the header of the local copy in cache, and, if there has been no change, the local copy is used. *Proxy servers* can be setup at any number of levels e.g. local, campus or national. The idea is that a browser will go to the proxy to see if the file is cached (i.e. it has been requested by at least one other user). If the proxy does not have the file it will attempt to fetch it, cache it and then return it to the client. In situations where large numbers of users who share a proxy server also request the same static HTML pages there can be good pay off. Unfortunately, neither of these optimisations improve the response times of server-generative applications, which tend to be highly interactive and non-cacheable. Accordingly, neither are addressed in the analysis of Finesse delay. HTTP 1.1(Fielding et al. 1997) offers an optional feature for caching database query results but there is little evidence of it having been deployed, and if so, what its effect on the server response time would be.

6.1.3 A Structured Timing Model

The model covers the entire period from the user-initiated selection of a URL, to the resolution of all its associated links within the scope of an HTML page. This period is referred to as the closure over a URL, or *CURL*. A *CURL* may terminate correctly or incorrectly. When a *CURL* terminates correctly all images, files, applets and other components that are associated with a URL are located or generated, retrieved, displayed and/or activated. A *CURL* may terminate incorrectly if there is a failure in any of its

constituent parts. There are numerous possible sources of failure and some will not result in any diagnostic message being delivered to the user. In order to simplify the description of the model we will only consider CURLs which terminate correctly. A CURL involving an HTTP URL is initially broken down into one or more instances of a top-level three phase pattern:

- *client-side HTML parse and request generation*
- *server-side request processing*
- *client-side rendering time consisting of parse time and display update*

This pattern is interleaved with network transit times (including source and destination protocol processing) represented by N_{CS} and N_{SC} , where N_{CS} denotes the time taken during the communication phase from client to server, and N_{SC} the delay in the other direction. Phase C may result in further requests and there is usually an overlap between request generation and other HTML parsing.

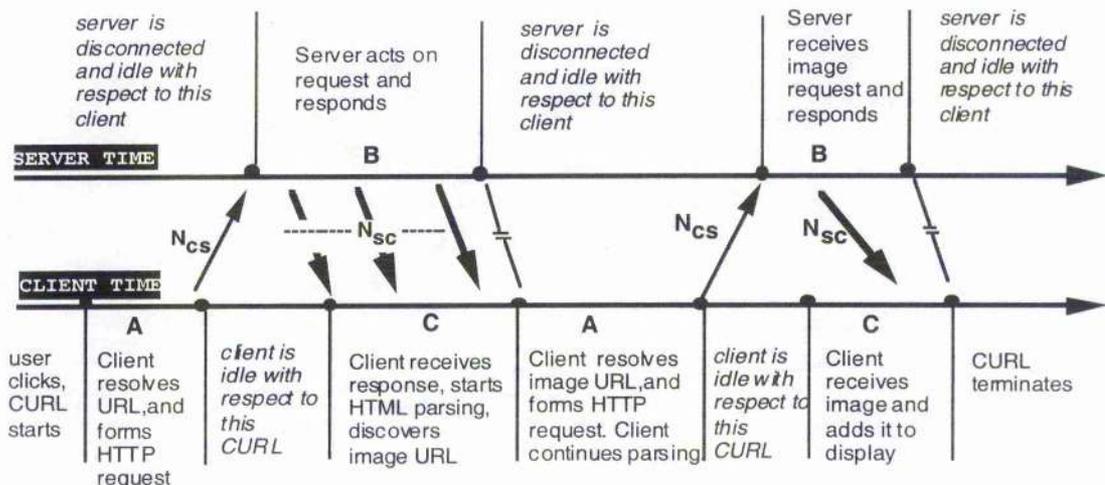


Figure 6.3: Three Phase Pattern and Overlap

Figure 6.3 shows an example CURL consisting of a URL which results in the retrieval of an HTML page which in turn contains a single image reference. The top-level phases can be broken down further. The following list illustrates basic web interaction:

Phase A

- Make a DNS lookup if raw IP number not used in URL (the typical case).

- Client opens a TCP connection to the server.
- Client sends an HTTP GET request which includes client-side environment variables.

Phase B

- Server listener process receives the GET request and spawns or calls a process to deal with it. Concurrent server implementation varies widely.
- Server parses the request.
- Server does a DNS lookup on client. Writes log entry.
- Searches for .htaccess files in directory containing requested file and parental directories.
- If client is not allowed returns fail message. Write to error log.
- Determines requested type and invokes appropriate handler e.g. CGI or Static HTML.
- In the case of CGI, handler starts CGI program, passes it CGI variables.
- In the case of Static HTML returns file.
- Return data is queued for network transmission.
- Server closes TCP connection.

Overlap between Phases B and C

- Client starts reading from the receive buffer and parsing the HTML
- The server flushes the buffer to the client and closes the connection
- Server writes to access log.

Phase C only

- Client finishes reading from receive buffer and parses HTML into part of viewable web page.
- If Client comes across an image URL it enters Phase A with the URL as a parameter.

6.1.4 Timing Issues

To fully analyse a CURL in detail would require access to the source code for the servers, browsers and their enclosing operating systems. In practice reasonable estimates may be achieved by using Javascript on the browser and CGI programs on the server. This allows for the insertion of timing probes throughout the resolution of a CURL. An important assumption is that the Network Time Protocol (NTP) is running on the server and all the clients. NTP clients are widely available for most platforms and were operational on all the machines used in the tests. This synchronised the client and server to within 10 milliseconds. Figure 6.4a is an example outline of a typical CGI script. Figure 6.4b is the modified version which displays timing information.

```
#!/usr/local/bin/tclsh
# a generic CGI script written in TCL

package requires cgi

cgi_eval {
  cgi_input
  # variables setting up the environment

  cgi_head {
  # specifying the HTTP header for the generated HTML document
  }

  cgi_body {
  #main block of CGI code creating main body of HTML document
  }
}
```

Figure 6.4a: CGI script outline - no timing information recorded or displayed

```

#!/usr/local/bin/tclsh
# a modified CGI script for timing

# get the start time
set start [clock seconds]

package requires cgi

cgi_eval {
  cgi_input

  # If the variable sendtime exists import it: this is the time that the
  # user made the request
  set sendtime ""
  catch { cgi_import sendtime }

  cgi_head {
    # javascript function to display load completion time
    cgi_javascript{ cgi_puts
      ( function whattimeisit() { return (new
Date()).toLocaleString() )
    }
  }

# The HTML body is modified to display clock time on completion of document parsing
cgi_body onLoad = alert(whattimeisit()) {
  # if sendtime does not exist allow user to reload document with sendtime.
  if {$sendtime == ""} { cgi_form timing
onSubmit=this.sendtime.value=whattimeisit(){
  cgi_export sendtime
  cgi_submit_button
  }
  } else {
  # run normal CGI code main block of CGI code creating main body of HTML
document
  # now display cgi parsing times
  puts "<P>Client request time:$sendtime\n"
  puts "<P>Start CGI time: [clock format $start]\n"
  puts "<P>End CGI time: [clock format [clock seconds]]\n"
  }
}
}

```

Figure 6.4b: script modified to provide timing

6.1.1.1 Minimum Client Rendering Time

In order to estimate the clients rendering time (Phase C) network overheads are eliminated by running the browser on the server and using the IP local host address for loopback communication. The end of client rendering time is obtained by using JavaScript's *onLoad* method. The start of client parse time is harder to record because there is no Javascript event emitted when the client starts to receive the server's response. A client may start parsing before a CGI program has finished running on the server. However, the minimum rendering time can be obtained by measuring the time elapsed between the server completing CGI execution and the browser completing the rendering of the resultant HTML page.

6.1.1.2 Network Overhead Time

An approximation of the time it takes to transport the data across the network can be obtained by eliminating the clients parse time and using the same CGI script as in the

previous timing method. This involves running the client browser on a platform equivalent to the server. Assuming that the parse time is approximately the same as that already obtained, the remaining time is the network transfer. Within the CURL this may be expressed as $(N_{sc} + \text{Phase C time}) - \text{Phase C time}$. A means of estimating Phase C time is described later.

6.1.1.3 Fastest CGI Run Time

As an aid to comparison it is useful to obtain the minimal run time of the CGI program. To do this the network, client, and server overheads must be eliminated. This may be achieved by running the CGI program in a command shell. This gives an absolute minimum possible Phase B time.

6.1.1.4 Actual Run Time

This is the actual Phase B time within the CURL i.e. the actual time taken for the CGI script used in 6.4 to run, plus the overheads involved when the output is being passed back through the web server to the browser running on the client.

6.1.1.5 Client Request Time

This is the period between the client submitting the request to the server and the server starting to send data back to the client's browser. It is measured by appending a time-stamp to the client request and comparing it to a time stamp at the start of the CGI script. This provides timings of Phase A + N_{cs} .

6.1.5 Measurements

The largest Finesse CGI program generates an HTML table of current stock market prices (See Fig.6.5). It has seven columns and over a thousand rows, and is about 180 KByte. This is used as a test case not only because it stresses browser and server-side processing, but also because it is the single most frequently requested page from the server in student induction sessions. Note that the relatively small raw data size of the table belies its ability to cause delay.

Share Price List

Date: Thu Mar 19 13:48:26 GMT 1998

Start From: 13:48:26

Prices as of: 19 3 1998

Sector: WATER

Type: "All types"

Sector	Price	Dividend	Dividends / Share	Earnings / Share	Currency	Type	Price (pence/share)
ANGLIAN WATER	10.4	10.25*	158.24	80.55	£	FTSE250	885.00
HYDER WATER	7.7	236/97	155.69	122.00	£	FTSE250	969.00
SEVERN TRENT	11.1	165/97	189.54	85.61	£	FTSE100	982.00
SOUTH WEST WATER	11.4	96/97	187.26	78.63	£	FTSE250	924.00
THAMES WATER	11.7	50/98	179.90	76.79	£	FTSE100	935.00

```

<br><table cellpadding=3 cellspacing=0 border=0
width=100%>
<tr bgcolor=#EEEEEE><td>Name
</td><td align=right>Price (p)
</td><td align=right>PE (p)
</td><td align=right>EPS (p)
</td></tr><tr bgcolor=#FFFFFF><td class=price>10
GROUP
</td><td align=right class=price>0.50
</td><td align=right class=price>#n/a
</td><td align=right class=price>#n/a
</td></tr><tr bgcolor=#F4FF4><td class=price>3I
GROUP
</td><td align=right class=price>1284.00
</td><td align=right class=price>62.3
</td><td align=right class=price>19.30
</td></tr><tr bgcolor=#FFFFFF><td class=price>ABBEY
NATIONAL
</td><td align=right class=price>1301.00
</td><td align=right class=price>13.7
</td><td align=right class=price>93.40
</td></tr><tr bgcolor=#F4FF4><td
class=price>ABERDEEN ASSET MAN.
</td><td align=right class=price>586.00
</td><td align=right class=price>27.5
</td><td align=right class=price>19.74
</td></tr><tr bgcolor=#FFFFFF><td
class=price>ABERDEEN
</td><td align=right class=price>1.12

```

Figure 6.5: The Stock Prices Page, as displayed, and a fragment of the source html

The timings listed in Tables 6.1a and 6.1b were obtained using the techniques described in the previous section. Table 6.1a consists of timings made with one client accessing the server. Table 6.1b shows the results for a concurrent load of four clients. The server timings in Table 6.1a were generated by a client being run on the server, giving a basis for estimating network overhead time when they are subtracted from the same constituent times measured from an identical model of computer as the server across the network.

The following computers were used as clients:

- Mac: 200 Mhz Mac 8600, 64Mb, MacOS 7.6.1
- Sun: Ultra 140, 64Mb, Solaris 2.6
- PII: 266Mhz Pentium II, 64Mb, Windows 95
- 486: Intel 66Mhz 486, 20Mb, Windows 95.

The server was a Sun Ultra 140, configured identically to the client. Two different types of browser were tested on the Intel clients.

	Mac	Sun	PII Netscape	PII Explorer	486 Netscap	486 Explore	Server (Sun)
total time (CURL)	60	45	38	24	285	103	43
submit time (Phases A+B - overlap)	0	0	0	0	0	0	0
CGI runtime overlap - Phase C	26	22	18	19	56	28	21
minimum network & client rendering time (Phase C)	34	23	20	5	229	75	22

Table 6.1a: CURL constituent timings in seconds for single client access

	Mac	Sun	PII Netscape	486 Netscape
total time (CURL)	130	81	85	283
submit time (Phases A+B - overlap)	0	0	0	0
CGI runtime overlap - Phase C	70	58	62	79
minimum network & client rendering time (Phase C)	60	23	23	204

Table 6.1b: CURL constituent timings in seconds for four client concurrent access

6.1.6 Analysis and Solutions

This simple but replicable set of tests provides some insight into the sources of delay as perceived by the user. Interestingly, in direct contrast to the "common sense" diagnosis, network overheads are relatively insignificant. Client rendering time and CGI runtime account for the bulk of the delays. This suggests that investing in a network upgrade is not a valid solution. There are a number of improvements that can reduce the delay without having to upgrade either computer or network hardware: choice of browser, revision of output format and optimisation of CGI programs. In summary:

- The choice of browser and version of browser is important, as shown by the differences between Netscape and Explorer in the same situation.
- Revising the format of the results sent to the client can reduce rendering time. For example, large HTML tables take a surprisingly large amount of time. One solution to this particular overhead is to format the data into a more concise form. Reducing the data from approximately one thousand to six hundred rows without information loss

or reduction in display quality resulted in a 25% reduction in rendering time. Further improvements could conceivably be achieved through the use of pre-formatted text or the conversion of table text to images.

- CGI program times have been reduced by tackling the problem of concurrent data loading. The results of commonly called functions are now cached in shared memory regions (outside of the HTTP server) and this has drastically reduced CGI run times.

While none of these optimisations are particularly novel or complex in themselves it is important to know where to optimise, and that is what the analysis based on this timing model provides. The combined effect of the optimisations described above reduced the CURL time for the worst case Finesse page by approximately 80%.

6.2 A Revised Model of Web Delay Analysis

The initial QoS investigation was prompted by the deployment of Finesse. In a similar way, the growth in usage and extra functionality provided by other TAGS resources such as the Document Approval Tool resulted in interactive delays that prompted further work on delay analysis.

The original model identified three major components of delay: the server, the network and the client, and three major phases, the request, the response, and the processing of the response. The results from that analysis showed the network delay to be relatively insignificant, and that a surprisingly large amount of delay depended on the platform/browser combination. Accordingly, little attention was paid to analysing network behaviour. However, certain observations made during the first investigation subsequently pointed to the need for a more detailed network level analysis of web traffic. For example, it was noted that when measuring CGI run time in Finesse that the slower the client machine was, the longer a CGI script would take to run on the server. This suggested that limited client processing power was resulting in limited client buffering and that could be a cause of server transmission slowdown. In terms of the producer/consumer problem the producer is being forced to slow down by the consumer.

In the web context it means that when both the client and the server buffers are full the CGI process sleeps until more buffer space is available for its output.

This is not an HTTP problem as the producer-consumer flow-control mechanism across the network in an HTTP transaction is effectively dictated by the underlying transport level protocol (TCP). A TCP receiver exercises flow control by advertising available receiving buffer space as *window size*, the term used in sliding window protocols (Tanenbaum 1981). The rapid growth of the Internet and TCP traffic caused TCP's original flow control window to be supplemented by a *congestion avoidance* window, which is maintained by the sender. So, the use of the advertised window size for flow control is no longer the full story. The effective window used by the sender is selected from the minimum of the advertised window and the congestion window. The congestion window is adjusted by the sender according to several mechanisms involving adaptive retransmit time-outs, packet loss, duplicate acknowledgements, additive increase and multiplicative decrease.

Accordingly, TCP congestion avoidance algorithms are treated as a separate delay component in the revised model, which we refer to as the *protocol* component. Another important point about TCP congestion avoidance algorithm is that it has given rise to the notion of a fair share of network bandwidth, and this serves as an appropriate basis against which to assess the acceptability of a particular network delay component. In other words, at the data link level it is possible to say what the bandwidth of a link is, but at the Internet network level this is not possible, so it is important to have this reference when assessing network delay per se.

Before discussing each component in detail it is useful to introduce the notion of a *limitation*. The client, transport protocol, network and server limitations each have two aspects. The first is the absolute delay attributable to a component, which is independent of the other components. For example the delay between a user clicking and the initial TCP SYN packet being sent and the delay between the server receiving an HTTP GET request and commencing data transfer. The second is the delay that is dependent on a

component's interaction with other components. For example, if a client advertises a small TCP window, this will affect the network throughput, but does not constitute an absolute limit on network throughput.

6.2.1 Client Limitation

We distinguish between relative and absolute client limitation.

The relative client limitation refers to how client factors can limit the speed the server and network can deliver data. An easily detected symptom of the existence of relative client rate limitation is the reduction in size of the advertised window. In contrast with the earlier investigation, no evidence of relative client limitation was found when using modern workstations and browsers. The absolute client limitation is the amount of time, in the absence of server and network limitation that the client takes to present data to the user, a combination of Phases A and C in a CURL (see Fig 6.3). This places a limit on the reduction in overall delay that can be achieved by network and server side improvements. Absolute client limitation can be decomposed into two elements – one at the start of the connection (Phase A in a CURL, see Fig. 6.3) and one at the end of the connection (the presentation time, Phase C in a CURL, see Fig 6.3).

To determine upper and lower bounds on the clients contribution to the Phase A delay four measurement points are required; the time the user clicks on a link (C), the transmission of the clients TCP SYN segment (S), the receipt of the server's SYN/ACK segment (A) and the transmission of the HTTP GET request (G). A lower bound on Phase A is given by: $(S-C) + (G-A)$. Note that this ignores network delay. An upper bound on Phase A is given by $G-C$, which includes network delay.

Next, we consider the presentation time (Phase C). The upper bound is given by the time to display the web page from the arrival of the first data. The lower bound is the time required to display a page once all the data has been received. The gap between the lower and upper bounds can be minimised by arranging for server and network limitation to be minimised. For the purpose of making measurements this was achieved by connecting the

server and client to the same Ethernet switch and by pre-generating static HTML for output that would normally be dynamically generated.

A small harness web application was developed which consists of a server script that dynamically generates a multi-framed set of web pages and records measurements in a server side data repository. Links to the page being measured are included in the presentation frame. When a user clicks on one of these links the system time is read using the JavaScript onCLick() method and temporarily stored in a data frame. When the new page finishes loading the onLoad() Javascript method is used to record the time in the data frame. These pairs of readings are periodically downloaded to the server pending analysis. Taken together each pair defines the duration of a CURL.

Measurements of the checkpoints intermediary to the CURL's end points were obtained by using passive packet level monitoring of the client.

These techniques allow statements to be made about the client limitation, which occurs for specific web pages and specific client configurations. By carefully choosing the web pages measured and the client configurations it is possible to make general statements. This approach is not however, appropriate for the automatic monitoring of live traffic.

6.2.2 Network Limitation

A connection is said to be network limited when the *fair rate* at which the network is capable of delivering data is not fast enough to keep up with the rate at which the data is produced by the server or consumed by the client. How is the fair rate determined?

For a given network path it has been shown that Additive Increase Multiplicative Decrease (AIMD) (Chiu & Jain 1989) algorithms lead to a fair and stable distribution of bandwidth between competing connections. TCP's steady state behaviour is governed by a particular AIMD, which is called Congestion Avoidance (Jacobson & Braden. 1990). The dominance of TCP traffic on the Internet has lead to the adoption of Congestion Avoidance as a benchmark against which the fairness of other traffic is judged (Mahdavi & Floyd 1997). This means that analysis of TCP's congestion avoidance algorithm can be used to determine the fair rate at which the network delivers data and therefore the rate at

which network limitation becomes active. Under the assumption that loss is deterministic and is shared equally between connections the throughput as a function of loss probability is given by the following equation (Mathis et al. 1997) where P is the probability of loss, MSS is the maximum segment size, RTT is Round Trip Time and C is a constant term (1.079): $T = C * MSS / RTT * \sqrt{P}$.

Packet loss on the Internet is understood to implicitly signal packet loss, referred to as an Implicit Congestion Notification (ICN). In practice estimating the fair rate for a connection as a function of packet loss is complicated by two factors. A side effect of using loss as an indication of congestion is that the assumption that congestion feedback is shared evenly amongst competing streams does not hold. Consequently it has been observed that TCP's implementation of AIMD algorithms only allows statistical fairness (Wang & Crowcroft 1991) to be achieved. The bursty nature of TCP traffic means that several packets may be dropped in a single congestion event. This suggests that only one packet loss per window of data should be counted as a signal of congestion. By employing the concept of a round, which lasts from the sending of a packet until the receipt of its acknowledgment it is possible to determine whether a retransmission should be counted as an ICN.

A Round Trip Time (RTT) is taken to mean the amount of time that it takes for a segment to reach its destination, plus the amount of time that it takes for an acknowledgment to return. It does not include the time between a segment arriving and its acknowledgment being sent, which can be considerable. It is important to know the range of RTTs experienced by traffic being analysed because if the RTT is negligible, even if it takes many rounds to complete data transfer, the effect of the delay will be negligible when compared to other sources. Furthermore, if the number of rounds that are required to complete data transfer are known then the RTT can be used to calculate a lower bound on the overall network delay.

To determine the RTT it is necessary to identify which packets would generate an acknowledgment directly upon their receipt by the client. The use of *delayed*

acknowledgments by many TCP implementations means that there may be a random delay of between 0 and 200 ms between the receipt of a data packet and the generation of its acknowledgment. However, if a second Maximum Segment Size (MSS) packet is received then an acknowledgment will be generated. This is utilised to enable RTT readings to be taken. Software was constructed, based on TCPDump (tcpdump) that passively monitors network traffic to determine values for P, RTT and the MSS.

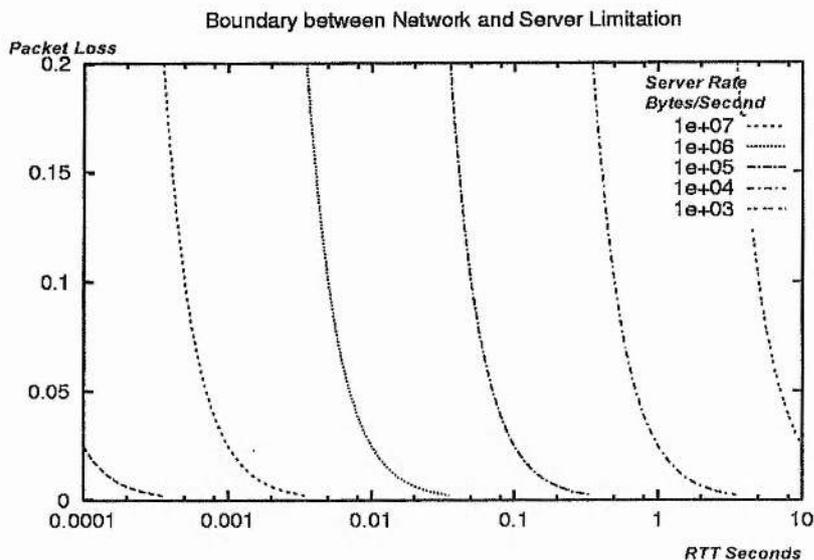


Figure 6.6: Network Limitation Boundaries

Figure 6.6 shows the boundary of network limitation for a range of round trip times and levels of congestion. Each line represents a rate in bytes per second. Co-ordinates to the left of the line are not network limited at that rate, whereas points to the right of the line are network limited.

Congestion	10 Kbytes	10 Kbytes	51 Kbytes	1.5 Mbytes
0.01%	1	1	1	10
0.1%	1	1	1	31
1%	1	2	4	95
10%	2	6	11	302
20%	3	8	15	426

Table 6.2: Expected Network Limitation in RTTs

Table 6.2 gives the expected network component of delay, which covers the range of data transfer sizes shown in Table 6.3 and for the levels of congestion found in an analysis of TAGS traffic. The results shown are for the latency attributable to the network in RTTs. They can be scaled for a particular RTT value by multiplying the value in the table by the RTT. An MSS value of 1460 bytes is assumed.

6.1.3 Protocol Limitation

In practice a number of factors mean that TCP connections often do not reach the throughput implied by the model described above. For example, TCP's loss recovery, congestion avoidance and Slow Start algorithms may cause throughput to be severely reduced. These types of factor can be thought of collectively as Protocol Limitation. The graph below (Fig 6.7) shows how these mechanisms can affect the progression of the effective window size during a TCP data transfer. The Y axis is the window size in KBytes and the X axis is time.

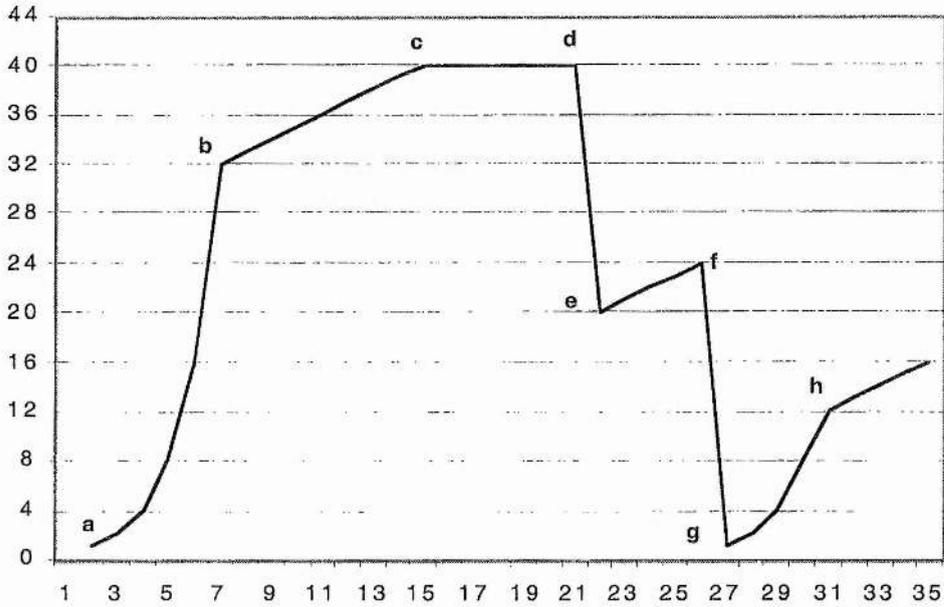


Figure 6.7: The Progression of Effective Window size in a TCP Connection

With reference to the labels:

- Phase a – b shows Slow Start occurring. The effective window (EW) is set initially to 1 segment. The Congestion Window is initially set to 32Kbytes. For every acknowledgement received before the retransmit timer fires EW is doubled, until CW is reached. This exponential increase is referred to as “Slow” because it replaces the pre-congestion aware approach, which was simply to send packets into the network at the rate suggested by the advertised window (AW).
- In phase b – c Additive Increase is evident, also called congestion avoidance. This cautiously increase the CW by one for each successful round trip, until it reaches the limit imposed by the AW, which is 40Kbyte in this case.
- Data is transferred at a steady rate in accordance with the AW between c and d, at which point a packet loss is signalled.
- Multiplicative decrease is shown in phase d – e. The CW is halved. In this case the packet loss has been signalled by three duplicate acknowledgements, and so the transmission goes into “fast recovery” mode, whereby additive increase is used to gradually grow the CW again between e – f.
- At label f, a packet loss is signalled by a retransmit time out. This is seen as much more serious than the loss signalled by duplicate acks, and the Congestion Window is reduced to one segment. A further variable, called the Threshold, is used to remember half of the most recent CW, which is 12 in this case.
- In phase g – h Slow Start is employed to reach the Threshold, at which point additive increase is employed.

If packet loss is detected as a result of a Retransmit Time Out (RTO) delay will be considerable. For this reason there have been a number of efforts to reduce the occurrence of RTOs. Random Early Detection (Floyd & Jacobson 1993) gateways seek to control the length of queues and prevent packets from being dropped in bursts, this increases the likelihood of Fast Recovery being successful and an RTO being avoided.

Selective Acknowledgements (Mathis & Floyd 1996) and New Reno (Floyd & Henderson 1999) seek to enable TCP connections to recover from multiple packet losses without an RTO. Explicit Congestion Notification (Ramakrishnan & Floyd 1998) seeks to replace packet loss as an indication of congestion with packet marking.

6.1.3.1 Slow Start, Congestion Avoidance and Transfer Size

The use of the TCP model and three dimensional graphs shown below emerged as a visualisation technique from work carried out in conjunction with Ruddle and Lindsay (Lindsay et al. 2002). It provides a reference for the analysis of protocol limitation for interactive DLE traffic. To facilitate comparison with the network limitation model the assumptions of deterministic and evenly distributed loss are maintained. The basis of the analysis is a program, which takes as input a grid of congestion probabilities and file sizes and traces the evolution of a connection's congestion window. The core of the program is shown in Figure 6.8. The program embodies similar assumptions as the TCP fair equation with the important differences that it accounts for both the influence of Slow Start at the beginning of a connection and the possibility of the receiver's advertised window size limiting throughput.

<i>Slow Start</i>	<i>Congestion Avoidance</i>
<pre> while (n > 0) { sent=cwnd/mss; last+=sent*mss; n-=sent; r++; if (last >= separation) { cwnd = max(cwnd/2,mss); last -= separation; break; } else { cwnd += cwnd/ackspace; } cwnd = min (cwnd,maxwin); } // end while </pre>	<pre> while (n > 0) { sent=cwnd/mss; last+=sent*mss; n -= sent; r++; if (last >= separation) { cwnd = max(cwnd/2,mss); last -= separation; } else { cwnd += mss / ackspace; } cwnd = min (cwnd,maxwin); } // end while </pre>

Figure 6.8: Calculating Congestion Probabilities

The central loop of the program calculates the window evolution in a round, during which it is assumed that a full window of packets are sent and acknowledged. The variables are: *n* is the number of packets that remain to be sent; *cwnd* is the size of the congestion window; *maxwin* is the maximum window size offered by the receiver; *ackspace* is

the data packets to acknowledgements ratio; *separation* is the number of bytes between congestion events and *last* is the amount of data that can be sent between packet drops. The congestion window, maximum window and separation between congestion events are all maintained in bytes. In this program, at the start of a connection the Slow Start period ends when the first loss occurs and the program moves into the Congestion Avoidance mode, which continues until all data has been transmitted and acknowledged. The rate of increase during both Slow Start and Congestion Avoidance is controlled by the *ackspace* parameter, which corresponds to the number of data packets that are required to arrive at a destination before an acknowledgement is generated.

The visualisation shown in Fig. 6.9, reproduced from (Lindsay et al. 2002), suggests that there are four distinct regions within which TCP's congestion control and flow control interact, resulting in very different behaviour. The X-axis is the log of the transfer size in packets. The Y-axis is the log of the Fair Window size. The Z-axis is the ratio: $R = \text{Average} / \min(\text{File Size}, \text{Fair Window})$

- In each graph there is a trough that stretches from the top right corner down and to the left. This is caused by the static initialisation of TCP to a small initial window size. Connections within this trough are not sufficiently long-lived to be able to probe the network for available bandwidth. Consequently even if the bandwidth is available and unused by other traffic these short TCP connections will not be able to utilise it. Much web traffic falls into this trough.
- The second region consists of a ridge where TCP connections receive more than their fair share of bandwidth. In the bottom two graphs the ridge runs from the top right hand corner of small transfers and high congestion to the bottom left hand corner of small congestion and large transfers. It is caused by the exponential increase of the Slow Start algorithm allowing connections of a certain size to receive more than their fair share of bandwidth. To the left of the ridge is the area within which TCP's average behaviour approximates to the steady state. To the right of the ridge TCP's average window size is smaller than the steady state average.

- The third region is a second trough, which stretches from the bottom left to bottom right hand corner of each graph. In this region loss levels are low, resulting in a large fair window size. However, TCP's flow control window prevents full utilisation of this bandwidth.
- The fourth region is in the top left hand corner of each graph. Here the combination of large transfer sizes and relatively high levels of loss mean that TCP's Congestion Avoidance algorithm dominates the average behaviour of connections. Consequently performance is close to optimal. It is interesting to note that these conditions correspond to those that were prevalent in the late eighties when TCP's congestion control algorithms were being designed and validated.

Figure 6.9 also illustrates the effect of the interaction between TCP's flow control and congestion control window.

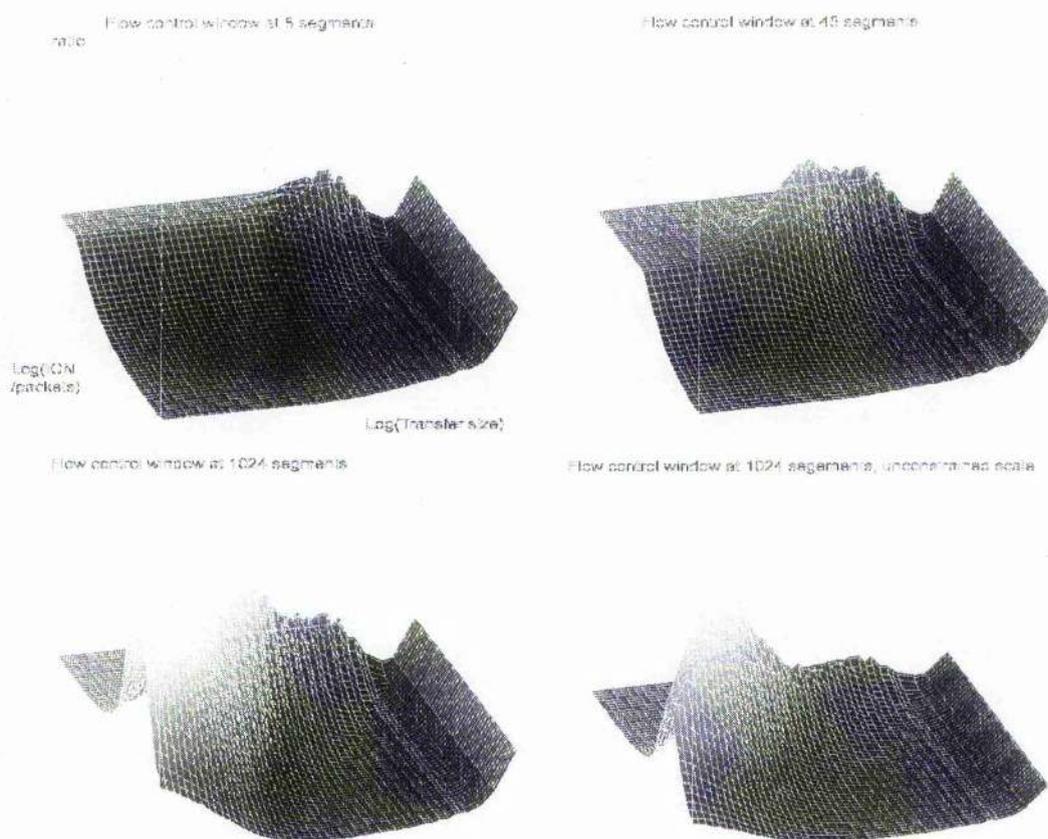


Figure 6.9: 3D Representation of TCP behaviour for ranges of flow control window size, data transfer size, and loss probability. Reproduced from (Lindsay et al. 2002).

The lower two graphs show the advertised (Flow Control) window parameter set to a large value (1024 segments). In this case the size of the overshoot caused by Slow Start is quite considerable. The top two graphs show the flow control window set to 8 and 45 segments respectively which effectively limits the aggressive behaviour of TCP. A MSS size of 536 bytes and window sizes of 4Kbytes were common in the late 1980s and correspond to the top left graph. Today the most common MSS size is 1460 bytes and 45 such segments equate to a 65 Kbyte window that is in turn larger than at least 90% of today's advertised window. The bottom two graphs correspond to a possible future where cheaper memory allows large flow-control windows. It can be concluded that TCP's flow control window has played an important role in capping the overshoot caused by Slow Start. This illustrates that the problem of overshoot was not a serious problem in the past although a significant body of work addressed it, nor is it a problem at the moment. However, as flow control windows and bandwidth increase it is likely to become more of an issue in the future.

In summary these graphs show that TCP's congestion control algorithms closely approximate fair bandwidth utilisation, when there is only one packet of data and when connections are large but the fair bandwidth is limited. This corresponds to the network conditions and traffic patterns against which Slow Start was validated and which were thought to be prevalent on the Internet when congestion control was introduced to TCP. However, where transfer sizes are large, congestion is low, and large advertised windows are used Slow Start will make TCP aggressive with respect to fair share equation. Where connections are small in size, as is the case with much Web traffic, the static initialisation of TCP limits utilisation of the available bandwidth. Therefore protocol limitation *must* be accounted for when determining the sources of delay in interactive Web applications.

6.1.4 Server Limitation

As with client limitation there are two elements to server limitation; an absolute amount, which contributes to the latency of each connection and a rate, which places a bound on

the speed with which data can be transferred. Absolute server limitation is the time taken for the server to process an HTTP GET request and start delivering data. This is the server's contribution to phase A in the structured timing model. It may be determined by reading packet level traces captured at the server and is the difference between the arrival time of the GET request and the departure time of the of the first data packet.

To determine if a connection is server limited the relationship between the window size utilised during a connection can be compared with the congestion and advertised window. If the utilised window is smaller than the minimum of the congestion window and the advertised window then at that point in the connections lifetime it can be said to be server limited.

The advertised window can be obtained directly from TCP packet headers. TCP's Slow Start and Congestion Avoidance algorithms are well known and packet losses can be detected by the retransmission of dropped packets so the congestion window evolution can be calculated.

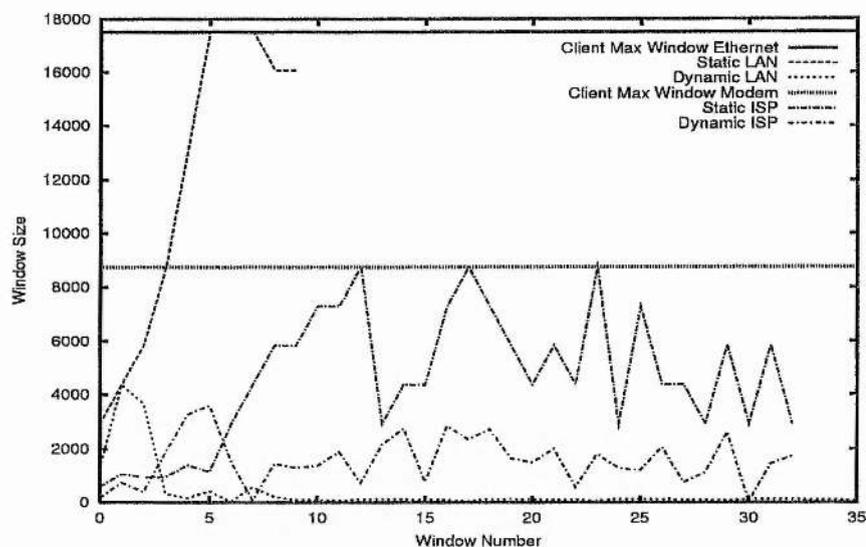


Figure 6.10 Utilised Window Sizes

Samples of the utilised window may be obtained by selecting a packet and then counting the number of data bytes transmitted between the sending of that packet and the receipt of

its acknowledgment. This value is the *utilised* window. The traces of window sizes for dynamically generated web pages in Figure 6.10 illustrate server limitation.

Once the existence of server limitation is established it can be quantified. This is achieved by taking readings of the times the first and last data packets were transmitted for each connection. The size of each transfer is known so the rate can be calculated as quantity of data over the data transfer time. To measure the size of server limitation for a particular server, load and web page it is simplest to arrange that client, protocol and network limitation are likely to be smaller than the server limitation. This can be achieved by using a powerful lightly loaded client connected to the server by a high bandwidth lightly used network.

6.2 Applying the revised model to TAGS

TAGS educational resources tend to be highly interactive and generate a two-way flow of information that stands in contrast to the simple client pull model of static web pages. The dynamic generation of these pages on the server means that caching is of limited use in reducing latency. Also, there is a stronger locality of reference than exists for much web traffic. A user can be expected to return repeatedly to the same set of pages during the lifetime of a course and may need to perform many operations whilst using a single resource. Much of the time users have high bandwidth access to the Internet and high bandwidth pathways to the TAGS server. For a relatively small proportion of the time, when the system is being accessed from home or whilst away from the host institution, much smaller bandwidths (modem/ISP routes) are available. There is therefore a wide variation in the network conditions that user's experience.

Two techniques were used to collect data on TAGS usage. Firstly the server logs were analysed to determine the distribution of transfer sizes. The results over a three-month period are shown in Table 6.3 and show that the distribution of file size is in line with that found in general on the Web. This informed the selection of a range of file sizes

associated with data from the second source, TCPDump³ packet level traces. These traces were post processed to determine the levels of congestion and the distributions of network RTTs.

Size (Bytes)	% of transfers	% of bytes
0	17.59	
11 - 100	0.56	
101 - 1K	35.55	1.27
1K - 10K	35.19	13.55
11K - 100K	10.51	35.62
100K - 1M	0.55	15.49
100K - 1M	0.06	26.33
11M - 100M	0.002	7.74

Table 6.3: Distribution of file transfer sizes

The following transfer types were selected for detailed analyses:

- Text file: a 27KB text file which corresponds to a TAGS help file,
- A 10 KB small form of the sort used to enter the marks for a small tutorial group,
- A dynamically generated 7 KB small table of the sort used to view information about a tutorial group
- A 51KB module overview table, dynamically generated from a database.

6.2.1 Client Limitation

Observations of TAGS packet traces found very few examples of client limitation and so it is assumed that it is of negligible significance in impacting the responsiveness. Results for absolute client limitation are presented in Table 6.4, The values represent the addition of the client parts of Phase A and Phase C, (see Figure 6.3).

³ Detailed network traffic data can be obtained by running a network snooping program such as TCPDump in the same collision domain as the server or client, or utilising kernel-level packet

File	Linux Netscape 4.7	Windows 2000 Netscape 4.7	Windows 2000 Netscape 6	Windows 2000 IE5
Static Text	0.124	0.060	0.220	0.070
Small Form	0.113	0.381	0.350	0.100
Small Table	0.090	0.100	0.240	0.150
Large Table	0.136	0.470	0.451	0.201

Table 6.4: Absolute Client Limitation in Seconds

6.2.2 Server Limitation

Measurements for the limitation attributable to the server on are presented in Table 6.5. The first column shows the average absolute limitation for each transfer type. The second and third columns show the transfer delay and the data transfer rate for the NSC phase in the CURL component model. The fourth column shows whether data transfer was server-limited. Despite the low RTTs and absence of congestion the transfer of static text files did not show server limitation. In contrast the dynamically generated TAGS traffic does show a high level of server limitation. Here there is a further contrast between the transfers that draw their data from a database, which performed better than the transfers where the data was read from flat files, which displayed stronger limitation.

File	Mean	Abs	Rate, Kb/s	Limited?
Static Text	0.171	0.001	1257	N
Small Table	2.231	0.137	27.2	Y
Small Form	2.230	0.189	37.4	Y
Large Table	11.857	0.137	34.4	Y

Table 6.5: Server Limitation in Seconds

6.2.3 Network Limitation

It would be expected that the server limitation rate would remain constant over a range of RTTs and congestion regimes. To test the hypothesis that server rate would remain

monitoring. This allows for reasonably non-intrusive monitoring of IP traffic of interest.

unchanged across a range of network conditions a number of transfers were undertaken using a 56 Kb/s modem to dial-in to a well known ISP. The results are shown in table 6.6. It can be seen that there is little change in the server rates for the dynamic files confirming the hypothesis and suggesting that the connections remain server limited. There is a significant decrease in the rate for the static HTML file which can be explained by an increase in the strength of network limitation.

File	Mean	Min	Rate Kb/s
Text File	3.172	3.057	67.8
Small Table	2.933	2.307	20.7
Small Form	2.852	1.759	29.3
Large Table	12.557	11.774	32.5

Table 6.6: Modem and ISP Data Transfer

6.2.4 Comparison of Client, Network, Protocol and Server Limitations

A structured timing model of the delay has been evolved, which facilitates an analysis of the proportion of delay that can be attributed to the network, transport protocol, client and server. Having discussed these limitations separately we are now in a position to compare the relative importance of each. Results indicate that for dynamically generated files server limitation is the most important factor. In the case of the larger dynamically generated tables, the server limitation is in the order of 10 seconds. The contribution of client limitation is in the order of 100s of milliseconds, most of which is rendering time. For network conditions with congestion less than 10% and RTTs of less than 100ms the combined value for protocol and network limitation will be less than ten round trip times or one second. So, while for many users of the system, as discovered in the first QoS investigation, the network component is negligible, for some it is not.

In the case of statically generated pages a different picture emerges. For paths with a significant RTT, and relatively low levels of loss, protocol limitation adds significantly to the latency experienced by the user. This suggests the need to address the mismatch between Web traffic and TCP's congestion control algorithms.

For the transfers studied it was established that under network conditions of low loss and small RTTs the main contributing factor to latency is server side limitation. However, even if measures are taken to reduce the extent of server limitation, network and protocol limitation remain important for a significant proportion of connections.

For certain file types, such as those containing large tables, client side limitation is also significant. In the example used it was approximately one third of a second.

6.3 Summary: Being QoS Aware

DLEs constructed using the TAGS framework are highly interactive distributed applications, and delay, as experienced by the user, is a key QoS parameter. A DLE should be aware of the values of this operational parameter in order to facilitate short-term adaptation and longer term change in the way a service is provided. The importance of these QoS investigations lies in their pointing the way to a means of automatically monitoring service degradation in terms of interactive responsiveness and identifying its source. In particular, server-side delay can be detected by comparing the advertised client window with the server production rate and the potential fair network throughput. If a server is deemed to be the source of an unacceptably large delay component then it is the responsibility of the service provider or application developer to take action. If the source of delay lies primarily with the network path then either the service provider or the client may be advised to use a different network service provider. If the problem lies with the client we assume that if a modern desktop computer is being used this will be due to rendering time. The service provider can change the way the data is formatted, or the user could be advised to change their browser or upgrade their computer. If the problem lies with protocol interaction between HTTP/TCP then it is more difficult to recommend corrective action as these two protocols are enshrined in the web's operational model.

7 Addressing the QoS Requirements for Interactive Resources

This chapter describes the design and prototyping of architectures that address the respective QoS needs of two generic interactive resource types, those that are web-server based, and interactive continuous media. Server-based learning resources are the subject of client-server interaction across the network, in a many-to-one pattern of communication. By contrast, interactive continuous media sessions may be initiated and co-ordinated from a server, but feature data flows to and from all participants in a peer-to-peer multicast pattern. Such resource types include multi-way audio/video conferencing, shared virtual reality and “co-laboratories”. We refer to the QoS requirement of the former as *responsiveness* and the latter as *timeliness*. These respective requirements differ principally in their degrees of delay sensitivity and loss tolerance, as summarised in Table 7.1.

QoS Requirement	Interactive Resource Type	Example Application	Bandwidth	Delay Sensitive?	Jitter Sensitive?	Loss Tolerant?
Timeliness	<i>Continuous Media</i>	Interactive video	high	< 10 ms	yes	yes
		Interactive audio	low-medium	< 300 ms	yes	yes
Responsiveness	<i>Web-Server Based</i>	CGI-based web pages	low - high	< 5000 ms	no	no

Table 7.1: QoS Characteristics of interactive continuous media and interactive responsiveness

Figure 7.1 summarises the relationship between these generic resource types and Internet protocols. In the case of responsiveness the HTTP protocol is predominantly used as the application level protocol. This is TCP-based, which satisfies the reliability requirement. With respect to delay sensitivity, neither of the two common IP data transports, TCP and UDP provide time guarantees. However, timeliness is much more delay sensitive than responsiveness, so the choice of transport can prove

critical. TCP is connection-oriented and maintains several timers as part of its reliability and congestion control features. This has the effect of making TCP potentially worse for time-sensitive traffic as it will defer transmission on occasions, and retransmit on others, when, for continuous media, the retransmission will arrive too late to be of use. Accordingly, most time-sensitive continuous media applications on the Internet make use of the UDP-based Real Time Protocol, RTP, which is described later in this Chapter. As RTP is UDP based, it can be used in conjunction with IP multicast, an important means of making good use of available bandwidth for video.

Distributed Learning Environment			
<i>Multiple Interactive Resources with Different QoS requirements</i>			
Responsiveness		Timeliness	
Shared server-based resources, may be replicated		peer-to-peer multicast media flows	
Hyper Text Transfer Protocol (HTTP)	other application level protocols	Real Time Protocol (RTP) <i>a time-aware protocol</i>	other application level protocols
Transmission Control Protocol (TCP) <i>two-party reliable data exchange, no time guarantees</i>		User Datagram Protocol (UDP) <i>best effort delivery, no time guarantees</i>	
Internet Protocol (IP) <i>unreliable, best effort, no time guarantees</i>			

Figure 7.1: Overview showing the major distinction between reliability and timeliness

How can these respective QoS requirements best be met? In the case of responsiveness, a server delay may be reduced by increasing the processing power of the service. In section 7.1 the use of multiple servers harnessed by a replicated resource architecture is proposed to achieve better responsiveness under widely

varying loads. In the case of timeliness the network limitations of bandwidth, delay and jitter are often outside the control or influence of both the clients and the service providers, especially if connections are made via a commercial ISP. In this case it is arguably better to adapt. Accordingly, in section 7.2 an adaptive Conference Control Architecture is proposed for addressing the QoS requirements of interactive continuous media.

Integration

Figure 7.1 shows that a DLE consists of a variety of interactive resources. In addition to supporting the network QoS requirements, it is also important that solutions can be assimilated into the framework for a DLE. This means, for example, that the management facilities for creating and allocating an instance of a conference resource type to a group should be the same as those used for other server-based resources, and that the means of specifying replication and coherence should be consistent with existing interfaces for resource type registration and instance creation. In other words, it is important to maintain an integrated framework (Allison et al. 1999). On the other side of the coin, it is also important that a framework can accommodate mechanisms for addressing QoS requirements.

7.1 Interactive Responsiveness – A Multiple Server Approach

Chapter 6 showed that server limitation is often a significant source of delay in a web interaction. There are broadly two approaches, of a complementary nature, which may be taken to address this performance problem: software tuning and hardware upgrade. In the case of the former, an approach based on application execution profiling, followed by an analysis to identify the code sections where most of the execution time is being spent, and then rewriting the code may deliver some benefits. For example,

after the initial QoS analysis of Finesse various changes to the concurrency handling and list processing sections of the code resulted in large improvements. More recently Finesse has been rewritten using Java servlets (Nicoll et al. 2002), and this has resulted in another significant improvement. However, the benefits of such improvements are only short-lived if the user population, and their usage of the resources, keeps growing. In the case of Finesse it is now probable that even large amounts of further software tuning will only result in relatively small performance increases, and would not justify the effort.

In the case of a hardware upgrade approach, initially provisioning a monolithic system of sufficient capability to meet peak demands is not a good, or even pragmatic solution, as small changes in the scheduling of the content or the size of the user base can quickly invalidate predictions about peaks. Consider an induction period for a class of two hundred students who are to use a DLE - that one hour period can easily generate a load which is an order of magnitude above the average.

An alternative hardware approach is to improve performance by using more than one physical server. Using multiple servers is an interesting strategy because it also has the potential to provide incremental scalability whilst maintaining responsiveness in the face of dynamically changing patterns of usage and load. If a DLE is distributed across multiple servers, and client requests are also distributed across the available servers, then there is less likelihood of any individual server limitation becoming a service bottleneck.

Availability is another desirable QoS feature for DLEs. Although it does not come automatically with multiple servers, the potential redundancy associated with multiplicity intuitively provides a good basis for its provision. If there is more than one server that contains copies of resources assigned to one or more groups, then users

belonging to these groups can escape server-side problems, provided that a fault management system is in place.

QoS is typically seen as an entirely technical issue, but in practice the financial cost of service provision must be considered when making any technical decision. This is especially true in education, where financial resources are subject to tight restrictions. So, when considering approaches to increasing the power of a service, *affordability* must be taken into account. Commodity clusters consist of off-the-shelf components that offer good price/performance and can be updated regularly and incrementally. Individual processor nodes or the underlying interconnection network may be upgraded independently as funds permit. It is also possible to construct a heterogeneous cluster using CPU technology from multiple vendors.

In contrast to more traditional proprietary designs, this economy can be maintained throughout technological advance as newer, higher performance components become available. The case study

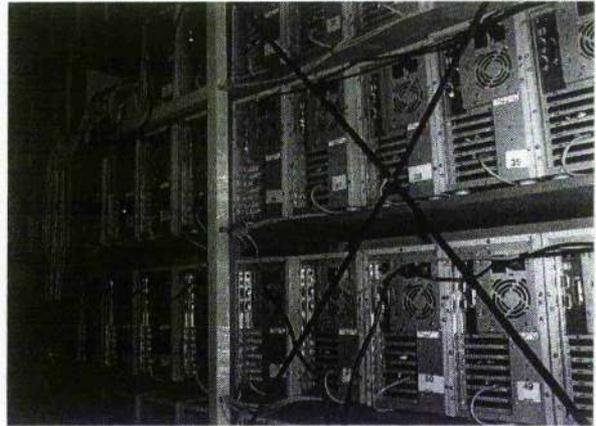


Figure 7.2: A Commodity Cluster

described later in this chapter made use of the cluster pictured in Figure 7.2 consisting of commodity computers (Pentium-based PCs), connected by a commodity network (fast, switched ethernet), and running a commodity operating system (Linux).

Although clusters conventionally consist of closely located computers, high performance wide area clustering is feasible if guaranteed minimum bandwidth and maximum delay communication links can be arranged (Allison 1998). As noted in Chapter 2, this is not possible on the conventional Internet. However, experimental

components of the Internet2 networks and SuperJANET-4 are aiming to pilot guaranteed bandwidth routes. The Internet2 Qbone, described in Chapter 3, obtains predictable levels of bandwidth using the Differentiated Services model (Blake & al 1998). ATM-based metropolitan area networks such as those used in the Scottish MANs have had the capability of reserving inter-site bandwidth on virtual channels since 1996, but relatively few end-users have been allowed access to these facilities.

The drawback with commodity clusters, is that the constituent parts of the distributed system were not designed to work with each other in a seamless fashion to provide a uniform resilient network service. This is why it is necessary to design an architecture to harness the potential benefits of multiple servers.

In summary, the use of multiple servers within a DLE is of interest on four counts: responsiveness, incremental scalability, availability and affordability.

7.1.1 The Case for Replication

If a service is to be provided by multiple servers it does not automatically follow that its components must be replicated. To explain why replication is attractive we ask the following question: *how are DLE service components to be distributed across multiple servers?*

The service components in TAGS are the resource instances, users and groupings that make up a particular DLE. Policies for distributing components include:

- By component type. Place all components of the same type on a single server. Any DLE consisting of more than one component type will then use multiple servers. The advantage of this approach is that resources can be located where they will obtain appropriate computing power. For example, demanding resources such as interactive scientific simulations can be placed on high-powered nodes.

- By DLE. Place all the components of a particular DLE on a single server. Each DLE will then be relatively self-contained and will not suffer interference and variance in performance caused by the independent usage of another DLE. This may also suit service administrators.
- By domain. Place all the components of a domain on a single server. A DLE may consist of more than one domain, but to the extent that a domain is an administrative unit this may prove useful for management purposes.
- By user resource set. Allocate a “home server” to each user, and place all their resources on that same server. If the server is topologically close to the user’s usual client machines in terms of network access then this will distribute the load by locality, for example, by campus.

While all these approaches can spread the load, they fail to support availability. The single point of failure in the case of the single server service is being replaced by multiple potential points of failure. On the other hand, providing a service based on distributed, fully replicated servers can negate periods of non-availability caused by server and network outages. It follows that for replication to provide an effective basis for supporting availability in a DLE *all the components of that DLE must be replicated*

7.1.2 A Replicated Resource Architecture

A TAGS resource consists of a set of components such as CGI programs, servlets, static HTML pages, applets, streaming media, multi-way conferencing channels and so on. Figure 7.3 gives a conceptual overview of a resource replication architecture (RRA) illustrated in the context of a DLE service provided by four server nodes. The layers are labeled 1 - 6. Layers 3, 4 and 5 are partitioned into four nodes. The functions of each layer are summarised below:

<i>Layer</i>	<i>Description</i>
1	Web browsers are used as client software.
2	Connections to a service pass through a single access point and load balancing mechanism. This masks the actual number of nodes in service, and may trigger attempts to recruit new nodes, or to shrink the pool of active servers, dependent on load. This may be implemented using one or more Internet protocols and/or technologies such as DNS Round Robin, http redirect, TCP Header Rewriting. Commercial products which implement this functionality are available from IBM and Cisco.
3	One conventional web server runs on each node.
4	One or more CGI programs and support programs run on each node.
5	One <i>coherence server</i> runs on each node.
6	The persistent state of the DLE is identical when viewed from any node. In practice the state is physically replicated, but should appear identical on every node when there are no outstanding messages (client requests, client responses, coherence messages) in the system.

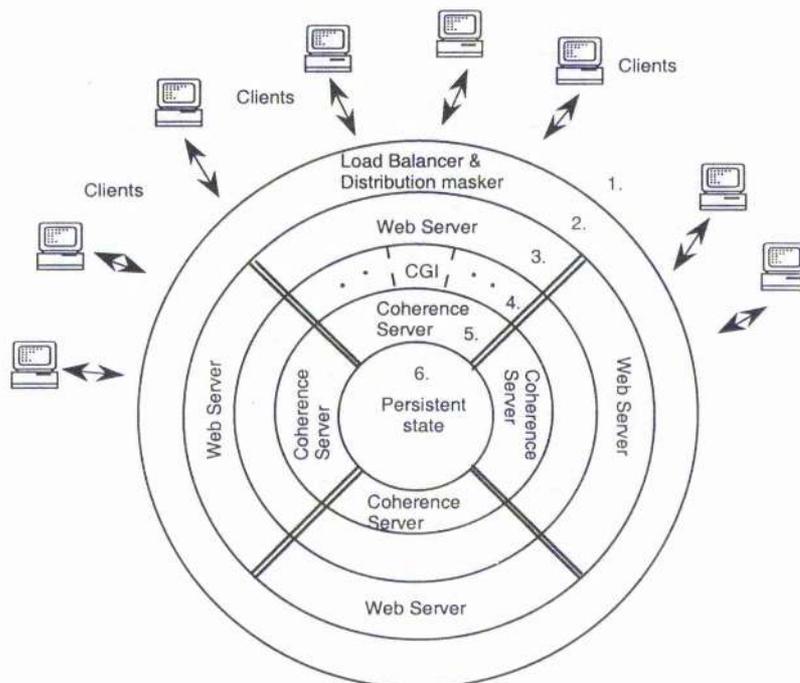


Figure 7.3: A conceptual overview of a Replication Architecture, illustrated with a 4 server scenario.

The issues raised by the core layer (6) are at the heart of the problems facing the successful construction of a replica-based service. Potential conflict may arise when two or more clients attempt to modify the state of a resource instance at either the same node, or on different nodes. If the time taken to stabilise an update, which may occur at any node, and which may involve resolving conflict, is too high then the illusion of a common shared state will not be maintained. In other words, the distributed, replicated version of the DLE will offer poorer performance to the end user. On the other hand, if a synchronisation mechanism is too optimistic then it runs the risk of breaking the common shared state by leaving the system in an inconsistent state. These issues are addressed by layer 5, which implements replication and coherence policies.

7.1.3 Types of Replication and Coherence

Replication can be implemented in various ways, depending on the type of service required. The taxonomy of replication shown in Fig. 7.4 identifies the following characteristics:

- passive vs. active
- read-only vs. read-write
- types of read-write coherence models

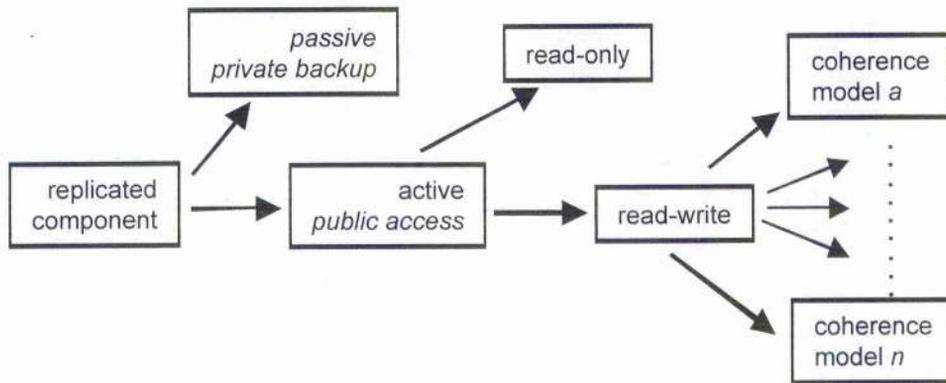


Figure 7.4: A Taxonomy of Replication Characteristics

A passive replica is a hidden backup. It is epitomised by the “Primary-Backup Approach” (Budhiraja et al. 1993). Clients of a service are not aware of the existence of backups. *Active replication* refers to the situation where any replica can interact directly with a client. The next major issue is the meaning of “interactivity” – in the read-only situation interactivity allows searching and navigation through a hypertext environment. In the read-write situation the state of the replicated service components can be changed by clients. In this situation the coherence problem emerges – how to maintain the same state at all the replicas in

the face of multiple concurrent readers and writers. Different coherence models are identified and discussed as part the replication architecture described later in this chapter, and the choice of synchronisation mechanisms to enforce these models.

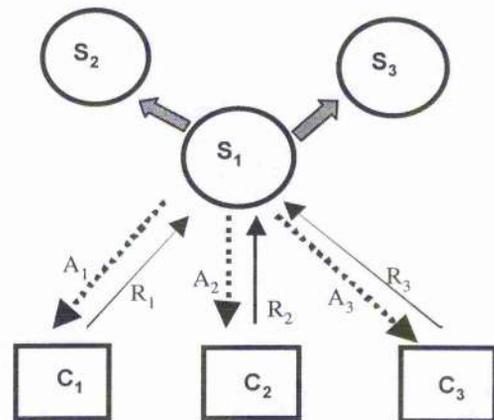


Figure 7.5: Primary-Backup with S2 and S3 acting as backup for S1

7.1.3.1 The Primary-Backup Passive Approach

In this type of approach all requests are sent to the primary server. The primary satisfies each request and updates the state on the backup(s). It may do this synchronously, and keep the client waiting, or asynchronously and respond more

quickly to the client. The backup(s) and the primary keep in touch with each other using “are you still there” messages. In the event of the primary failing, a “failover” takes place and the backup takes over from the primary. It is assumed that either i) the clients know how to reach the backup server; or that ii) the backup can substitute itself for the primary, including network address, and thereby catch traffic for the primary.

In Figure 7.5 requests R1, R2 and R3 are all sent to the primary server, S1. S1 updates the copies of its state on backups S2 and S3. If S1 fails then it is possible that requests which are in transit will be lost. (The state-machine approach described later does not have this problem).

Another potential drawback with this approach is that the backup servers are not actively sharing the load when generated by client traffic, and performance will not be enhanced. In practice server nodes may act as backups for each other. Further problems include reliable failure detection and re-admission of a recovered server.

7.1.3.2 Read-only Replication

The degree of interactivity is a major consideration. That is, the extent that a service provides read-only information which is periodically updated as part of that service, but not in direct response to client interaction. A web site that allows interactivity by navigation of hyperlinks and use of search facilities is considered to be read-only. For example, *mirroring* is frequently used for read-only services such as software archives and sports news. The UK academic mirror service¹ is motivated by the need to save on the financial cost of bandwidth usage outside the academic network, but also provides a potentially better service in that client-server connections span much less of the Internet than would be the case otherwise. In other words, the collection of read-only

¹ <http://www.mirror.ac.uk/>

replicas is being used as a cache for popular software archives. Major sporting events such the World Cup and the Olympic games attract a very large amount of web traffic, and distributed server replication using mirroring was one of the strategies adopted by IBM (Iyengar et al. 2000) and Hewlett-Packard (Arlitt & Jin 2000) to cope with the very high volumes of demand for sports results in real time.

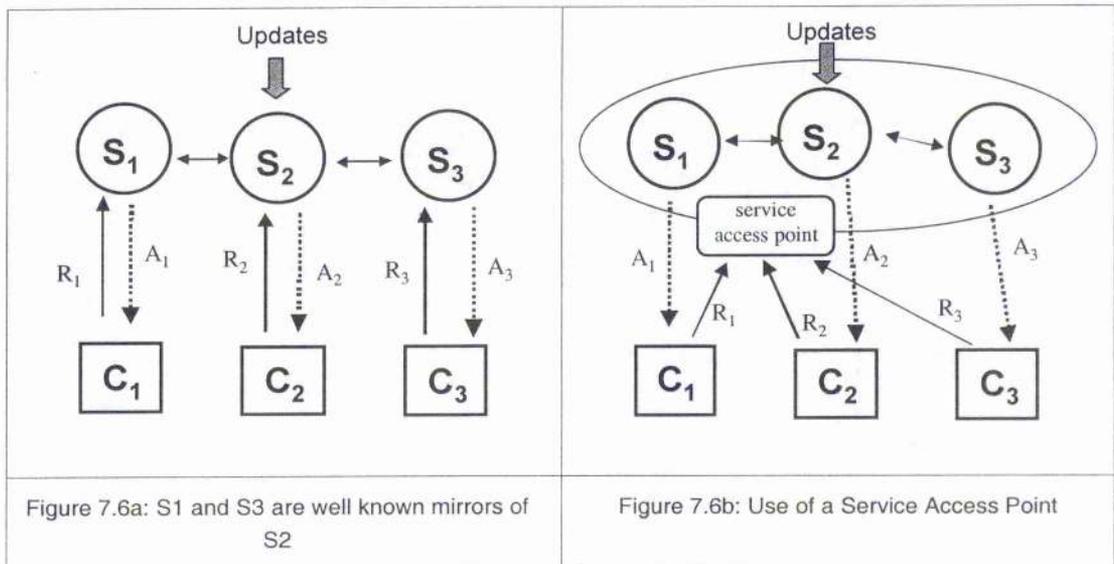


Figure 7.6 illustrates variations of a read-only replication service. Updates applied to server S2 are asynchronously mirrored to Servers S1 and S3. In 7.6a clients C₁ and C₃ send information requests R₁ and R₂ to well-known mirrors S1 and S3. Clients C₁ and C₃ receive answers A₁ and A₂ from these mirrors. A more subtle version of the read-only mirroring approach is shown in 7.6b. All requests are sent to a service access point (typically a URL). The service distributes requests R₁ R₂ R₃ internally; answers A₁, A₂ and A₃ are returned from different servers.

7.1.3.3 Active Replication

Mirroring, although simple and effective for read-only information services and archive sites is not suitable for interactive services where the state of the server is directly modified by client interaction, as is the case in a DLE. We refer to this type of

situation as *active replication*. The traditional approach taken to active replication is referred to as *state-machine* (Schneider 1993). This approach pre-dates the dramatic growth in Internet usage and the advent of the Web, and can now be seen as a special case of a wider variety of approaches based on pragmatic considerations of Internet technologies. For example, the state machine approach assumes that a system designer will be providing both client and server software in a closed universe system, and that it can therefore be arranged that each client broadcasts all requests to all servers. In the Web, the client software is assumed to be a browser, and the server software an http server. However, there is no standard mechanism for a browser to send the same http request to multiple servers, and no standard protocol for multiple http servers to co-operate in the selection of which one server will respond to a request.

7.1.3.4 · The State Machine Approach

In the state machine approach a replica is characterised as consisting of a state, and some atomic, serialisable, operations which can modify that state. All the clients know the addresses of all the servers and send each request to all servers. The servers implement a coherence protocol so

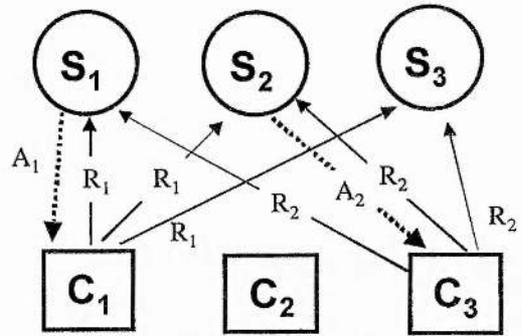


Figure 7.7: Active Replication using the "state machine" approach

that exactly one of them responds to each client request. The coherence protocol must operate synchronously to some extent as a decision must be made as to which server will respond. Figure 7.7 illustrates a typical example: clients C1 and C3 send requests R1 and R2 to all known servers, S1-S3. The servers reach agreement amongst themselves using an intra-server coherence protocol, and S1 responds to R1 with A1, while S2 responds to R2 with A2. S3 does not respond, but its state is identical to that of S1 and S2, so it can easily respond to future requests without compromising the integrity of the service.

The advantage of the state machine approach is that in an N server service, N-1 failures can be tolerated. The disadvantage is that there is no standard protocol that allows for broadcasting an http request to multiple servers simultaneously.

7.1.3.5 Coherence Models for Replication

A coherence model in the context of replication has two main properties: reliability and event ordering. The types of event ordering supported are based on those identified by Hadzilacos and Toueg (Hadzilacos & Toueg 1993), which are similar to those found in ISIS (Birman 1991) and related projects. In the case of the RRA the events are

updates to the state of an instance of a specific resource type. The exact nature of an update will depend on the type of the resource.

Although some resource types may not require reliability, for example continuous media streams, for most resources reliable updates will be essential. Reliability properties are:

Validity: If a correct resource transmits an update, U, then all correct replicas eventually receive U.

Agreement: If a correct resource applies an update U, then all correct replicas eventually apply U.

Integrity: For any update U, every correct replica applies U at most once, and only if some resource transmitted U.

The basic event orderings are:

Unordered: Updates are eventually applied at all replicas, but in no guaranteed order.

Source ordered (FIFO): If a correct site transmits an update, U, before it transmits an update U' then all correct sites only apply U' iff they have already applied U

Causal: If the transmission of an update U causally precedes the transmission of U' then no correct site applies U' unless it has already applied U.

These basic orderings result in a *partial* event ordering. That is, concurrency² is possible. *Total* orderings are defined thus: If correct sites S1 and S2 both apply updates U and U', then S1 applies U before U' iff S2 applies U before U'. The notion

of a total ordering is orthogonal to the basic orderings, but is opposed to concurrency. A total ordering can be combined with the basic orderings i.e. FIFO Atomic and Causal Atomic. Table 7.2 summarises the range of update types that are possible.

Update Type	Ordering Characteristics
Unreliable	best effort, unordered
Reliable (Unordered)	Validity + Agreement + Integrity
FIFO	Reliable + Fifo Order
Causal	Reliable + Causal Order
Atomic	Reliable + Total Order
FIFO Atomic	Reliable + Total + FIFO
Causal Atomic	Reliable + Total + Causal

Table 7.2: Update Reliability and Orderings

7.1.4 Replication Strategy

A goal of the replication architecture is to minimise the delay caused by replica synchronisation overhead, and this in turn raises two inter-related questions:

- what type of synchronisation is appropriate? for example, are synchronous atomic transactions suitable for all cases?
- must synchronisation be synchronous (blocking) and add delay to a response, or asynchronous(non-blocking), and therefore more responsive?

We distinguish initially between resource-specific requirements and generic service needs. If a resource is shared between many concurrent users then it will require resource-specific concurrency control. If a resource is the focus of a collaborative

² Concurrency according to Lamport's definition described in Chapter 2.

effort then it will also need specific multi-user awareness features. Whenever a resource is replicated it will need *resource-specific replication coherence*. The motivation for distinguishing between resource-specific types of replication coherence is twofold. Firstly, different types of resource will have their own functional requirements with respect to QoS and replication, and the system should aim to *efficiently* match these with appropriate replication mechanisms. Secondly, it is desirable to avoid a “one-size-fits-all” approach because the synchronisation of replica updates must always be able to satisfy the most stringent requirements. For example, the use of synchronous, blocking, distributed atomic transactions. The undesirable consequence is that all resources, including those with relatively relaxed coherence requirements, must pay the same high price in terms of delay and protocol complexity. So, with respect to these issues the philosophy that has been adopted is:

- to use the TAGS resource type as the unit of replication management
- to offer a choice of replication coherence models, to be chosen from according to a resource’s semantics
- to offer the option of blocking or non-blocking response, dependent on the resource’s semantics

7.1.5 Coherence Strategy

The coherence server design is based on the Twarp approach described in Chapter 4, in that a DLE is defined as consisting of one or more nodes, and each node executes a *coherence server* in

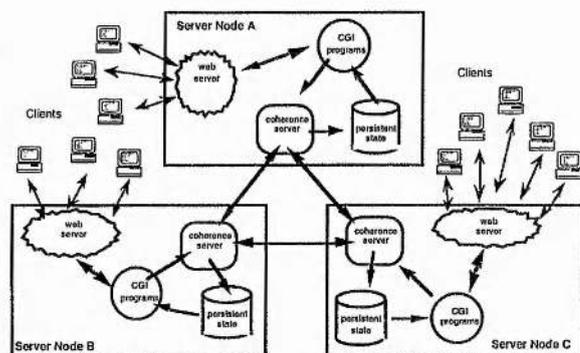


Figure 7.7: A Three Node Service: All Node-to-node communication is carried out by the coherence servers.

parallel with the web server and the other resource-specific code being executed. Fig.7.8 illustrates a three-node service. It is assumed that clients have been allocated primary server addresses, or that a dispatcher redirects them to a particular server node. The coherence server implements the range of update orderings listed in Table 7.2. Each resource type is represented by a *handler* which handles communication with the coherence server. The coherence server ensures integrity of a shared object in the face of local concurrent updates (if necessary) and global replication. Figure 7.9 illustrates the roles of Resource Handlers and Coherence Servers in an example scenario for the RRA. Five physical nodes are shown. For clarity web servers and resource handlers have been omitted from nodes 2-5 in the diagram, although they would typically co-exist as shown on node 1. The Groups and Resources Management resource (GRM) is shown as all requests are still subject to authentication and access control. The GRM is also extended to include information on where resource instances are replicated, including itself.

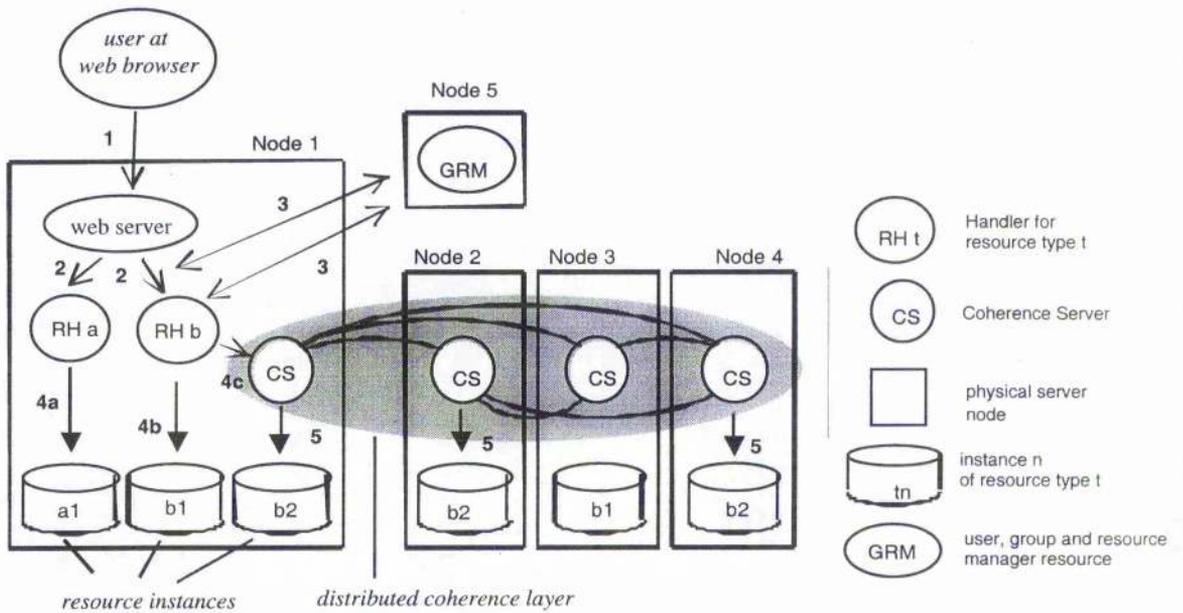


Figure 7.9: RRA coherence management example

The progress of a user's request is illustrated in the diagram by stages 1 - 5.

- 1) The user triggers a request to a resource from a browser by selecting the URL of the appropriate resource handler.
- 2) The web server receives the HTTP request and invokes the resource handler, passing along all the information provided by the user in the CGI parameters (instance, user, command, etc.)
- 3) The resource handler asks the GRM whether the user is allowed to access the requested instance and where it is replicated.
- 4)
 - a) If the instance is not replicated the resource handler accesses it immediately, modifies the state according to the request, and sends an appropriate response to the user.
 - b) If the instance of the resource is replicated and the request is a read, then there is no need to broadcast the request, as it will not change the global state of any of the replicas of the instance. The resource handler will read its local copy and return the result to the user. This is an optimisation distinct from resource-specific requirements.
 - c) If the instance is replicated, and the request is a write, the resource handler sends the request to its local coherence server. This server sends the request to other coherence servers on server nodes where the instance is replicated in order to keep the different replicas consistent across the distributed system. The algorithm and mechanism used depends on the type of replication. The user may or may not be sent a response at this point, depending on whether the resource type has adopted blocking or non-blocking user response properties.

- 5) When all the coherence servers have received the request and it meets the coherence requirements (event ordering), each server node will execute it and update its local replica of the instance accordingly. An update can be blocking or non-blocking, depending on the coherence model.

The message interfaces for coherence servers and resource handlers are simple queues, but, like TimeWarp, may be re-ordered by timestamp. As all update messages contain a logical timestamp the timing of the application of an update is independent of its queue position.

7.1.5.1 The Replication API

For a resource to be replicated, each replica must understand the requests coming from its local coherence server. Moreover, resource handlers must know how to ask the coherence server to forward a request to all the replicas. This is achieved by the API between the Resource Handler and the Coherence Server (steps 4b, 4c in figure 7.9):

Resource Handler	Coherence Server
send_request (instance, command, user, parameters)	<i>returns request id.</i>
wait_response (request id)	<i>blocks and returns result.</i>
send_and_wait (instance, command, user, parameters)	<i>blocks and returns result.</i>
is_busy (instance)	<i>returns list of pending requests for that instance</i>

The delivery of requests from the coherence server to the Resource Handler (step 5 in Figure 7.9):

Coherence Server	Resource Handler
execute_request (instance, command, user, parameters)	returns success (and state of instance if read)

The API is implemented as a library that is dynamically linked to the resource handler and the coherence server. Once a resource is registered and its coherence model is

specified, instances can be created. When creating instances of a replicated resource, each resource handler can specify where replicas of each instance should be located. The coherence server guarantees that updates will reach all replicas of a resource instance in the order specified by its coherence model.

7.1.5.2 Implementing Event Ordering: Synchronisation Mechanisms

It is anticipated that the number of server nodes making up a single DLE will not be very large. Say, less than a hundred. Accordingly, a system based on logical time and logical clocks is quite feasible. The size of a server cluster is important as vector-clock based synchronisation requires an entry for each member of the cluster in every timestamp attached to an update. All the coherence servers in a DLE will be required to maintain logical clocks using rules such as those specified in Lamport's Clock Algorithm, the Birman-Schiper-Stephenson protocol (BSSP) (Birman et al. 1991) or Peterson and Hutchinson's pSync (Peterson et al. 1989). The choice of clock mechanism for synchronisation depends on the required semantics. The default approach is to use the BSSP (described in chapter 2) as it supports the full range of event orderings listed in Table 7.2 above.

The first request is a conventional http unicast message from the client to the contact server. The inter-replica message types which follow this are:

Request Forward	from contact server to all replicas
Request Ack	from each replica to the group
Request End	from each replica to the group

The BSSP distinguishes between receipt and delivery. Time of delivery is dependent on the vector clock comparisons and the semantics of the replication coherence.

7.1.6 An analytical model of coherence delay

The model provides a basis for calculating the cost of coherence, so starts from the point where a request is received at the server. In terms of the CURL model presented in Chapter 6 the coherence delay would appear as part of server limitation, although the “server” is now multiple servers. That server is referred to as the contact server. In the RRA the contact server is also the server which responds to the client. We ignore the cost of directing a request from the client to a particular server. The following periods are identified within the context of the overall coherence delay:

Server: This is the processing time it takes a server to apply an update once received

Network: This is the round trip time taken for a request to be transmitted from the contact server to all other servers, and for acknowledgements to be received. Transmission is multicast-based.

Coherence : This is the time it takes for all processing to complete and to be reported to the cluster.

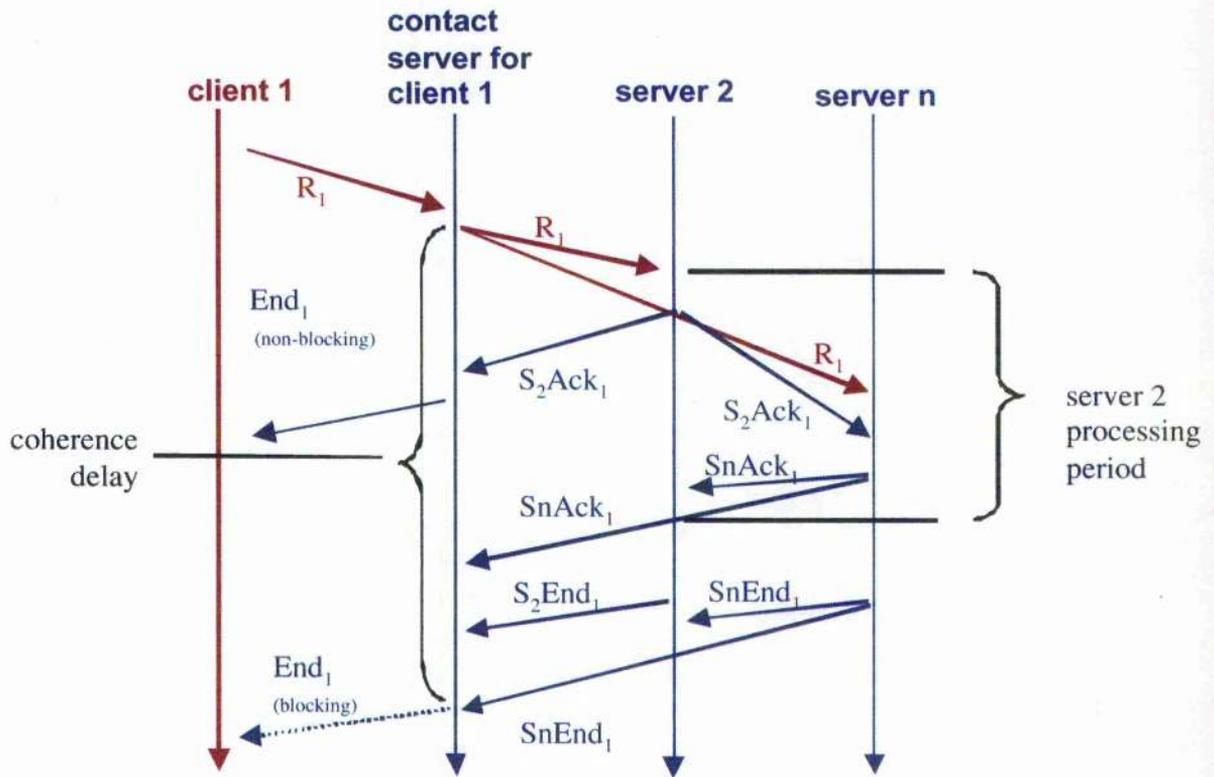


Figure 7.10: Coherence and Server Delay

Figure 7.10 illustrates the situation with one active client and three active replicated servers. The client request R_1 is sent to the contact server. The network delay component is the time it takes for the request to be propagated and acknowledged by all other servers in the cluster. The request is IP multicast, so depending on the level of infrastructure support for multicast, this may take differing amounts of time to reach different servers. The server delay is the processing time it takes a server to apply the update. The coherence delay is the time it takes for all servers to complete their updates. If the resource semantics are non-blocking then the delay experienced by the client is (optimistically) the time it takes for the contact server to process the request and any protocol, network and client delay identified in a CURL. If the request is blocking, then the time will be the coherence delay plus the other CURL components.

7.1.7 Case study – The TAGS Notebook Resource

The Notebook was described in Chapter 5. To briefly recap, it is a lightweight groupware tool that integrates bulletin board and mailing list functions. Updates are either additions (new notes) or deletions. Members of the group to whom a Notebook is allocated may *subscribe* to the Notebook - in which case they receive notification of all notes by e-mail (client pull model). The poster of a note also has the facility to enforce e-mail notification to the whole group, regardless of who has subscribed (client push model).

Concurrent additions should be applied in some serial order but multiple deletions of the same entry must be detected and only one applied. There should never be a need to refuse a user request – two users concurrently deleting the same message should see that message deleted. Let us say that the target response time for a notebook update to take place and return to the user is one second. Local concurrency control requirements are minimal – the notebook data is write-locked while being updated and released on completion. Concurrent additions are serialised on the write-lock. If there are concurrent attempts to delete a note the resource handler must ensure that only one delete is applied. This is achieved in a single server DLE by allocating each note a unique serial number and ignoring attempts to delete a non-existent note. So, the replication requirements for a notebook include the need to avoid duplicate deletions being applied, but also mandate that all copies of a notebook are identical when there are no outstanding update messages. Both these requirements can be met by replacing the assignment of locally unique serial numbers with globally unique numbers. Message processing order and serial number maintenance are then based on logical clocks.

7.1.7.1 Notebook Coherence Model

Let us assume that a totally ordered, FIFO, coherence model is required. An adaptation of Lamport's logical clock algorithm is used to ensure that each note is assigned a globally unique serial number, and that serial numbers are totally ordered. Messages between resource handlers include a unique message identifier consisting of the server identifier and the value of the logical clock in the originating handler. On receipt of a message a handler checks the clock value of the message timestamp and compares it with its local clock. If its local clock is less than the received timestamp it is incremented to be greater than the received timestamp. This ensures that messages arrive after they have been sent in terms of global logical time and provides the basis for lost message detection and a total ordering of notes. The following steps are involved in a notebook update on multiple servers:

- The user submits the update (addition or deletion) to a web server.
- The web server invokes the CGI notebook application, in the form a resource handler.
- The resource handler validates the request with the GRM and passes it to the coherence server.
- The coherence server multicasts an update request to the full group of servers (including itself).
- Each server processes the request and multicasts an acknowledgement.

In blocking mode: handlers wait until they have seen all acknowledgments before writing the new state of the notebook.

In non-blocking mode: the contact handler returns the updated notebook to the client immediately.

If there are concurrent updates at a single node or across the system they are serialised by the coherence server based on globally unique serial numbers.

7.1.7.2 Results

Two clusters were used as testbeds:

- eight 75 MHz Sun Sparc 4 computers running Solaris attached to a 10Mb/s shared ethernet
- sixteen 450 MHz Pentium computers running Linux connected by a fast 100M/s ethernet switch, as part of a commodity cluster

The application was modified so i) serial numbers of notes were globally unique; ii) it could be run in a blocking or non-blocking mode running in blocking mode, which is to say, the coherence delay. The results are shown for two concurrent updates of a Notebook replicated on 2, 4, and 8 nodes on a group of Sparc4s connected by a 10Mb/s shared ethernet, and for up to 16 nodes on a clustered group of Pentium 450MHz PCs connected by a high performance 100MB/s ethernet switch.

<i>Sparc4s, (100 MHz), 10Mb/s shared ethernet; ms</i>				<i>Pentium IIs (450Mhz), 100Mb/s switched ethernet; ms</i>			
Number of Servers	Contact Server	Coherence	Cluster Network	Number of Servers	Contact Server	Coherence	Cluster Network
1	0.71	0.71	0	1	0.49	0.49	0
2	3.05	4.04	2.3	2	1.46	1.37	0.54
4	9.25	11.39	2.4	4	1.68	1.65	0.54
8	22.7	38.18	2.77	8	2.82	3.23	0.67
				16	4.83	7.52	1.01

Tests were carried out for both additions and deletions, but there was no significance difference in the server processing cost. Although the coherence delay for the sixteen server cluster was an order of magnitude above the single server case, it was still acceptably low – in the region of tens of milliseconds. The increase in the delay when 8 Sparc4s are used shows the limitation of this particular (early) cluster technology.

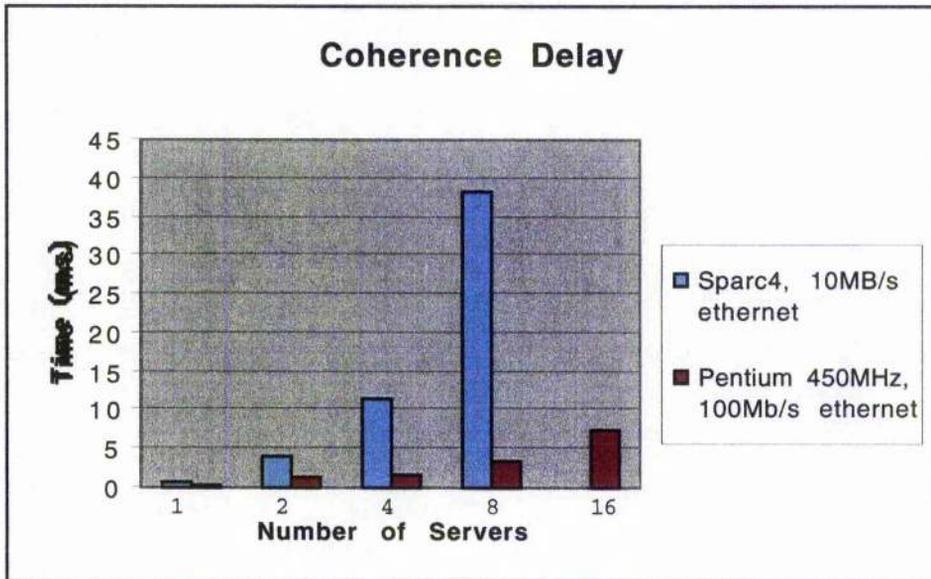


Figure 7.10: Coherence Delay

7.1.7.3 Summary

The experiments with Notebook replication indicate that the RRA is a feasible and promising approach, and that the analytical model and experimental framework provide a suitable basis for the further development of replicable versions of resources. For example, suppose the Notebook were to be redesigned in light of the coherence models available. The experimental version implemented a total ordering, obscuring any concurrency in additions. An alternative design would be to preserve the partial ordering, so that users were aware of concurrency and did not wrongly infer causality where there was none. This could be achieved by viewing the Notebook as a sequence of slots into which notes are posted. In the case of concurrent additions two or more notes would be assigned to the same slot. The users of the notebook would then be aware of the concurrency (lack of order or causality) and respond appropriately. The vector clocks used in the BSSP and the choice of coherence semantics support this flexibility of design.

7.2 Interactive Continuous Media – Adaptability and Integration

Interactive audio and video can be demonstrated, with some effort, on modern networked desktop computers, and this has actually been the case since the early 1990s. What is slightly strange is that it is only slightly easier to do so now than it was ten years ago. This is because interactive network video is a relatively demanding application, not only in terms of network bandwidth, delay and jitter (see Table 7.1), but also in terms of what is required from end-user systems. In recent years desktop systems have reached the point where they can display realtime video at an acceptable quality level in an acceptable time frame, but they still fall short on video capture, although the cost of extra hardware to provide this capability has fallen. It is important to be clear as to what is meant by “acceptable” quality interactive video and audio – the term can be used subjectively. Let us say that a “good” interactive audio/video channel is similar to a traditional two-way television link – the size of the image is at least 420 lines of 640 full colour dots per line, the frame rate is sufficient to display full motion smoothly, both ends of the circuit can transmit sound at the same time without interference or audio blanking, delay is not noticeable, and the video appears to be synchronised with the audio. When the video channel is only one component in a groupware environment an “acceptable” quality may use a smaller display format, perhaps a reduced frame rate that is still adequate for talking heads, and less than full colour pixels.³ In a distributed learning context, we would also often wish to use multi-way conferencing as opposed to the two-way link as described above.

Deployment of resources is a system support task for a DLE, and so the question of how to routinely provide interactive video channels is still very much an open research

issue. Indeed, this type of facility is a major dimension in the Internet2 Digital Video Initiative⁴. The Internet2 Applications FAQ expresses the belief that “the area that will provide the widest benefit and largest aggregate use of the Internet2 network capacity is digital video. Video-based applications cover everything from video conferencing to on-demand content to remote control of microscopes and other instruments”.⁵ However, in the absence of access to the Internet-2 Qbone, or similar bandwidth-rich network, it is more realistic to think in terms of being QoS aware, and being prepared to advise and adapt. Adaptation can take advantage of better network conditions when they are available to boost quality for the end-user, but also drop quality in terms of frame rate, picture size and pixel depth when a network connection cannot smoothly support higher quality.

Integration is also an important issue when using interactive video. Although a significant amount of groupware has been developed as embedded web applications, video conferencing is still very much a bolt-on, and not at all satisfactorily integrated. So, pre-web groupware exemplars such as the Warp shared spreadsheet described in Chapter 4 are particularly difficult to realise on the web, and not surprisingly it has proved difficult to incorporate video conferences into the TAGS framework. Efforts to date have resorted to starting applications such as vic (McCanne & Jacobson 1995) and rat (Perkins et al. 1998) either totally independently or from a browser script, as “helper applications”. This is not adequate as it means that (i) the conference session is not integrated in any useful way with the standard DLE resource allocation mechanism,

³ Web Cam quality, which is quite poor, is rarely acceptable.

⁴ Internet2-DV: <http://www.internet2.edu/dvn/>

⁵ Quote from Ted Hanss, Internet2 Director for Applications Development, at <http://apps.internet2.edu/html/faq.html>

and (ii) a degree of technical expertise beyond that found in typical DLE users is required.

The approach described in the following sections implements adaptation through the use of a Conference Control Architecture (CCA), and integration by using TAGS in conjunction with the Java Media Framework (JMF) (Sun_Microsystems 2001). The CCA collects and analyses information about network paths in order to provide advice that allows a TAGS conference resource to select JMF codecs, embed them in applets, and initialise them with parameters that best meet the needs of individuals joining the conference from different networks with different capabilities.

7.2.1 Adaptation Strategy

The general benefit of an adaptive system is that in the scenario where bandwidth is plentiful, the system can make full use of the available resources, thereby ensuring a high Quality of Service for the participants. On the other hand, when bandwidth is seriously constrained, a lower Quality of Service should be specified that does not waste the constrained resource or act unfairly in relation to competing traffic. At what point in time should an adaptive system react to changes in infrastructure QoS? Research (Bouch & Sasse 2000) suggests that *consistency of quality* is often more important to users than the actual quality of service achieved. For example, if during the lifetime of a video session, the achievable frame rate varies between two bounds, it is arguably better to transmit at the lower bound throughout the session rather than try to dynamically optimise the quality. Given that it is desirable to avoid adaptation to network conditions during the lifetime of a session, it follows that a long period of testing from a low quality starting point for an appropriate operating point is undesirable.

The approach described here is premised on the assumption that most participants will connect from a finite number of access points, and that the QoS available on routes from these access points to other participants will not change dramatically from session to session. Firstly, statistics are collected about the conditions experienced by all the connections to a conference in a common repository. These are analysed with a view to making predictions at the start of subsequent conferences about the conditions that are likely to prevail during their lifetimes. The predictions are used to initialise relevant parameters, so that a Quality of Service appropriate to the network connections can be specified at the start of, and maintained for the duration of, the conference. This can be achieved without expert user intervention, which is an important consideration where the participants in a conference are not expert users. This adaptive approach also allows for improvements in network infrastructure to automatically result in higher quality connection parameters being selected at conference startup. For example, if a participant who normally connects from home upgrades from a modem to ADSL then this will be reflected in improved QoS statistics and higher quality parameters will be used when initialing the codec for that connection in future.

7.2.1.1 A QoS Aware Conference Control Architecture

Figure 7.11 shows a QoS aware Conference Control Architecture (CCA). The main components are

- a Traffic Data Repository (TDR)
- a Conference Controller (CC)
- Participant Agents (PAs).

When a DLE user wishes to schedule a conference, a conference resource is allocated to a group. When a user joins the session the CC is responsible for providing a

suitable applet-based codec for downloading. In practice this download event should not need to be repeated often as the client machine will cache the applet. The applet also contains a participant agent, whose job it is to initialise the codec with appropriate QoS and other conference parameters, and also report network conditions back to the CC. The CC stores traffic reports in the TDR. When a connection is made from a particular IP address the CC analyses information about this type of network connection and suggests parameters to the PA.

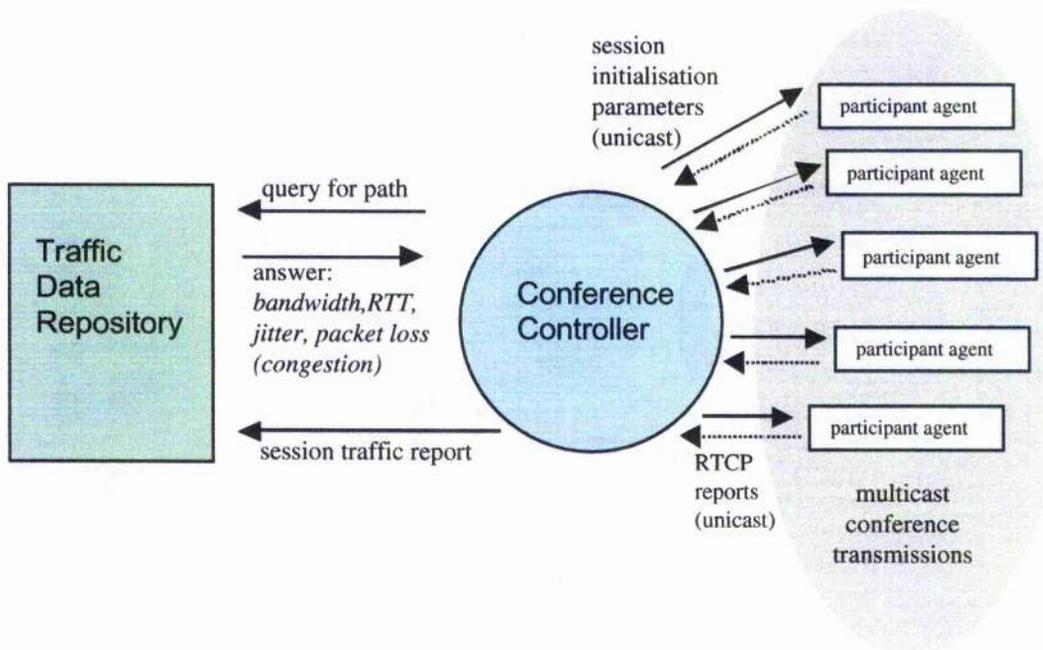


Figure 7.11: Conference Control Architecture

Traffic reports are based on facilities provided by the Real Time Protocol (RTP) and the Real Time Control Protocol (RTCP) (Schulzrinne & al. 1996).

7.2.1.2 RTP and RTCP

The two networking protocol suites realistically capable of supporting multi-way IP-based video conferencing are the Mbone, described in Chapter 2, which implements a fully distributed multicast transmission algorithm, and H.323, which relies on a central Multi-Channel Unit (MCU). In both cases RTP is used on top of UDP. RTP has the

notion of a session as a short-lived entity consisting of members identified by a set of IP multicast and/or unicast addresses and port numbers. An overview of the relationship between a TAGS conference resource and the two protocol stacks of interest is shown in Figure 7.12.

TAGS Video Conference Resource Type	
H.323 Codecs and Multi Channel Units (MCU)	Mbone Tools (true multicast)
Real Time Protocol (RTP) and Real Time Control Protocol (RTCP)	
IP Unicast, UDP	IP Multicast, UDP
Link level technologies: modems, ethernet, PPP	
Physical connections, 9.6Kbps – 100Mbps	

Figure 7.12: Protocols abstracted over by the TAGS Conference Resource Type

It is important to state at the outset that RTP does not provide resource reservation or call admission, and does not guarantee any QoS parameter. As shown in Figure 7.1, RTP is encapsulated in UDP. The main fields of the RTP packet header are shown below.

payload type	sequence number	timestamp	synchronisation source identifier	other fields.....
--------------	-----------------	-----------	-----------------------------------	-------------------

The data transport is augmented by a control protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality. An RTP session provides the association between a set of participants that are communicating using RTP. For each of the participants the session is defined by a network address and two ports. One port is for data and the other for control traffic. In a multimedia conference, each payload type is carried in a separate RTP session with its own control packets. For example, in a conference utilising video, voice, and text chat there would be three RTP sessions.

The session is also the granularity at which information about the state of the network is collected and disseminated to participants. In a session there may be multiple senders and receivers of data. RTCP Sender Reports contain a Synchronisation Source identifier (SSRC), which is unique within the scope of a session. Associated with each SSRC report are a number of traffic statistics. These include:

- timestamps to facilitate the calculation of Round Trip Times (RTTs),
- total counts of the packets sent and lost during a session
- the proportion of packets lost since the last report – this may be indicative of worsening or improving quality
- an estimate of the inter-arrival jitter

These reports are collected by the Participant Agents, processed, and sent to the Conference Controller, which processes them prior to storage in the Traffic Data Repository.

7.2.1.3 Collecting the Path Characteristics

The Participant Agent monitors RTCP reports and generates reports to be utilised by the CC. This involves two functions: determining the end point IP addresses of sources and receivers and extracting meaningful traffic statistics from RTCP Sender and Receiver Reports. These statistics are aggregated into reports for the CC. The CC is then able to take any necessary control actions and generate its own reports to the TDR. RTCP packets contain reports for Synchronisation Sources and Receivers. It is necessary to establish a mapping from the SSRC to unicast IP addresses, because Synchronisation Source identifier may change between sessions. This is achieved by observing the (unicast) IP address of the source. The IP address of the report originator can also be determined from the source IP address on the reports. With these addresses established statistics generated by RTCP can be associated with the

path between the two unicast IP addresses (even if the traffic is multicast). There are at least three traffic statistics of interest: The proportion of packets lost (congestion), the round trip time and the inter-arrival jitter.

- Congestion: Each receivers report contains the fraction of packets lost since the previous report. The receiver calculates this fraction by dividing the number of packets it received by the number of packets expected since the last report.
- Round Trip Time Estimation: It is necessary to determine the RTT to facilitate mapping between fair window sizes and fair rates. Each Senders Report contains a time stamp for the time that the report was generated. This is intended to be used with receivers reports to facilitate the calculation of RTTs.
- Jitter: It is necessary to determine the amount of jitter to properly configure the play out buffer for real-time media. Too large a buffer will increase delay, too small a buffer will result in packets arriving after they should have been played thereby reducing the quality of audio and video playback. A measurement of inter-arrival Jitter is included in receivers reports. This is an estimate of the statistical variance of the RTP data packet inter-arrival time, measured in time stamp units and expressed as an unsigned integer.

7.2.1.4 Traffic Data Repository

The TDR provides a central repository for information about network paths. Consequently, information gathered from one conference can be made available to future conferences. The age of the data held in the repository is also maintained. This enables the CC to make a judgement about the reliability of the data. The TDR provides an interface that allows it to be remotely queried by a CC or other application about the expected traffic characteristics for a path. The TDR also receives reports from the CC about the network conditions experienced during a session.

7.2.1.5 Participant Agent

A PA is located on the client machine connected to the DLE. At start up it receives a description of the path to each participant from the CC and the sessions available in the conference. Based upon the bandwidth information the agent decides which sessions to subscribe to and, based upon the jitter description, it configures the size of the play out buffers for the subscribed sessions. The play out buffers must be set to an appropriate size in order to maintain the real time interactive nature of the system. There are situations where the play out buffer could be increased, for example in a lecturing scenario. During the lifetime of the conference the PA receives RTCP Receivers and Senders reports. These reports contain estimates of packet loss and jitter for the media sources. The agent maps these statistics from synchronization source identifiers to IP numbers. At the end of the conference and periodically throughout the conference traffic statistic reports are generated and sent to the conference controller.

7.2.1.6 Conference Controller

The CC is located in the same cluster as the DLE service hosts. At the initiation of the conference it queries the TDR for predictions of traffic conditions on the paths between conference participants. Based upon this information, it determines which media to use in the conference and the total bandwidth available on each path. From this information it uses a policy to partition the bandwidth between the available media and determines appropriate frame rates, resolutions and sampling rates. These policies can also take the relative importance of the users into consideration. For example, in a tutorial session the tutor's media could be given higher bandwidth, allocations and therefore higher quality, since what the tutor is saying is normally of increased importance. For the duration of the conference, the CC monitors the entry and exit of participants and receives reports from PAs of the network conditions. In the case of a strong mismatch between expected and experienced conditions, it may be necessary to

adjust bandwidth allocation and other parameters. When this happens, updates instructions are sent to each PA. At the end of the conference, the CC generates a report to the TDR of the network conditions and traffic characteristics between each of the participants. This report allows the repository to determine the accuracy of its predictions and to update the data held on the utilised paths.

7.2.2 Integration Strategy

In TAGS, tutors construct a collaborative learning environment by using the Users, Groups and Resources management tool. In order to be part of the TAGS framework, there must be a facility to allocate conference sessions to groups from within TAGS. As the use of H.323 Multi-Channel Units are not freely available to the academic network (at time of writing) we will concentrate on the use of IP multicast. The traditional method of joining a multimedia conferencing session on the Mbone is via the Session Directory Protocol. This typically achieved through use of the SDR tool. SDR displays a list of sessions which are either scheduled to take place or which are currently underway. The entries consist of a multicast IP address, a port number, the multicast scope in the form of a Time To Live (TTL) value, the RTP payload type (audio or video codec specification) and some textual information about the session. From this information any session advertised by SDR may be joined. The mechanism proposed to support the TAGS conference resource type replaces the need to use SDR as an external helper application but does not eliminate the need for the Session Directory Protocol per se. The resource makes conferences easier to setup and join by

- i) removing the complexity of identifying and specifying multicast addresses and port numbers from the user; and
- ii) abstracting over the selection of codec type, frame rate and peak bandwidth.

Perhaps most importantly, the aim is to make a conference just another routine shareable resource type within a DLE, and not a standalone, technically daunting undertaking.

7.2.2.1 Conference Security

There is no inherent security built into Mbone multicast sessions. However, the scoping of the multicast session (the TTL value) does provide a limited form of security in that a participant must be within the scope of the multicast session in order to transmit or receive the RTP session⁶. There is also an element of security through ignorance since in order to receive a multicast session the multicast address, the port number and the time of the session must be known. The issue of multicast security is a concern for the Internet2 Middleware group⁷, and is one of the key deployment issues for multicast IP (Canetti et al. 1999, Chang et al. 1999, Diot et al. 2000).

In H.323 the concept of call admission is enforced, at the central, multi-channel unit. This obviously offers greater potential for control over entry into a conference.

7.2.2.2 Applets and Codecs

The role of the Java Media Framework (JMF) in the TAGS Conference Resource type is to facilitate the production of applets that support various audio and video codecs and transmission protocols. The JMF is a collection of APIs that aim to provide a method for handling time-based multimedia within Java. It does this by allowing the capture, transmission, storage and displaying of various formats of audio and video. A major advantage when compared to other video conferencing technologies is that since it is Java based, it can be incorporated into applets that can then be integrated into a web environment.

⁶ This is analogous to protecting web page access by IP address range.

There are two main distributions for the JMF, a pure Java implementation and a native library version. The pure Java implementation is limited in that it is unable to capture or transmit video/audio. It can usually however render audio/video streams that it receives (dependent on the host computer). In the native version the majority of the processing is performed by platform specific code. This has significant performance advantages over a pure Java implementation and offers greater capabilities for the programmer and user.

A JMF application has three main parts:

- a DataSource, which receives multimedia data from devices such as video capture cards or sounds cards
- a Processor that can be used to change the format, frame rate, image depth or bitrate of the multimedia data - for example, video could be encoded to H.263 at 23 fps 120Kbps
- a DataSink which is used when sending the data to its final destination - this could be to the screen, to a file, or the network.

The JMF supports various network protocols for the transmission of multimedia, including RTP.

⁷ Personal communication from Ken Klingenstein, Director of the Internet2 Middleware Programme.

A/ V	Media Encoding	Cross Platform	Native Library	Bit Rate (sample)
A	G.711 (U-law) 8Khz, 8 bits mono	Rcv/Tx	Rcv/Tx	64 kb/s,
A	G.723 mono	Rcv	Rcv/Tx	6.5 kb/s
A	4-bit mono DVI, 8 Khz	Rcv/Tx	Rcv/Tx	32 kb/s
A	4-bit mono DVI 11.025 Khz	Rcv/Tx	Rcv/Tx	44 kb/s
A	4-bit mono DVI 22.05 Khz	Rcv/Tx	Rcv/Tx	Varies 0.0 -100 kb/s
A	MPEG Layer I 48 Khz @ 16 bits per sample, mono,			64 kb/s
A	MPEG Layer II 22 Khz @ 16 bits per sample, mono,			32 kb/s
A	MPEG Layer III 44 Khz @ 16 bits per sample mono	Rcv/Tx	Rcv/Tx	64 kb/s
V	JPEG (411, 422, 111) *	Rcv	Rcv/Tx	1.5Mb/s @ 320x200 @ 19fps
V	H.261	-	Rcv	-
V	H.263 **	Mode A Only	Rcv/Tx	120kb/s @ 176x144 @ 25fps
V	MPEG-I ***	Tx	Rcv/Tx	Depends on encoded media

* Video dimensions must be multiples of 8 pixels

** Can only be transmitted as one of 128x96, 176x144, 352x288

*** Only from pre-encoded media e.g. mpeg encoded file.

Table 7.3: JMF Supported Multicast RTP payloads

Table 7.3 shows the RTP payload types that are currently supported by the JMF. Additional multiplexers, demultiplexers, filters and codecs can be added to the JMF via means of a plug-in architecture. The bandwidth required by a connection to a conferencing session can be varied by altering the video frame rate, video size, image resolution, image depth, audio resolution, and audio sample rate. It is also possible to change codec or media selection.

7.2.3 Case Study: Small Group Collaboration

A typical requirement of a conference resource type in a DLE is to support realtime interactive communication between small groups using a variety of media. For example, in a tutorial session consisting of a tutor and half a dozen students the

allocated resources would include real-time multi-way video and audio channels, and shared learning resources.

The components of the CCA described earlier have been implemented as a proof-of-concept system. The Participant Agents are JMF-based applets. This makes it possible to allocate conference sessions to groups, in the same way that they are allocated Portfolios and Notebooks. Ideally, in Finesse, teams would be allocated video conferencing sessions to further increase the real world dimension of the learning environment. An early attempt to do so using the Mbone tools co-located on the same machine as the web browser suffered from the same deployment problems as the Warp-based shared spreadsheet, namely that outside of laboratory conditions conferences were unreliable and too difficult to debug.

Let us consider an acceptable quality video connection: an H.263 video codec set to 176 x 144 display size and 24 frames/s requires 120Kb/s; an audio connection using LPC encoding at 8 Khz mono requires 5.6Kb/s. per second. Given these parameters a conference would require a reasonable total of 125.6Kb/s per sender. This is slightly less than the bandwidth available over an ISDN-2 home circuit (128Kb/s) and well within the 300Kb/s promised by ADSL.

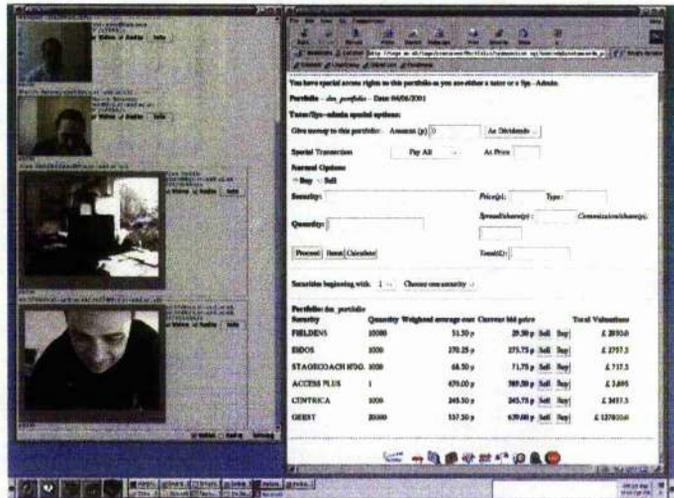


Figure 7.14: A Finesse Session

Fig. 7.14 shows the Finesse environment (the transaction page is showing) expanded to include an allocated conference resource. The desktop machines taking part were equipped with a mixture of relatively inexpensive frame grabbers and cameras. The

QoS for this extended Finesse groupware environment is unusual in that the share data for the shared objects (portfolios) has to be both timely *and* reliable. However, the delay sensitivity for share data can be specified in terms of minutes rather than seconds, or fractions of seconds. The participants in the test sessions were able to chat and use visual cues from each others telepresence to talk about portfolio maintenance decisions.

7.2.4 A Scenario Driven Testbed for Conference Resource Development

No formal evaluation has been carried out on the conference sessions described above, but the progress that has been made suggests that the Conference Controller Architecture is a valid approach to addressing the need for a routinely and easily allocable conference session as a learning resource.

As part of the testing and development of the CCA it was necessary to stage several multiparty conferences on various networks types. This approach has the problem that people are needed to engage in video conferences as guinea pigs. In order to facilitate development and evaluation of the CCA without involving real people a scenario driven network emulation system has been designed and built (Bateman et al. 2002). The testbed emulates a variety of network connection types – ADSL, Modem, 100Mb/s ethernet; it allows for a range of network conditions – packet loss, bandwidth, delay – to be emulated based on measurements made of real traffic; and it provides a way of testing the CCA with a variety of scenarios without the need to involve multiple real participants. Scenarios are specified in terms of the number of participants, how much talking they are likely to do, and how likely they are to be quiet when someone else starts talking.

7.3 Summary

This chapter has described research work that seeks to identify productive approaches to meeting two QoS challenges for a DLE framework, replication and video conferencing.

Resource replication across multiple servers was identified as an attractive policy because it can in principle support incremental scalability, improved performance through load sharing, and higher availability by providing alternative active backup servers. The pragmatic consideration of cost also favours replication in that the commodity cluster approach offers very good price/performance. In order to realise these benefits a replicated resource architecture has been designed. At its core is a coherence layer that allows resource developers to select a coherence model appropriate for a particular resource. The coherence layer has been implemented and tested on two clusters and has performed sufficiently well to merit its further development and use with other resources.

Although video conferencing is seen as an essential part of distributed groupware there is still little sign of its presence in the growing number of web-based groupware environments. When it is present it is not integrated with the web and is typically difficult to install and use. An approach based on the use of TAGS groupware resource allocation system to incorporate RTP-based conferences implemented by applets has been developed and used successfully. The issue of QoS has been addressed by the design of a Conference Controller Architecture. The CCA is QoS aware and employs an adaptation strategy to optimise the conference transmission parameters dependent on the information stored in its Traffic Data Repository. Adaptation is carried out at the beginning of a session, where the lower bound is chosen in order to avoid chopping and changing during a session. The main

contribution of this work has been the demonstration of treating a video conference as just another learning resource, subject to the same allocation mechanisms as any other resource type. This approach contrasts with that taken by the Mbone tools (described in Chapter 3) where all resources in a groupware environment must be IP-multicast based.

TAGS was used as the framework into which these QoS-aware extensions were added. This required some modifications. On developing a new resource type, a developer can select a replica update strategy, appropriate for the type of resource, and on registering a new resource, the developer has the option of stating that it should be replicated. The Users, Groups and Resources management tool has been extended to maintain this information. With respect to a video conference resource type, the CCA is largely transparent to the allocation mechanism, and the creator of a conference simply picks a date and time, and allocates the instance to a group. The CCA uses standard TAGS API calls to find out who is in a group, and maintains its own mappings of users to network end points. It is the likely properties of a network end point that are important, rather than the person connecting from it, so the association can only meaningfully be made at conference startup time.

8 Conclusion

This thesis has reported on investigations concerned with identifying and addressing systems issues associated with the design, development and use of Distributed Learning Environments. This chapter concludes with a review of the original contributions in the main themes – Coherence, QoS, Framework - and gives examples of the type of future work that can build on what has been achieved and learned. It starts with a review of what is meant by a DLE.

8.1 Distributed Learning Environments

DLEs differ significantly from the more traditional forms of computer assisted learning where a student typically works in isolation. They are networked, multi-user environments and as such are well suited to meet the pedagogical goals of group-oriented, anytime/anywhere, student-centered education. The systems support challenges are concerned with realising DLEs on top of the (largely given) public infrastructure (see Fig.8.1).

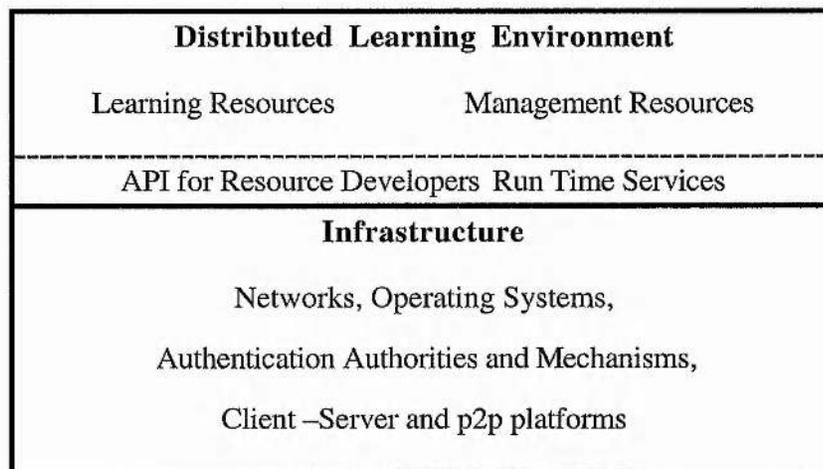


Figure 8.1: Distributed Learning Environments and the Infrastructure on which they execute

Deployment is a major issue, so it is not sufficient or appropriate to create a system which does not take the reality and nature of the infrastructure into account. Accordingly, much of this work has been informed by the direct experience of building,

deploying and maintaining DLEs for a range of disciplines and purposes across several Universities between 1995 and 2002 (Ritchie 2000, 2002).

Three main inter-related systems support themes have been identified: Coherence, QoS, and a Framework. Coherence is necessary for maintaining system integrity in the face of concurrent access to, and replication of, distributed and shared instances of learning resources. Particular types of QoS are required to provide the timeliness of continuous media and interactive responsiveness of server-based learning resources. A framework is essential to provide commonly required features such as authentication, access control, role allocation, resource registration, resource allocation, user-centric portals, and management facilities for tutors, developers, systems administrators and service providers.

8.2 Coherence

Group working and information sharing are key features of DLEs. A potential problem arises when multiple users attempt to modify a shared resource at the same time. Due to the pervasive load-modify-store semantics in computer systems there is always the possibility of unintended outcomes, and inconsistent copies. The task of a coherence mechanism is to ensure that an interactive distributed system remains in a consistent state, and that users of that system can trust the view they have of its state. In a group work setting this must be done without compromising interactive responsiveness for the users of that system. This problem was addressed by the Warp coherence mechanism. A Warp system consists of *atoms* and *objects*. Atoms augment the conventional database transaction properties of serialisability and atomicity with other features useful for groupware applications – automatic conflict resolution, fairness, and liveness. Serialisability is the basis for correctness in the Warp mechanism. The first access to an object is referred to as the atom's *initial touch* of the object. The state of the host process immediately prior to the touch is checkpointed. An atom may proceed optimistically but must own all the objects in its touch set in order to commit. When an atom commits it may cause other atoms holding copies of these objects to *backtrack* if any of these

objects have been updated by the commit. A backtracked atom does not terminate but obtains a new copy of each object it needs. If an atom does not own all the object clones it needs to progress it may be required to *backoff*, in order to allow for re-arbitration, which is performed by the Warp mechanism. Arbitration is based on the unique timestamps associated with atoms, so eventually each atom will become the oldest in the system and win all the ownership tokens it needs to commit.

8.2.1 A Shared Spreadsheet

The utility of the Warp mechanism, and its relevance to interactive real-time groupware, was demonstrated through the construction of a shared multi-user distributed spreadsheet. An *editing session* is the interactive view of an atom provided by the spreadsheet user interface. A user must explicitly start, end or abort an editing session. During an editing session a click on a unit (selection of cells) of the spreadsheet results in a *touch* on the objects representing that unit. All of the objects touched in an atom form the *touch set* of that atom. They are guarded by the atom and are subject to backtrack. The status of a users touch set is reflected by colour coded cells. The coding is user-centric, showing the status of cells relative to the user. The spreadsheet maintains the integrity of a consistent shared state in the face of multiple concurrent readers and writers, a feature that does not appear to be available in any other similar system, commercial or otherwise. Shared spreadsheet sessions were carried out between sites at St Andrews and Dundee to develop a realistic multi-user realtime business game for management and accountancy students – an early example of a DLE component.

8.3 QoS

A DLE consists of multiple interactive resources, all with their own QoS requirements. Figure 8.2 shows the principal distinction that can be made between the two high level types of QoS resource requirement: interactive responsiveness for server-based resources, and timeliness for multi-way video conferences.

		Bandwidth	Delay Sensitivity	Jitter Sensitive?	Loss Tolerant?
Timeliness (continuous media)	Video	high	high	yes	yes
	Audio	low- medium	medium	yes	yes
Responsiveness (server-based resources)		low - medium	low	no	no

Figure 8.2: Characteristics of the two main types of QoS Requirement

Until higher bandwidth communication links become more plentiful affordable multi-way video conferencing will remain a marginal activity and the majority of the resources making up a DLE will be server-based.

8.3.1 Understanding Delay in Server-based Resources

Chapter 6 reported on the modeling, measuring and analysis of QoS for the interactive responsiveness requirement. The work was carried out in two phases, firstly in 1998 when the Finesse DLE was the subject of study, and again in 2000 when TAGS in general was the focus. The approach taken distinguishes itself from other work on QoS by aiming to understand the overall delay as experienced by the end user.

The concept of a Closure over a URL (CURL) was introduced. A CURL covers the entire period from the user-initiated selection of a URL, to the resolution of its associated links within the scope of an HTML page. It may terminate correctly or incorrectly. When a CURL terminates correctly all images, files, applets and other components that are associated with a URL are located or generated, retrieved, displayed and/or activated. The time taken for a CURL to complete can be broken down into four high-level delay components: server, client, network and protocol. These components are inter-related and it is therefore necessary to distinguish between the absolute and the relative amounts of delay they contribute within a specific CURL.

The Server component refers to the delay attributable to the processing of the user request on the server(s) and the execution of the resource(s) triggered by it. The absolute server delay represents a minimum cost and is equivalent to the time that would be taken if the request had arrived from a source internal to the server, and the output

was returned there. The relative server time refers to the extent that other components may be blocking the speed at which the server resources execute. For example, flow control signaling from the client may result in the server processes suffering many wait states and being paged out by the operating system.

The Client component refers to the time taken by the client to initiate a network connection in response to the users action, and the time taken to resolve and render the output returned by the server. A client can be characterised by three main features: hardware configuration, operating system, and browser type and version. The absolute client limitation refers to the time that the client would take to detect and process the users action, and then render all the output, assuming it was already in the browsers input buffers. The relative client time is what is attributable to the client's interaction with the network, protocol and server components. For example, a poor network connection with frequent packet loss may cause the client to delay in asking for more data from the server, and add to the overall delay.

The Network component refers to the limitations caused by the delay experienced by data between two end points. In the simple case where the two end points are directly connected then the minimum can be expressed as a function of the amount of data to be transferred, the available bandwidth on the link, and the propagation delay. In most cases the path between two end points involves traversing multiple hops (and router queues), so the available bandwidth is calculated using the TCP fair share equation. The relative component is the extent to which the available bandwidth causes either the client or server to slow down.

The Protocol component refers to the nature of TCP. There are two features of TCP which mean that HTTP may not receive its fair share of the available bandwidth on a link – the Slow Start algorithm, and the Congestion Avoidance algorithm. Slow Start effectively probes the network, doubling the window size with each successfully acknowledged packet until it reaches a stable state. If there is an implicit congestion notification (lost packet) then the sender's window size is either halved or reduced to one segment. The combined effect of Slow Start and Congestion Avoidance is not

significant for long, large data transfers, such as bulk e-mail and ftp, but means that many typical HTTP transactions, which are for files of less than one kilobyte, never reach a stable state, or obtain their fair share of the available bandwidth.

These four limitations form a model for understanding the sources and causes of delay for users interacting with server-based resources. Measurements were made using a mixture of packet traces and modified versions of resources across a variety of platforms and network connection types. In 1998 the client and the server were clearly the main contributors to the delay. In 2000 clients had clearly improved and the main delay component was the server, although, for modem connections protocol and network limitation were relatively much more significant than in 1998.

8.3.2 Addressing Server Limitations

The analysis of delay has repeatedly pointed to server limitations as being the most significant source of delay for most users. It is possible and desirable to improve server performance by i) overhauling software and ii) purchasing newer faster hardware with more memory. Neither of these approaches are adequate as longer term solutions. This is in part due to the variance in load that can be experienced in a DLE. A good example was provided when the Finesse DLE was used concurrently by over seventy students as part of an induction session at Aberdeen University – in contrast to the normal concurrent load on the DLE which was between one and two users. In Chapter 7 a solution to this problem in the form of a Replicated Resource Architecture (RRA) was presented and prototyped. The RRA harnesses multiple servers, possibly in the form of a *commodity cluster*. This approach has four potential advantages: i) it provides the basis for better responsiveness through load sharing; ii) it provides for incremental scalability; iii) it provides a suitable basis for high availability; iv) it is affordable, which is important for education. The work carried out earlier, reported in Chapter 4, on coherence, was used to inform replication strategies. In contrast to the earlier work, which had attempted to use the Warp mechanism as a single means of meeting all coherence requirements, the RRA Coherence Layer allows for different types of

coherence, in order to optimise performance. A case study using the TAGS NoteBook resource was carried out, and showed the RRA approach to be feasible and suitable for further development, through application to other resource types.

8.3.3 Addressing Timeliness

Server limitation can be addressed in a DLE when the servers and software belong to the DLE service provider. This is not a feasible approach for addressing the QoS requirements for interactive realtime continuous media - the effective bandwidth, delay and jitter depend on the prevailing conditions across the range of networks constituting a particular set of Internet paths used in a video conference. These networks are typically outwith the control or influence of the service provider. Although various Internet infrastructure features have been proposed to support QoS based on call admission and resource reservation, these have not been implemented outside of special projects run within controlled single networks. For example, the Scottish Universities benefited from an ATM-based video conferencing network between 1997-2002. This provided near broadcast quality multi-way audio and video for nodes attached directly to the network, but the quality of external ISDN or Mbone links was extremely poor in contrast.

So, assuming realistically that guaranteed QoS is not available for students attending a video session from a variety of locations and network connections, an *adaptive* approach was adopted. Adaptation means being able to take advantage of better network conditions when they are available to boost quality for the end-user, or dropping quality in terms of frame rate, picture size and pixel depth when a network connection cannot smoothly support higher quality.

The adaptive approach has been reflected in the design of a *Conference Controller Architecture (CCA)*. The CCA consists of a Traffic Data Repository, a Conference Controller and Participant Agents. The agents lodge RTP-based traffic reports with the controller, which distills them and stores them in the repository. The controller uses information based on past path characteristics to select codec type and initialisation

parameters for a new conference. Adaptation is carried out at the beginning of a session, where the lower bound is chosen in order to avoid chopping and changing during a session, a feature that is known to be unpopular with users. The CCA has facilitated the development of a Video Conference resource type that can be allocated in the same way as other DLE resources.

8.3.4 Integration of QoS-Oriented Support Features

The development version of TAGS was used as a base onto which the RRA and the CCA were added. On developing a new resource type, a developer can select a replica update strategy, appropriate for the type of resource, and on registering a new resource, the developer has the option of stating that it should be replicated. The Users, Groups and Resources management tool was extended to maintain this information. With respect to the video conference resource type, the CCA is largely transparent to the allocation mechanism, and the creator of a conference simply picks a date and time, and allocates the instance to a group. The CCA uses standard TAGS API calls to find out who is in a group, but dynamically maintains its own mappings of users to network end points.

8.4 The Framework

The high-level goals of a framework are to facilitate the development, deployment and management of distributed learning environments. The TAGS framework described in Chapter 5 achieved this for a range of disciplines across several institutions. The systems requirements addressed by the production version of TAGS include:

- Authentication
- Access Control
- Role Allocation
- Resource Allocation
- User-centric Views

- Group Communication
- Group Membership

At the core of the TAGS framework is a simple model that supports these features, and provides suitable abstractions for their use by non-specialists. The model is based around the three components of *users*, *groups* and *resources*. Users and groups are unique by name; resources are unique by name and type. A “group” is an overloaded abstraction that provides an arbitrary mapping between users and resources (see Figure 8.4). As such it forms the basis of i) resource allocation; ii) privileges and access; iii) role allocation; iv) building the user’s personal portal; v) dissemination channels for communication resources; vi) membership of an administrative grouping.

There is no restriction on the nature of a resource. It can be a simple timetable, an interactive multi-user spreadsheet or a live data feed. Access rights can be specified when a resource is allocated to a group. The range of resource types developed to date can be categorised as follows:

- Management Resource Types for Creating and Maintaining DLEs e.g. Users, Groups and Resources Tool
- Generic Resource Types for Managing Online Learning e.g. The Assignment Tracking Tool
- Generic Resource Types for Collaborative Working e.g. the Notebook
- Subject-specific Resource types e.g. the Finesse Portfolio Management Facility

In practical terms, tutors construct and maintain a DLE by using the Users, Groups and Resources management tool. When a resource is modified the changes are reflected to all members of all groups it has been allocated to. If it is deleted it is removed from all the groups it was associated with. When a group is deleted the mappings it has formed between individuals and resources are removed. If a group’s membership is changed then only the new membership will have access to resources allocated to that group. When a user is removed from a group they lose access to that group’s resources and if a user is deleted they are removed from all groups in the system. Various commonsense

rules apply as to who can manipulate which users, groups and resources. If a user is a member of multiple groups which have different access privileges to the same resource then they are credited with the highest level from their set of privileges. (This has proved satisfactory to date in TAGS). A simple example scenario is shown in Figure 8.3.

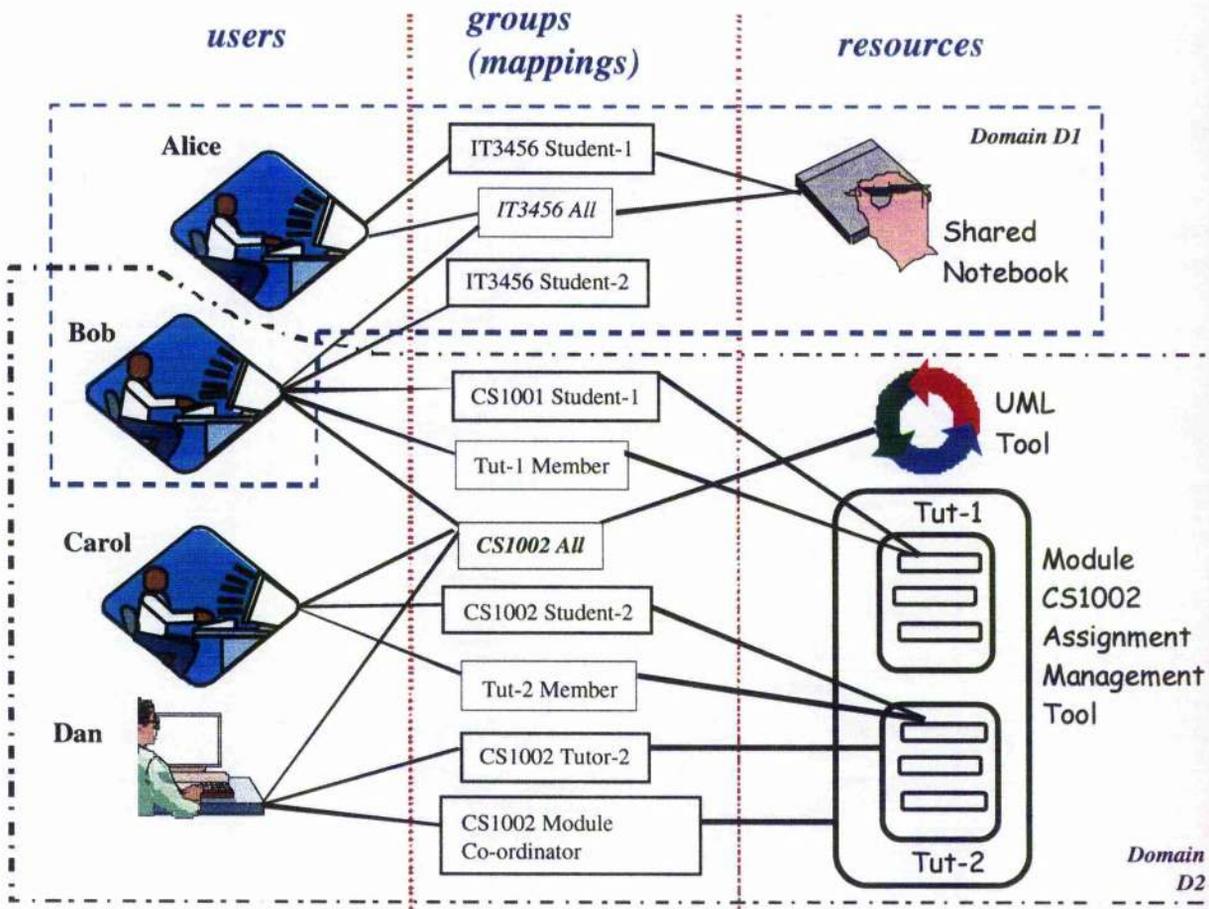


Figure 8.3: Users, Groups, Resources and Domains

- Alice is a member of group module IT3456. All students who belong to this module have a shared Notebook for group discussion. Alice has subscribed to the Notebook, so her identity is automatically used when she posts a note.
- Bob is a member of IT3456 and also CS1002. Bob has not subscribed to the IT3456 Notebook. All members of CS1002 have been allocated a UML tool. This is a common resource, but not a collaborative one.

- Bob is also a member of CS1002 Tutorial Group 1. He has access to his part of a module assignment management tool. It allows him to see his assignments with their due dates and acknowledge by dates, to submit work electronically and to receive qualitative and quantitative feedback online. Feedback may include how he is doing relative to the rest of the class.
- Carol is a member of the CS1002 module and also of CS1002 Tutorial Group 1. She has access to the UML tool, and to her own part of the CS1002 module assignment management tool.
- Dan is a member of academic staff. He is a tutor and co-ordinator for CS1002. As a tutor he has a view of the assignments and marks of just his own tutees, but as co-ordinator he can also select a wider view of the whole class. As a member of the main module group he has been allocated the UML teaching tool.

All of a users allocated resources appear on their home page, which is dynamically generated from their group memberships.

8.4.1 Domains

The role of *system administrator* is supported in TAGS. It is somewhat akin to that of *root* in Unix systems. Course co-ordinators were originally created belonging to the *sys-admin* group, thereby enabling them to create tutors as well as students. As use of the TAGS-based DLEs grew it became clear that a distinction was needed between system-wide privileges and DLE-wide privileges. Domains were added to the framework in order to provide the administrative protection that is required without unnecessarily constraining the freedom of communication that collaborative group work requires. Collaboration depends on group membership, and is therefore treated as a tighter binding than domain membership. Consequently, assigning members of different domains to the same group will facilitate inter-domain communication. A group may appear in multiple domains while the users remain in separate domains. For example, in Figure 8.4 two domains might exist, Languages (labelled D1) and Computer

Science (D2). Student Bob exists in both domains, whereas module co-ordinator Dan only exists in one.

The utility of the Domain abstraction is that it can be used to bound privileges by any required theme. The Finesse Domain contains tutors and students from different institutions. This facilitates inter-institutional co-operation and comparison of investment strategies and performance results. In the case of Economics and Language Teaching domains are bounded by institutional units. In Computer Science domains are typically bounded by module, that is, there is no Computer Science domain per se. To date, domains have met all of the bounded administrative privilege requirements requested from a range of academic staff across Departments, disciplines and institutions.

8.5 Future Work

The general strategy followed when addressing DLE issues has struck a balance between placing too much functionality in the framework (thereby constraining resource developers) and providing enough appropriate interfaces and facilities for students, tutors, developers and service providers. The framework approach has been successful, and it can now be revised and extended. Topics of interest include automated group maintenance and QoS-oriented services.

8.5.1 Automatic Group Maintenance

The use of the *group* is central to the formation and maintenance of a DLE in the TAGS framework (see Figure 8.3). At present groups are updated through the use of various management tools. In the situation where the user population and their roles are changing frequently this requires repeated manual intervention and it would obviously be useful to automatically maintain groups where possible. Consider the case of a DLE organised around a curriculum of taught modules as an example. The two main features are dynamic links and rule-based relations.

8.5.1.1 Dynamic Links

Educational institutions typically maintain staff and student records on separate centralised database systems. Student records tend to be highly detailed whereas staff descriptions are less specific. So, while a student record system can be queried for lists of students taking a module, a staff system cannot be queried to find out which staff are associated with a module, and what their roles are with respect to that module. However, academic departments typically maintain local information on both the students taking their modules, and the staff that are associated with each module. This is a simple operational necessity as it will often be the case during volatile periods at the beginning of a module that the local information is not consistent with central information. When creating a group of students within a DLE it should be possible to specify that group as a dynamic link to either a local or central database listing.

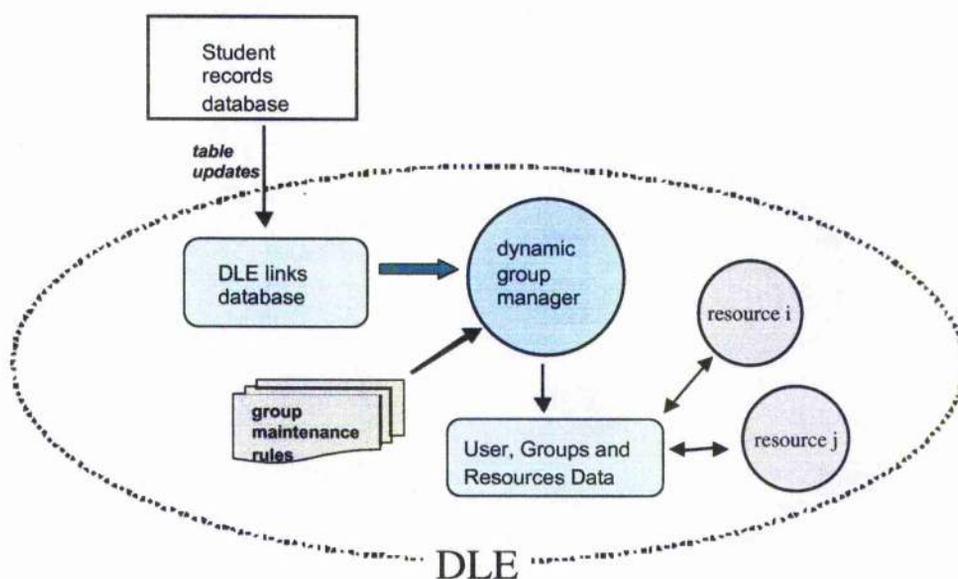


Figure 8.4: Automatic Group Maintenance

The effect that is wanted is for external database updates to be noticed and acted upon within the DLE, in some programmable way. There are various ways that this can be accomplished. For example, external changes can be noticed by establishing an ODBC link between the external database, and one internal to the DLE. A dynamic group management resource can then update the DLE group that is mapped to the external table. Figure 8.4 gives an overview. If a user is added to that group then they will

automatically be allocated the resources for that module, if they are removed then they will no longer find these resources in their (view of the) DLE.

8.5.1.2 Maintaining Relations

A module will typically consist of multiple groups, reflecting different views, roles, privileges and resource allocations (see Fig. 8.3). The dynamic group manager (DGM) resource will therefore have to do more than add and remove users from a single group. We address this by defining a set of groups and relationships between them.

A teaching module M consists of the following groups:

M_All	All users (staff and students) associated with a module.
-------	--

We assume that M_All has been allocated a set of resources, such as an assignment management tool, a shared workspace tool and subject-specific learning resources, and that a resource will export an interface dependent on the users role and identity (see Fig 8.4). We also assume, (as is the current case with TAGS) that a resource such as the assignment tracking tool will use runtime calls to dynamically obtain group membership.

Other groups are:

M_Students	All students associated with a module.
M_Students_Left	Students who have left the module.
M_Staff	All staff associated with a module.
M_Lecturers	All lecturers associated with a module
M_Tutor	All tutors associated with a module
M_Co-ordinator	All co-ordinators associated with a module.
M_Tut_Group_n	Member of tutorial group n.

We assume that M_Students is defined as an automatically managed group linked to an external database via the DGM. The following rules are applied by the DGM:

- Any student who leaves the module is moved from M_Students to M_Students_Left. These groups are mutually exclusive.
- Any student who joins the module is added to M_Students.
- All tutorial groups, M_Tut_Group_{1..n} are subsets of M_Students.
- Each student in M_Students is a member of exactly one tutorial group.

- A new tutorial group M_Tut_Group_n will be created and students balanced across groups according to policy for tutorial group formation (size, gender mix, etc).

In addition to applying rules, the dynamic group manager can also issue warnings if a manual alteration breaks a rule.

8.5.2 Automating QoS-Oriented Runtime Services

QoS in the wider sense includes interactive responsiveness, timeliness and availability. The work that has been carried out on understanding the problems in providing QoS in DLEs has mostly been of a post-mortem nature, and requires considerable manual intervention and analysis. Ideally we would like to constantly monitor the QoS being provided by a DLE, with a view to signaling resources which wish to adapt, or triggering warnings for service providers. The investigations into the sources of delay in Chapter 6 provided insight into the use of measurement-based monitoring for interactive responsiveness, and a variation on this technique was used in the Conference Control Architecture presented in Chapter 7. This approach showed that useful information can be obtained from observing TCP client and server window sizes, and RTP reports.

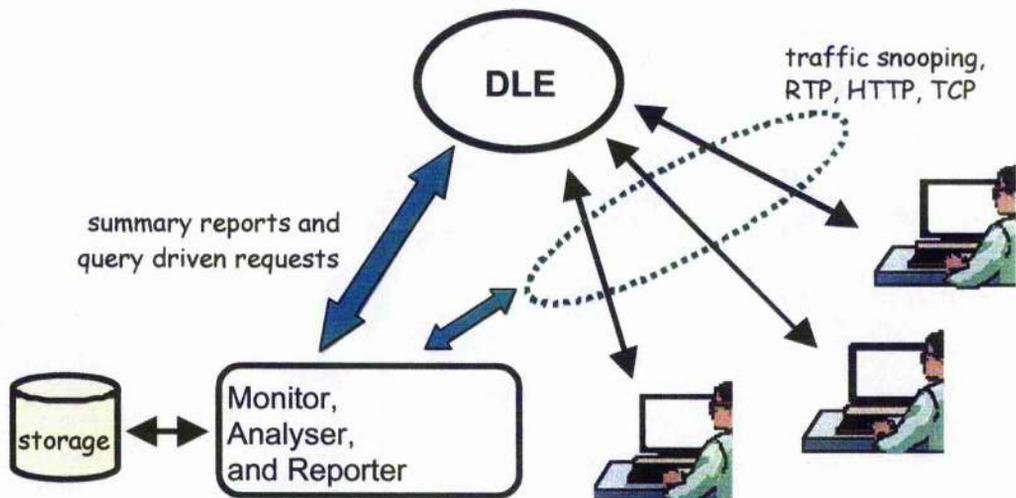


Figure 8.5: A QoS Monitoring and Reporting System

Figure 8.5 shows a model of how a dedicated QoS monitor could operate. A fundamental requirement is to have a traffic snooping mechanism. This can be implemented either on a separate computer in the same collision domain as the DLE

server(s), or by using an OS facility for packet monitoring on the same server as (a single server) DLE.

The job of the monitor is to:

- capture, filter and analyse the HTTP/TCP packets associated with a connection to a server-based resource
- act as repository for RTP traffic reports
- provide a failure suspector service
- provide a QOS-feedback service to any resource which requests it

The TCP analysis process would produce a brief summary report for an HTTP session giving the estimated server, client network and protocol contributions to the overall delay. A DLE resource would be responsible for configuring the monitor and could also issue specific queries on behalf of the service provider or manager. The configuration would include thresholds, indicating the level of interactive delay that would trigger a report. Reports could be notified directly, by e-mail for example, to the service provider.

8.6 Concluding Remarks

The work reported in this thesis has been written up against the background of constantly shifting economic, technical and social perspectives on what a “successful” set of Internet and Web services should look like in an educational context. Several of the issues identified and addressed in the thesis are now beginning to emerge within the context of a common Internet infrastructure. These include role-based portal technologies, interactive IP-based video and audio, information sharing and group working. Perhaps the most distinguishing system feature of DLEs as opposed to other web-based portals and services is the high degree of interactivity that they require. The bulk of current Internet and Web based services provide only limited interactivity such as simple click and follow, download, media streaming, site search and HTML form upload. While there is growing awareness of the importance of response times even in these generic contexts, the prospect of routine web services being QoS aware in order to

provide a high degree of interactivity is still in the future. The work described in this thesis can be used to inform the successful design and deployment of the next generation of distributed learning environments.

References

Advanced_Distributed_Learning_Inc. 2001. SCORM: The Sharable Content Object Reference Model, <http://www.adlnet.org/>.

Agrawal, R. & Dewitt, D. J. 1985. Integrated Concurrency Control and Recovery Mechanisms: Design and Performance Evaluation. *ACM Transactions on Database Systems* 10(4), 529-564.

Allison, C. 1998. Virtual Networks and QoS for Cluster Computing. In: 6th Euromicro Workshop on Parallel and Distributed Processing (PDP '98). IEEE, Madrid, Spain, 127-132.

Allison, C., Bain, A. L., McKeachan, D. & Michaelson, R. 2000a. Using TAGS for Distributed IT Project Management. In: "The Move from Teaching to Learning", 1st Annual Conference of the LTSN Centre for ICS (edited by Scott, T. & Pooley, R.). LTSN ICS, Edinb

Allison, C., Bramley, M. & Serrano, J. 1998. The World Wide Wait: Where Does the Time Go? In: *Engineering Systems and Software for the Next Decade*. Euromicro Annual Conference. IEEE Press, Vasteras, Sweden, 932-940.

Allison, C., Bramley, M., Michaelson, R. & Serrano, J. 1999. An Integrated Framework for Distributed Learning Environments. In: "Advances in Concurrent Engineering", 6th ISPE International Conference on Concurrent Engineering (edited by Chawdry, P. K., Gh

Allison, C., Harrington, P., Huang, F. & Livesey, M. J. 1996. Scalable Services for Resource Management in Distributed and Networked Environments. In: 3rd Workshop on Services in Distributed and Networked Environments (SDNE '96). IEEE Press, Macau, 98-105

Allison, C., McKeachan, D., Lawson, H. & Michaelson, R. 2000b. The TAGS Framework for Web-Based Learning Environments. In: *Web Based Learning Environments* (edited by Ribeiro, L. M.). University of Porto, FEUP Editions, Portugal, 10-14.

Allison, C., McKeachan, D., Ruddle, A. & Michaelson, R. 2001a. A Group Based System for Group Based Learning. In: *European Perspectives on Computer-supported Collaborative Learning, the proceedings of Euro CSCL 2001* (edited by Dillenbourg, P., Eurelings,

Allison, C., Ruddle, A., McKeachan, D. & Lindsay, P. 2001. Determining the Sources of Delay in a Distributed Learning Environment. In: *PGNet 2001*. John Moores University, Liverpool, 71-77.

References

- Allison, C., Ruddle, A., McKechn, D. & Michaelson, R. 2001b. The Architecture of a Framework for Building Distributed Learning Environments. In: *Advanced Learning Technologies*. IEEE Press, Wisconsin, 29-35.
- Appelt, W. & Mambrey, P. 1999. Experiences with the BSCW Shared Workspace System as the Backbone of a Virtual Learning Environment for Students. In: *World Conference on Educational Multimedia, Hypermedia and Telecommunications ED-MEDIA 99, SEATTLE*.
- Arlitt, M. F. & Jin, T. 2000. A Workload Characterization Study of the 1998 World Cup Web Site. *IEEE Network* 14(3), 30-37.
- Arlitt, M. F. & Williamson, C. L. 1997. Internet Web Servers: Workload characterization and implications. *IEEE/ACM Transactions on Networking* 5(5), 631-644.
- Babaoglu, O., Alvisi, L., Amoroso, S., Davoli, R. & Giachini, L. A. 1992. Paralex: An Environment for Parallel Programming in Distributed Systems. In: *6th ACM International Conference on Supercomputing*. IEEE CS Press, 178-187.
- Bal, H. 1990. *Programming Distributed Systems*. Silicon Press.
- Bateman, M., Allison, C. & Ruddle, A. 2002. Scenario Driven Network Emulation. In: *PGNet 2002, the 3rd Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*" (edited by Merabti, M.). JMU, Liverpool, 46-51.
- Bayerdorffer, B. 1995. Broadcast Time Warp. In: *HICCS'28* (edited by El-Rewini, H. & Shriver, B. D.) II. IEEE Press, 602-611.
- Bentley, R., Appelt, W., Busbach, E., Hinrichs, D., Kerr, D., Sikkell, K., Trevor, J. & Woetzel, G. 1997. Basic Support for Cooperative Work on the World Wide Web. *International Journal of Human Computer Studies* 46(6), 827-846.
- Berners Lee, T. 2000. *Weaving the Web*. Texere Publishing.
- Berners-Lee, T., Fielding, R. & Frystyk, H. 1996. RFC 1945 Hypertext Transfer Protocol -- HTTP/1.0. IETF.
- Bhatti, N. & Friedrich, R. 1999. Web Server Support for Tiered Services. *IEEE Network* 13(5), 64-71.
- Birman, K. 1991. *The Process group approach to reliable distributed computing*. Cornell University Department of Computer Science.

References

- Birman, K., Schiper, A. & Stephenson, P. 1991. Lightweight causal and atomic group broadcast. *ACM TOCS* 9(3), 272-314.
- Birman, K. P. & van Renesse, R. 1993. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society Press.
- Blackboard. 2000. <http://www.blackboard.com>
- Blake, S. & al, e. 1998. An Architecture for Differentiated Services, RFC2475.
- Boggs, D. R. 1983. Internet Broadcasting. Xerox PARC, Palo Alto.
- Bouch, A. & Sasse, M. A. 2000. The Case for Predictable Network Service. In: MMCN'2000. ACM, San Jose, 188-195.
- Braden, B., Zhang, D., Estrin, S. & Jamin, S. 1996. Resource ReSerVation Protocol (RSVP) -Version 1 Functional Specification, IETF Working Document.
- Bradner, S. & Mankin, A. 1995. The Recommendation for the IP Next Generation Protocol, RFC-1752.
- Bradstreet and Dunn. Datastream.
- Britain, S. & Liber, O. 1999. A framework for pedagogical evaluation of Virtual Learning Environments. JTAP.
- Brown, A. L. 1989. Persistent Object Stores. Unpublished Ph.D. thesis, University of St Andrews.
- Budhiraja, N., Marzullo, K., Schneider, F. B. & Toueg, S. 1993. The Primary-Backup Approach. In: *Distributed Systems* (edited by Mullender, S.). ACM Press, 199-216.
- Burgess, P., Livesey, M. J. & C., A. 1994. BED: A Multithreaded Kernel for Embedded Systems. In: WRTP'94, (Proc. 19th IFAC/IFIP Workshop on Real Time Programming). IFIP, Lake Constance.
- Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M. & Pinkas, B. 1999. Multicast security: A taxonomy and some efficient constructions. In: *IEEE INFOCOM 1999 - The Conference on Computer Communications*, 708-716.
- Cardellini, V., Colajanni, W. & Yu, P. S. 1999. Dynamic Load Balancing on Web-server Systems. *IEEE Internet Computing* 3(3), 28-39.
- Chang, I., Engel, R., Kandlur, D., Pendarakis, D. & Saha, D. 1999. Key management for secure internet multicast using boolean function minimization

References

- techniques. In: IEEE INFOCOM 1999 - The Conference on Computer Communications. IEEE, 689-698.
- Chang, J. & Maxemchuk, J. N. 1984. Reliable Broadcast Protocols. *ACM TOCS* 2(3), 251-273.
- Chiu, D. M. & Jain, R. 1989. Analysis of increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems* 17, 1-17.
- Comer, D. 1999. *Computer Networks and Internets*. Prentice-Hall.
- Coulson, G. & de MEER, J. B. 1997. Guest Editors Introduction. *Distributed Systems Engineering Journal* 4(1), 1-3.
- Dearing, R. 1997. Higher Education in the Learning Society" - The Report of the National Committee of Inquiry into Higher Education. HMSO.
- Deering, S. 1989. Host extensions for IP multicasting.
- Deering, S. 1996. An Architecture for Wide-Area Multicast Routing. *IEEE/ACM Transactions on Networking* 4(2).
- Dennis, J. B. & Van Horn, E. C. 1966. Programming Semantics for Multiprogrammed Computations. *CACM* 9(3), 143-155.
- Dewan, P. & Choudhary, R. 1995. Coupling the User Interfaces of a Multiuser Program. *ACM Transactions on Computer Human Interaction* 2(1), 1-39.
- Dijkstra, E. W. 1968. Cooperating Sequential Processes. In: *Programming Languages* (edited by Genuys, F.). Academic Press, New York, 43-112.
- Diot, C., Levine, B. N., Lyles, B., Kassem, H. & Balensiefen, D. 2000. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network* 14(1), 78-88.
- Ellis, C. A. & Gibbs, S. J. 1989. Concurrency Control in Groupware Systems. In: *Proc. of the ACM SIGMOD '89 Conference on the Management of Data*. ACM, Somewhere, 399-407.
- Ellis, C. A., Gibbs, S. J. & L., R. G. 1991. Groupware -- Some Issues and Experiences. *CACM* 34(1), 39-58.
- Ellis, C. A., Gibbs, S. J. & Rein, G. 1988. *Design and Use of a Group Editor*. MCC Software Technology Program, Austin.

References

- Engelbart, D. C. 1990. Knowledge-domain interoperability and an open hyperdocument system. In: ACM Conference on Computer-Supported Cooperative Work. ACM, 143-156.
- Fidge, C. J. 1988. Timestamps in message passing systems that preserve the partial ordering. In: Proc. of the 11th Australian Computer Science Conference, Brisbane, 56-66.
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H. & Berners-Lee, T. 1997. RFC 2068: Hypertext Transfer Protocol -- HTTP/1.1.
- Finkel, R. A., Zaslavsky, A., Monostori, K. & Schmidt, H. 2002. Signature Extraction for Overlap Detection in Documents. In: Twenty-Fifth Australasian Computer Science Conference (ACSC2002). ACM, Melbourne, 59 - 64.
- Finlay, R. M. 1999. Tele-Learning: The "Killer App"? IEEE Communications 37(3), 80-118.
- Fischer, M. J., Lynch, N. A. & Paterson, M. S. 1985. Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM 32(2), 374-382.
- Floyd, S. & Henderson, T. 1999. RFC 2582. The New Reno modifications to TCP's Fast Recovery algorithm.
- Floyd, S. & Jacobson, V. 1993. Random Early Detection for Congestion Avoidance. IEEE/ACM Transactions on Networking 1(3), 397-413.
- Freier, A. O., Karlton, P. & Kocher, P. C. 1996. The SSL Protocol, Version 3. IETF.
- Goralski, W. J. 1995. Introduction to ATM Networking. McGraw-Hill.
- Greenberg, S. & Marwood, D. 1994. Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface. In: Proc. of the Fourth ACM Conference on Computer Supported Cooperative Work, 207 - 217.
- Grief, I., Seliger, R. & Weihl, W. 1986. Atomic Data Abstractions in a Distributed Collaborative Editing System. In: Proc. of the 13 Annual Symposium on Principles of Programming Languages. ACM, 160-172.
- Grudin, J. 1997. Groupware and Social Dynamics: Eight Challenges For Developers. CACM 37(1), 92-95.
- Hadzilacos, V. & Toueg, S. 1993. Fault Tolerant Broadcasts and Related Problems. In: Distributed Systems (edited by Mullender, S.). ACM Press/Addison-Wesley, 97-145.

References

- Hales, M. 1994. Where are Designers? Styles of Design Practice, Objects of Design and Views of Users in CSCW. In: Design Issues in CSCW (edited by Rosenberg, D. & Hutchinson, C.). Springer-Verlag, 151-157.
- Handel, R., Huber, N. M. & Schroder, S. 1994. ATM Networks. Addison Wesley.
- Handley, M. & Crowcroft, J. 1997. NTE: A scalable shared text editor for the MBone. In: SIGCOMM '97. ACM, France.
- Handley, M. & Jacobson, V. 1996. "SDP: Session Directory Protocol (draft 2.1)". IETF.
- Helliar, C. V., Michaelson, R., Power, D. M. & Sinclair, C. D. 2000. Using a Portfolio Management Game (FINESSE) to Teach Finance. Accounting Education 9(1), 37-51.
- Highfield, J. & Hasler, K. 1999. WBD: <http://www-mice.cs.ucl.ac.uk/multimedia/software/wbd/>.
- Holtz, B. 1992. CoEd: a distributed shared text editor.
- IEEE. 1995. IEEE 802.3u-1995: Supplement to Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications: Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment, and Repeater for 100 M
- IMS Global Learning Consortium Inc. 2000. Digital Repositories Interoperability v1.0 Public Draft.
- IP2. 1999. The Internet 2 Project <http://www.internet2.edu>.
- ISO/IEC. 1996. IS 10476-2. ISO.
- Iyengar, A., Challenger, J., Rias, D. & Dantzig, P. 2000. High Performance Web Site Design Techniques. IEEE Internet Computing 4(2), 17-26.
- Jacobson, V. & Braden., R. T. 1990. Congestion control and avoidance. ACM Computer Communications Review 18(4), 314-329.
- Jacobson, V., Liu, C., McCanne, S., Zhang, L. & Floyd, S. 1995. A Reliable Multicast Framework for Lightweight Sessions and Application-Level Framing. In: ACM SIGComm 95, 342-356.
- Jefferson, D. R. 1985. Virtual Time. ACM TOPLAS 7(3), 404-425.

References

- Kaashoek, M. F. & Tanenbaum, A. S. 1991. Group Communication in the Amoeba Distributed Operating System. In: Proc. Eleventh International Conference on Distributed Computer Systems. IEEE Computer Society, 222-230.
- Klöckner, K. 2000. BSCW - Educational Servers and Services on the WWW. In: International C4-ICDE Conf. on Distance Education and Open Learning, Adelaide.
- Kung, H. T. & Robinson, J. T. 1981. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems* 6(2), 213 - 226.
- Kurose, J. F. & Ross, K. W. 2001. *Computer Networking*. Addison Wesley.
- Lamport, L. 1974. A New Solution of Dijkstra's Concurrent Programming Problem. *CACM* 17(8), 453-455.
- Lamport, L. 1978. Time, clocks, and the ordering of events in a distributed system. *CACM* 21(7), 558-565.
- Lampson, B. 1974. Protection. *ACM Operating Systems Review* 8(1), 18-24.
- Lampson, B. 1998. Computer Security in the Real World. In: *UVC Distinguished Lectures*. UVC.
- Liao, T. 1999. Light-weight Reliable Multicast Protocol Specification. INRIA, 1-45.
- Lin, J.-C. & Paul, S. 1996. A Reliable Multicast Transport Protocol. In: *IEEE Infocom*. IEEE Press, 1414-1424.
- Lindsay, P., Ruddle, A. & Allison, C. 2002. Visualisation of TCP Behaviour. In: *PGNet 2002, the 3rd Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*" (edited by Merabti, M.). JMU, Liverpool, 231-236.
- Liskov, B. 1988. Distributed programming in Argus. *CACM* 32(3), 300-312.
- Livesey, M. J. & Allison, C. 1992. A General Purpose Time Warp Toolkit. Dept. of Computer Science, University of St Andrews, 34.
- Long, P. D. 2002. OpenCourseWare: Simple Idea, Profound Implications. In: *Syllabus* 15.
- M. Mandviwala, L. O. 1994. What Do Groups Need? A Proposed Set of Generic Groupware Requirements. *ACM Transactions on Computer-Human Interaction* 1(3), 245-266.
- Mahdavi, J. & Floyd, S. 1997. TCP-Friendly unicast rate-based flow control. Note sent to end2end-interest mailing list.

References

- Mathis, M. & Floyd, S. 1996. RFC 2018. TCP Selective Acknowledgement Options.
- Mathis, M., Semke, J. & Mahdavi., J. 1997. The macroscopic behaviour of the TCP Congestion Avoidance algorithm. *Computer Communication Review*(27).
- Mattern, F. 1989. Virtual Time and global states of distributed systems. In: *Proceedings of the International Workshop on Parallel and Distributed Algorithms* (edited by Cosnard, M.). North Holland, 215-226.
- McCanne, S. & Jacobson, V. 1995. VIC: A Flexible Framework for Packet Video. In: *ACM Multimedia*, San Francisco, 511-522.
- McCanne, S. 1999. Scalable Multimedia Communication - Using IP Multicast and Lightweight Sessions. *IEEE Internet Computing* 3(2), 33-45.
- Michaelson, R. 1999. Web-based Group Work. In: *Proceedings of the 10th Annual CTI-AFM Conference*. CTI Accounting and Finance, Brighton, 58 – 64.
- Michaelson, R., Helliard, C., Power, D. & Allison, C. 2002. Group Work and the Web: Finesse and TAGS.
- Minoli, D. & Schmidt, A. 1997. *Client/Server Applications on ATM Networks*. Manning.
- Mistra. 1986. Distributed Discrete-Event Simulation. *ACM Computing Surveys* 18(1), 39-65.
- Mogul, J. C. 1995. The case for Persistent Connection HTTP. In: *ACM SIGCOMM '95*, 299-313.
- Montgomery, T. 1994. *Design, Implementation and Verification of the Reliable Multicast Protocol*, West Virginia.
- Nicoll, J. R., Ruddle, A. & Allison, C. 2002. Redressing Server Limitations in Network Service Delay. In: *PGNet 2002, the 3rd Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*" (edited by Merabti, M.). JM
- Obiczka, K. 1998. Multicast Transport Protocols: A Survey and Taxonomy. *IEEE Communications* 36(1), 94-102.
- Ousterhout, J. K. 1994. *Tcl and the Tk toolkit*. Addison-Wesley.
- Perkins, C. S., Hodson, O. & Hardman, V. 1998. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. *IEEE Network*.

References

- Peterson, L. L., Hutchinson, N. C. & Schlichting, R. 1989. Preserving and Using Context Information in interprocess communication. *ACM TOCS* 7(3), 217-246.
- Plank, J., Beck, M., Kingsley, G. & Li, K. 1995. Libckpt: Transparent Checkpointing under Unix. In: *Usenix Winter 1995 Technical Conference*. Usenix Association.
- Power, D. M., Michaelson, R. & Allison, C. 1998. The Finesse Portfolio Management Facility. In: *9th CTI-AFM Conference* (edited by Fletcher, K. & Nicholson, A. H. S.). CTI-AFM, York, 119 -125.
- Prakash, A. & Shim, H. S. 1994. DistView: Support for Building Efficient Collaborative Applications using Replicated Objects. In: *Proc. of the ACM Conference on Computer-Supported Cooperative Work*. ACM, 153-164.
- Presley, M., Ebling, M., Wieland, F. & Jefferson, D. 1989. Benchmarking the Time Warp Operating System With a Computer Network Simulation. In: *SCS Multiconference on Distributed Simulation* (edited by Unger, B. & Fujimoto, R.). Society For Computer Simulat
- Ramakrishnan, K. K. & Floyd, S. 1998. A proposal to add Explicit Congestion Notification (ECN) to IP. In: *draft-kksjf-ecn-03.txt*.
- Ramsay, J., Barbesi, A. & Preece, J. 1998. A psychological investigation of long retrieval times on the World Wide Web. *Interacting with Computers* 10(10), 77-86.
- Ritchie, J. 2000. The USE of MANs Initiative: Achievements, Outcomes, Recommendations. SHEFC, Edinburgh, 1-58.
- Ritchie, J. 2002. The ScotCIT Programme: Achievements, Outcomes, Recommendations. SHEFC, Edinburgh, 1-87.
- Rodriguez, P., Spanner, C. & Birsack, E. W. 2001. Analysis of Web Cahcing Architecture: Hierarchical and Distributed Caching. *IEEE/ACM Transactions on Networking*, 404-418.
- Sasse, M. A. & Handley, M. J. 1996. Collaborative Writing With Synchronous and Asynchronous Support Environments. In: *Groupware and Authoring* (edited by Rada, R.). Academic Press, 205-218.
- Satyanarayanan, M. 1995. *RPC2 User Guide and Reference Manual*. School of Computer Science, Carnegie Mellon University.
- Schmuck, F. 1993. Efficient Broadcast Primitives in Asynchronous Distributed Systems. In: *Reliable Distributed Computing with the ISIS Toolkit* (edited by Birman, K. P. & Van Renesse, R.). IEEE, 263-283.

References

Schneider, F. B. 1993. Replication Management Using the State Machine Approach. In: Distributed Systems (edited by Mullender, S.). ACM Press, New York, 169-215.

Schroeder, T., Goddard, S. & Ramamurthy, B. 2000. Scalable Web Server Clustering Technologies. *IEEE Network* 14(3), 38-45.

Schulzrinne, H. & al., e. 1996. IETF RFC 1889 RTP: a transport protocol for real time applications.

Scotcit. 1998. The Scottish Higher Education Communications and Information Technology Programme. <http://www.scotcit.ac.uk>.

Security, S. C. 2001. SSH: Secure Shell.

Sikkel, K., Neuman, O. & Sachweh, S. 1998. Process Support for Cooperative Work on the World Wide Web. In: 6th Euromicro Workshop on Parallel and Distributed Processing. IEEE, Madrid, 325-331.

Sikkel, L., Gommer, D. & van der Veen, J. 2001. A Cross Comparison of BSCW in different educational settings. In: European Perspectives on Computer- supported Collaborative Learning, the proceedings of Euro CSCL 2001 (edited by Dillenbourg, P., Eurelings,

Silbershatz, A. & Galvin, P. 1994. Operating System Concepts. Addison-Wesley.

Spero, S. E. 1998. Analysis of HTTP Performance Problems, <http://sunsite.unc.edu/mdma-release/http-prob.html>. UNC.

St. Laurent, S. 1998. Cookies. McGraw-Hill.

Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. & Tatar, D. 1987. WYSIWIS Revised: Early Experiences with Multiuser Interfaces. *ACM Transactions on Office Information Systems* 5(3), 147-167.

Suchman, L. 1993. Do Categories Have Politics? The language/action perspective reconsidered. In: Third European Conference on Computer Supported Cooperative Work. Kluwer Academic Publishers, Milan.

Sun_Microsystems. 1990a. Network Programming Guide.

Sun_Microsystems. 1990b. Programming Utilities and Libraries.

Sun_Microsystems. 2001. JMF: The Java Media Framework, <http://java.sun.com/products/java-media/jmf>.

Tanenbaum, A. S. 1981. Computer Networks.

References

Tanenbaum, A. S. 1987. Classical IPC Problems. In: Operating Systems Design and Implementation. Prentice-Hall, 75-77.

Tanenbaum, A. S. 2002. Computer Networks. Prentice-Hall.

tcpdump:. <http://www.tcpdump.org/>.

Teitlebaum, B., Hares, S., Dunn, L., Neilson, R., Narayan, V. & Reichmeyer, F. 1999. Internet2 Qbone: Building a Testbed for Differentiated Services. IEEE Network 13(5), 8-16.

Thayer, R., Doraswamy, N. & Glenn, R. 1998. IP Security Document Roadmap.

The_ATM_Forum. 1996. Traffic Management Specification Version 4.0.

UKERNA. 1999. PIPVIC: Piloting IP Video Conferencing.

UPDATA. <http://www.updata.co.uk>.

W3C. 2002. W3C Accessibility Guidelines: <http://www.w3.org/WAI/Resources/#gl>

Wang, Z. & Crowcroft, J. 1991. A new congestion control scheme: Slow Start and Search (Tri-S). ACM Computer Communications Review 21(1), 32-43.

Watson, A. & Sasse, M. A. 2000a. Distance Education via IP Videoconferencing: Results from a National Pilot Project. In: CHI'2000, The Hague, Netherlands, 113-114.

Watson, A. & Sasse, M. A. 2000b. The Good, the Bad, and the Muffled: the Impact of Different Degradations on Internet Speech. In: ACM Multimedia 2000. ACM.

WebCT. 1999. <http://www.webct.com/>

Weihl, W. E. 1993. Transaction-Processing Techniques. In: Distributed Systems (edited by Mullender, S.). ACM Press/Addison-Wesley, 329,352.

White, P. P. 1997. RSVP and Integrated Services on the Internet: A Tutorial. IEEE Communications 35(5), 100-106.

Yeong, W., Howes, T. & Kille, S. 1995. RFC 1777: Lightweight Directory Access Protocol. IETF.