Support for Collaborative Authoring via Email: The MESSIE Environment

Martina Angela Sasse, Mark James Handley
Department of Computer Science, University College London, UK

and

Shaw Cheng Chuang
Computer Laboratory, Cambridge University, UK

Abstact: MESSIE is a collaborative authoring environment to support the production of large-scale documents by teams of geographically distributed groups of authors working with hetereogenous systems. The environment allows authors to submit text at various stages of gestation (e.g. list of topics, first draft) to a shared filestore via email. All authors collaborating on a document can read each others' contributions, and add suggestions, comments and additional material directly to the document. The system integrates automatically answered electronic mail, shared file store administration, and a version control tool in a UNIX environment. The paper describes design and implementation strategy, and reports observations and a number of changes which were made during a 4-month trial period with three collaborative authoring teams.

1 Introduction

Collaboration between geographically dispersed groups is becoming increasingly common. In Europe, there are a number of programmes, such as ESPRIT, RACE and DELTA, to promote research collaboration on an international scale. Most of the projects funded in such programmes require the joint authoring of comprehensive reviews, reports or large-scale technical documentation. Most authoring teams hold regular meetings, and these meetings are a considerable drain on authors' time and travel funds.

Being involved in many such projects, we began to look for collaboration support which could reduce the number of meetings related to joint authoring activities. We conducted a case study to investigate the use of Multimedia Conferencing as a support environment (Baydere et al., 1993). We found that Multimedia Conferencing provided the rich channels of communication which creative groupwork supposedly requires (Chalfonte et al., 1991), and is normally only achieved through face-to-face meetings.

Most teams of authors, however, do not have access to expensive Multimedia Conferencing systems. Even if they did, previous research (Grudin, 1990) has shown that less technology-experienced users than the ones in the case study would be likely to reject such a sophisticated groupware system because of the learning overhead required.

An additional insight gained from the case study was that only the initial phases of document production (generating ideas and determining scope and structure of the document) could be described as *creative*. The other phases did not necessarily require rich channels of communication - authors actually preferred *asynchronous* communication via email, since they felt it was more effective than synchronous sessions. The case study also re-inforced previous observations that considerable time and effort needs to be spent on *managing* a collaboratively authored document. The problems of managing the process of producing large documents between a number of project members in different locations can be summarised as follows:

- Document integrity

  Authors will send copies of their contributions to other authors for information and comment, and amend their contribution as they receive feedback. Since it would require considerable effort to send a new version to all other authors every time a change is made, or only send it to the co-author in response to whose suggestion the change was made, different authors might hold different versions of some parts of the document.

- Duplication of effort

  Several authors might write the same comments or supply the same additions to the document. Authors might repeat explanations or background material which is already covered in other parts of the document.

- Integration of contributions

  Contributions written by different authors are likely to vary in use of terminology and style. Since this does not make for a very readable document, the project member charged with editing the final version has to spend considerable time and effort to (a) integrate the contributions into a coherent and readable document, and (b) provide cross-references between various parts of the document.

- Editing and formatting

  Most authors would prefer to use their favourite word processor or editor and text formatter to produce their contributions. These are likely to be different tools in large authoring teams. If authors do not use the same tool, considerable effort needs to be spent on re-typing and re-formatting parts of the document. Teams in which authors use different tools often supply the person doing the final formatting with an ASCII file and a hardcopy of the formatted version - this approach avoids re-typing text but formatting needs to be done at least twice.

We decided to identify a set of requirements and design objectives for an asynchronous colloboration environment for collaborative authoring to provide support for dealing with these document management problems. The original system requirements and design objectives are elaborated in Section 2, and the implementation is described in Section 3. The user's view of interaction with the environment is described in Section 4, followed by a summary of results from the trial phase in Section 5.

2 Requirements and Design Objectives

The intent was to specify a simple system which could be installed locally and administered independently by each authoring team. We wanted to implement a basic system quickly by using existing tools, and offer it to a number of authoring teams for producing real documents to gain feedback for further development and improvement.

At the outset of the project, we started with two sets of requirements for the environment: requirements of *individual authors* and requirements of the *administrator*, a role assumed by one member in every joint authoring team using the environment.

The *author* requirements we identified we identified for asynchronous collaborative writing support were:

1. Make drafts available as soon as possible

   To ensure misunderstandings are discovered as early as possible, and to ensure that the document grows in a uniform way, it is important that early drafts of sections are made available early in the writing process.

2. Preserve the integrity of the document

   In order to preserve the integrity of the document, or its various parts, all authors should have access to the latest version of any file. Authors should have the facility to work on a single master copy of each part of the document. Clearly, authors should not be allowed to edit the master copy of a document section while another author is editing it.

3. Avoid duplication of effort

   In order to avoid duplication of effort, all comments and additions to a document should be entered into the document itself, so that authors can identify which comments and additions have been made by other authors.

4. Distribute editing and formatting work

   In order to distribute the effort involved in final editing and formatting of document, a prime requirement was to allow authors to exchange *revisable* text as much as possible. Imposing any single document exchange format for authoring teams would preclude this. Whilst it is not possible to support WP and DTP applications which produce non-revisable formats, the environment should support handling of a variety of revisable text formats in addition to ASCII. In order to deal with diagrams, the system should handle PostScript, which, even though it is non-revisable, is so ubiquitous that most authors can view or print such files locally. There is also a requirement for tools which facilitate compilation of reference lists and glossaries.

5. Avoid large learning overhead

   The system should be simple and transparent in use, and require users only to learn and remember a small number of commands. Where possible, it should allow users to use familiar tools for familiar activities.

6. Access without direct login

   Not all authors have the facility to directly access remote machines today. In addition, it is important to consider that not all sites who might want to install such an environment would want to allow remote logins and give direct access to a shared filestore facility.

7. Deal with heterogeneity

   Although there are several synchronous authoring tools available, this system must provide access from a wide range of remote systems. No existing tool would run on all the available remote systems, and the overhead in developing any software to run on all such systems would be too great. Thus the system should only involve one installation - at the site where the authoring team's version of MESSIE and the filestore are kept.

8. Policy-free collaboration

   The system should be as policy-free as possible. It should provide the basic collaboration environment, but the users should decide the details of how that environment should be applied to their collaborative task.

The requirements of the authoring team's *administrator* can be summarised as follows:

1. The system should be simple to install, maintain, and port.

2. Storage overhead for the documents should be kept to a minimum.

3. It should be possible to manage documents remotely as well as locally.

We decided that these requirements could be met by integrating and developing the functionality provided by existing tools - shared filestore, electronic mail, and a version control tool - into a support environment which would provide authors with a basic set of facilities to submit, read, edit, delete and list files. The environment allows authors to submit text at various stages of gestation (e.g. list of topics, first draft) to the shared filestore via email. All authors collaborating on a document can request files submitted by the other authors, and add suggestions, comments and additional material directly to the document.

3 MESSIE Design and Implementation

All parts of a collaboratively authored document supported by the environment are held in a *shared filestore*, which is administered by one team member. The document can be created and accessed by sending files and requests by *electronic mail* to a MESSIE email account. MESSIE accepts email messages containing MESSIE commands and new text, performs the actions specified by the command (subject to access control), and returns the final status and results. This is shown in Figure 1:

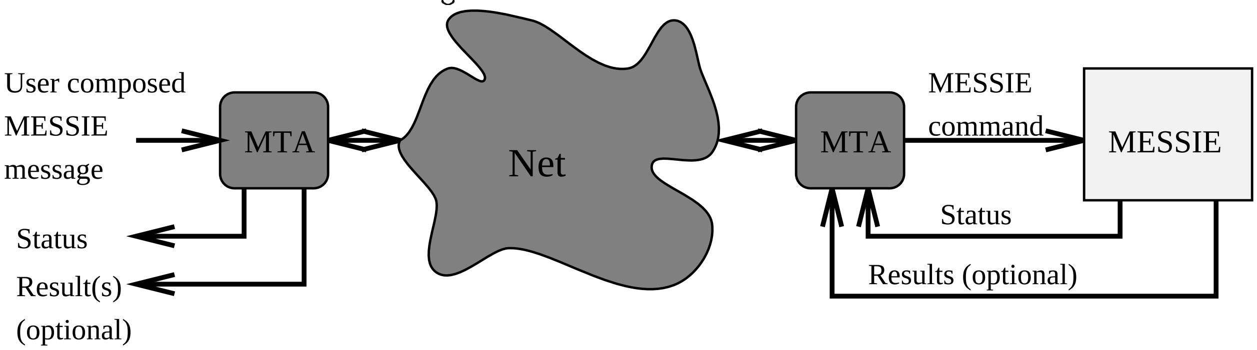

Figure 1: Interacting with MESSIE

MESSIE places no specific requirements on the end-user environment. An author composes a message using their favourite mail interface program - on a PC, workstation, mini or mainframe, running their respective operating systems. The message is then submitted to MESSIE as a command file. There is no restriction on the type of mail system authors can use, as long as the message can be gatewayed intact to the MESSIE address. When a message addressed to MESSIE arrives in the MESSIE mailbox, MESSIE is activated (using one of the mechanisms described in the following section). A status message will always be returned to the user to provide an overview of the outcome of the command submission. Optionally, if the command generates outputs for the user, these outputs will also be returned to the author in message separate from the status message.

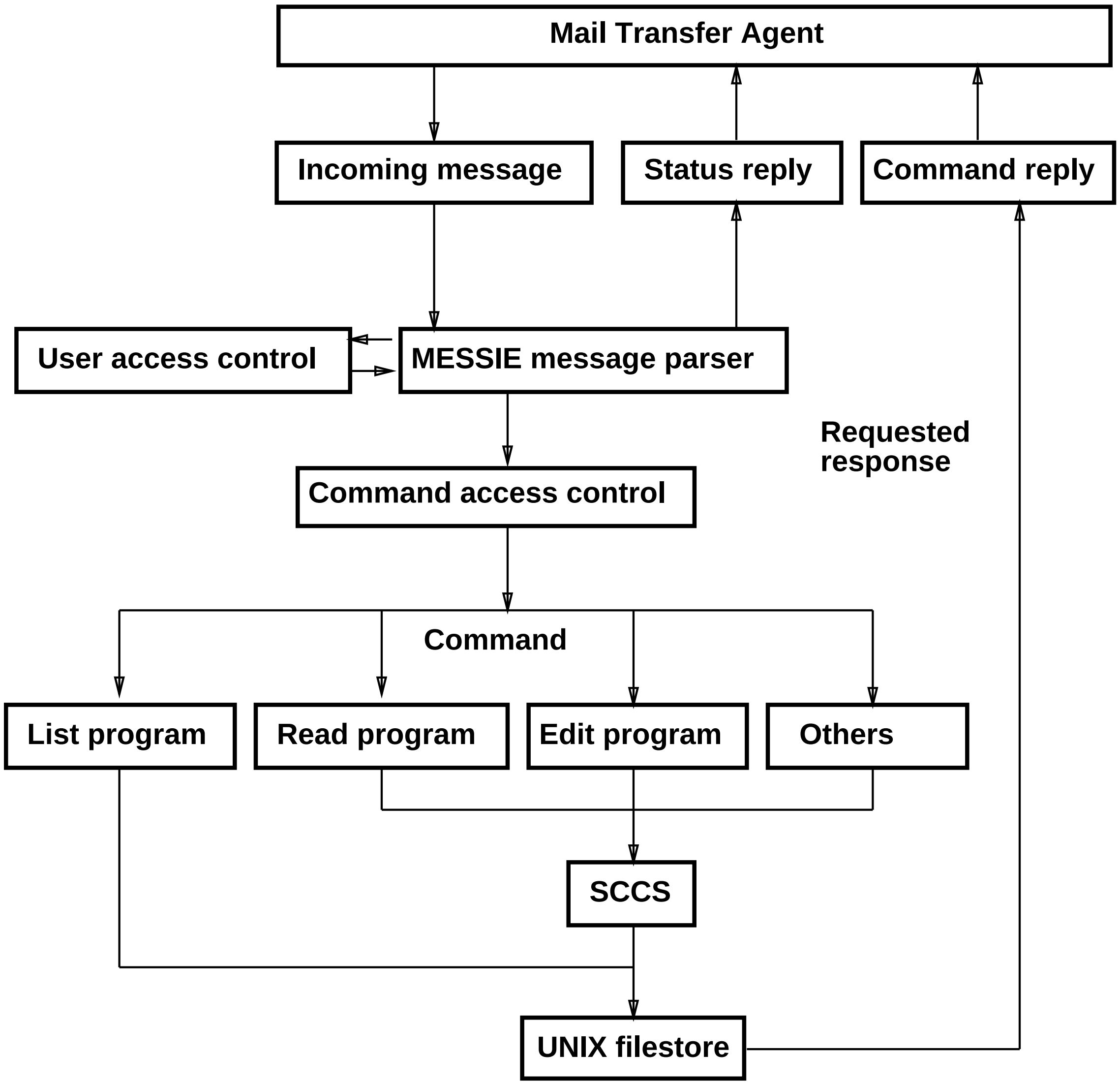

Figure 2: MESSIE Structure

3.1 MESSIE Internal

In this section, we will take a closer look at the MESSIE box as shown in Figure 1. The logical relationship between MESSIE components is illustrated in Figure 2.

MESSIE has been designed to consist of two highly independent parts. The first part is a generic *command interpreter*, which performs validation and execution of commands, user access control, and notifications. The command set and user access and addressing information is stored in separate databases, currently implemented in dbm. This allows MESSIE to be easily expanded to add new functionality.

The *command database* uses the command name as key to locate the actual program which will perform the actions of the command. The access mode of the command can also be indexed. Currently, only *read* and *write* access modes are supported. These access modes are used in conjunction with the user access authorisation.

The *user database* uses the mail-id as a key to check the user's access authorisation. For every command to be executed, the user authorisation is checked against the command's access mode. Only users with the correct access authorisation are allowed to execute a particular command. To simplify administration, a wildcard authorisation is also allowed. A wildcard of read-only access would turn MESSIE into an info-server.

The second part of the environment are the MESSIE *command executables.* These could be implemented in any language, independent of the front-end command interpreter. The front-end communicates with the MESSIE commands using shared-file message passing. This avoids using any operating-system dependent IPC mechanisms. Since there are no concurrency activities in MESSIE, very simple message passing mechanisms can be used.

The command executable also writes the status of the command execution to the same status file used by the front-end. Thus, after execution of the commands in a message, a single status message is always returned to the user. The individual commands will also return their own messages where appropriate. By having a separate command status file, the user will always be given feedback on a particular MESSIE job submission. This also enables the command executable to send as many (or as few) messages as are required (e.g. one for each file requested for reading or editing). This helps to avoid the creation of large return messages, which could potentially cause problems with some mail systems.

Since MESSIE's front-end and command set are virtually independent, MESSIE can be used as a generic e-mail based remote command shell (see Section 7).

3.2 Implementation Details

Since MESSIE has been implemented using a number of simple C programs (approximately 100 lines of code), and shell/sed/awk/perl scripts, it contains very little system-specific code. It adopts a modular design approach: the components used can be substituted by others. If a team, for instance, wants to use a version control system other than SCCS (e.g. RCS), they can substitute it. The few operating-system dependent details can be easily re-written. The modular design approach also makes it easy to extend existing functionality.

The shell/sed/awk/perl scripts are mainly used for the construction of command executables. We found this approach to be extremely useful for rapid prototyping and experimentation: the current set of commands was coded in less than 40 man-hours. Certain commands were subsequently re-written, either to improve performance or extend functionality, in response to the results of experimentation with the first version. For instance, the *list* command, originally written in *bourne shell*, was re-written in *perl* - this increased the speed with which the command could be executed by a factor of 100.

4 User view of MESSIE

4.1 Email Access

All parts of a collaboratively authored document supported by the environment are held in a shared filestore, which is administered by one of the authors. (During the implementation and evaluation period, all those documents were are held in a filestore at UCL, which was administered by the authors.) The document can be created and accessed by sending files and requests by electronic mail to a MESSIE email account (see Figure 3 for an example message).

```
From: a.sasse@uk.ac.ucl.cs
To: messie@uk.ac.ucl.cs
Subject:
-----------------------------------------------------------
#COMMAND read Race_deliverable/chapter2
```

Figure 3: Example of a email message to MESSIE (for read-only copy of a file)

4.2 MESSIE commands

MESSIE understands a set of commands contained in the body of an email message sent to the email account. All commands have to be placed a the beginning of a line, starting with the instruction #COMMAND, and consisting of a command name and directory/file name (see Figure 3 one for an example of the READ command). The basic command set is fairly small, covering the following basic activities:

- submitting a new file
- reading an existing file
- editing an existing file
- deleting an existing file
- obtaining a listing of a directory

4.3 Return messages, status reports and notifications

Regardless of the status of the request (i.e. whether it was successful or not), MESSIE will always send a request status report back to the sender (see Figure 4 for a successful request, Figure 5 for a failed one).

```
To: a.sasse@uk.ac.ucl.cs
Subject: Request reply read Race_deliverable/chapter2
From: The UCL-CS Messie Service <messie@uk.ac.ucl.cs>
------------------------------------------------------
read Race_deliverable/chapter2: Valid command
Read done
```

Figure 4: Example of request status report (successful)

```
To: a.sasse@uk.ac.ucl.cs
Subject: Request reply read Race_deliverable/chapter2
From: The UCL-CS Messie Service <messie@uk.ac.ucl.cs>
------------------------------------------------------
read Race_deliverable/chapter2: Valid command.
edit error: Race_deliverable/chapter2 does not exist;
   please choose another name
```

Figure 5: Example of request status report (unsuccessful)

For requests that generate a return message (e.g. the *read* command) this return message will always be sent in a separate email message (see Figure 6). The message *subject field* in both return messages and status reports will indicate in response to which request the message is sent.

```
To: a.sasse@uk.ac.ucl.cs
Subject: Request reply: read Race_deliverable/chapter2
From:  The UCL-CS Messie Service <messie@uk.ac.ucl.cs>
------------------------------------------------------
WARNING: This is a read-only copy

Chapter 2: The need for remote collaborative authoring support

The need for an environment to support collaborative authoring
of large-scale technical documents, such as Esprit, Race and
Delta deliverables, has been established in Chapter 1.  In this..
```

Figure 6: Example of return message

MESSIE can handle multiple commands in one request - it is possible to request more than one file in a single email message. Each file will be returned as separate email message.

In addition to return messages and status reports, MESSIE sends a variety of notifications to authors when requested files are locked, when checkout periods expire, and when changes have been made to a file.

4.4 Shared filestore

The shared filestore is a tree-based structure of directories and files similar to UNIX and MS-DOS filestores (and supports both UNIX and MS-DOS style file naming conventions). To find out what files are available in the directory containing the document, authors can request a listing of the directory contents. MESSIE will return a list of all existing files and their current status along with any embedded meta-data (see 4.8).

All files submitted to MESSIE will have one owner - the author who first submitted a particular file become its owner. The owner is notified of any changes made to the file (see 4.5), and only the owner can delete an existing file.

4.5 Version Control

Once a file has been checked out for editing, MESSIE will lock its copy of the file for the specified amount of time - the file cannot be checked out for editing until it is returned, or the timeout period has expired. This locking mechanism is the simplest way to preserve the integrity of the individual files in the document. The tool employed to implement version control in MESSIE is the Source Code Control System (SCCS), though other version control tools can be substituted.

Read-only copies can still be obtained while a file is locked, and are automatically sent when a request for editing a locked file is received. If a file which has been checked out for editing is not returned within 48 hours (or the time specified at check-out), MESSIE will assume that the checked-out copy of the file has been "lost" and unlock the last version. This *timeout* function prevents files from remaining inaccessible if an author requests an edit and then forgets to return the file.

When a file is returned, MESSIE registers this as a new version of the file. If an altered version of a file is returned after the timeout has expired, MESSIE will only accept it as a new version if the file has not been checked out by someone else since the lock was removed.

All changes made to a file are registered, and so all versions of a document can be accessed if authors wish to do so, and MESSIE will provide any *diff* file on demand. Furthermore, the owner of a file is notified when changes have been made to a file (through an email message containing the name of the person who has made the changes and a list of the changes). If other authors want to receive these notifications for any file they do not own, they can join a subscription list. This *subscription facility* was added following authors' suggestions.

4.6 Diagrams

Diagrams, figures and drawings are stored as PostScript files. Read-only versions of PostScript files will be sent in response to requests for these files. Authors can print or view copies of these on their home printers. Each new version of a picture will be a new file with its own filename (e.g. pic1, pic2). Each diagram file has an associated text file, into which comments can be entered. The comment filename is naming using the convention of diagram file name suffixed by the word "comment" (e.g. the name of the comment file for diagram pic1 will be pic1.comment). Comments can be appended to this file using the add_comment command and read-only copies of the comment file can be requested.

4.7 Glossary and Reference Files

One of the biggest problems associated with collaborative authoring of larger documents tend to be the time and effort involved in compiling a glossary and references as a deadline approaches. The environment provided by MESSIE allows authors to collect such information throughout the project in a joint file for the glossary, and a joint file for references. Authors request a current version of these files, and, if the glossary entry or reference does not exist, send a message which appends the glossary entry or reference to the file.

4.8 Document history

SCCS only provides a mechanism to achieve version control. A set of protocols is still required to ensure effective joint authoring. Typically, sections of a document are held in separate files. The owner of each file is the principal author, who is ultimately responsible for this section of the document. When the first version of a document is submitted to MESSIE, it is automatically put under SCCS control, and then can be commented on by the other authors.

A modification history should be kept at the top of each file. An example of such a modification history is shown in Figure 7.

```
#
# MODIFICATION HISTORY
#
#      DATE         YOUR_NAME      MARKER      REMARK
#    14/10/92       MH              W001    2nd attempt at draft
#    15/10/92       AS              W002    Revised, additions in 3.1
#    26/10/92       TC              W003    Comments througout
#    28/10/92       MH              W004    Comments incorporated
```

Figure 7: Example of modification history

This is necessary because although SCCS stores this information, it is not immediately obvious to an author when reading the actual text of the most recent version. [1]The *marker* is used indicate the exact place to in the document text where changes have been, and can

[1]SCCS does not visibly mark where changes have been made in the file. In order to identify changes, authors would have to view the current version and diff file(s).

be used to locate comments by particular authors. The owner of each file will remove the markers when dealing with the comments.

Furthermore, the creator of a file is encouraged (though it is not mandatory) to include at the beginning of the file some of the following entries.

```
#TITLE          title of document
#DESCRIPTION    A brief statement of what this file is
#AUTHOR         author and responsible person of document
#LEDIT          last editor of document (filled in automatically by MESSIE)
#DATE           last modify day month year (filled in automatically by MESSIE)
#IDENTIFIER     unique document identifier  (filled in automatically by MESSIE)
#VERSION        version number (filled in automatically by MESSIE)
#TYPE           type of document (Monitor point, Minutes, Document ..)
#STATUS         status of document (draft, out for approval, approved
#CONFID         confidentiality (WP internal, CAR internal, external)
#ACTION         actions that should be applied to the document
#TO             list of those who should carry out the action
```

All information appropriate to these file fields can, if desired, be provided by the document owner using the above commands. The information given in those fields appears in listing of directories requested by authors, thereby making it easier for authors to identify files which they want to request for reading and editing.

4.9 Commenting

It is important that authors can easily recognise which parts of a document have been changed or added, when and by whom. This information can be obtained from *diff* files (Neuwirth et al., 1992). Like Beck & Bellotti (in this volume), however, we found that authors prefer to have this information grounded in the document itself. In order to achieve this, authoring teams need to agree a set of rules - for which we have coined the term *human protocols* - for commenting. A example for human protocols (developed by one of the teams involved in the trials) is:

1. If the change is very small, such as spelling, an omitted word, etc, make the change without marking that you have changed it.

2. Make any additions to the document in such a way that they stand out from the original text. (our convention is [**W003 TC: this is a comment**]). A more complete example looks like this:

```
..............
Synchronous communication occurs when two or more people
```

```
interact simultaneously and in real-time, e.g. in a telephone
conversation [**W003 TC do we need both examples**] or a video
conference.
```

The marker (eg. ***W003) should always be at the place of change. It is always useful and convenient to put both the commenter's name and a brief comment next to the marker to give some clue of why changes are made.

3. No text should be deleted by a commenter. Mark the text for deletion or replacement, but let the person responsible for the section include the changes as they see fit:

```
[**W004 Start MH 9/10/91 The above text should be replaced with:
Text should not be deleted...
W004 End **]
```

5 Results of the trial period

In order to evaluate the effectiveness of MESSIE as a support environment for collaborative authoring by email, we released the system to a restricted group of users for a trial period of 4 months. All groups used the system during this period to produce a real, life-size document:

- Group 1: 10 authors at 5 sites in the UK and Germany, producing a 200-page final deliverable for a RACE project over a period of 4 months;
- Group 2: 3 authors at 3 different sites in the UK and Belgium, producing a 40-page intermediate deliverable for a DELTA project over a period of 2 months;
- Group 3: 3 authors at 2 different sites in the UK, producing a 6-page conference paper over a period of 3 weeks.

Altogether, the users issued 856 commands, divided as follows:

```
272 Read commands
202 List commands
155 Submit commands
 96 Edit commands
 65 Write commands
 35 Delete commands
 10 Requests for Help
 21 other commands
```

The general response of those users to the system was very positive indeed, the single most important factor being that users could continue to use their own email and text editing facilities, and only had to learn a small number of additional commands in order to use the system effectively - authors felt that the environment provided very useful support for very little investment.

Authors were able to produce their contributions, and read others, using the hardware and software that they were familiar with: IBM PCs, Apple Macs, Sun workstations, and IBM and DEC mainframes. All groups started out storing the documents in ASCII format. In Group 1, most authors started to store their files in RTF format, since the final document was to be formatted in MS-Word. The document in Group 2 remained in ASCII format until the very end, and was formatted by one of the authors after the collaborative authoring had finished. Group 3 merged the files halfway through the writing process into a single document, and formatted it using LaTeX. On two occasions, TeX or PostScript files were damaged in transit, a surprisingly low number considering the total number of reads and edits performed on formatted files.

On the basis of the logs of the system use, and authors' comments, we compiled a list of desired changes, which have been incorporated into MESSIE 2.0. These changes fall into two different groups: changes to the commands and messages, and additional functionality. Commands and messages are briefly described here, whereas additional functionality is discussed in section 6.

- *Command recognition*: most frequent cause of failed requests was mistyped command names (e.g. "sumbit" instead of submit or "lsit" instead of list, or lower-case "#command"). Instead of just returning these as errors, the system can be made to recognise typos and execute the "most likely" request, and return it together with the error message.

- *Error recognition*: mis-remembered command names (e.g. "create" instead of "submit", or "write" instead of "submit"). Again, an aliasing mechanism can be set up to recognise the most common confusion, execute the most likely request, and remind the user of the correct command. Obviously this should only happen where no document will be damaged by the assumption.

- *Return files with failed submissions*: There were several cases where "write" commands failed because authors mis-spelt filenames or incorrect directory paths. Even though authors were requested to keep a copy of all submissions until the status request report confirmed successful submission, we found that many did last-minute editing of files in their mail editor, and neglected to save the emailed version. Since MESSIE did not return failed submissions, the last-minute changes were lost to the author (though they could be retrieved from the system backup mailbox). Now, the complete file is returned to an author when submissions fail.

6 Discussion

In developing MESSIE, we have attempted to provide a general-purpose environment to support collaborative authoring with a minimal set of user commands. We decided to start with a simple, basic environment and add functionality as requested by users. We adopted this design strategy in an attempt to dodge the fate of some sophisticated groupware systems, which were rejected by their intended users (Grudin, 1990), and feel that the approach was successful.

Clearly, there is some debate over exactly which commands should be provided for the users. This is largely due to different groups' models for interaction with the system. For instance, if the system is to be strongly locally administered, providing a "delete" command for remote users may be undesirable. However, if the system is to be administered in a more distributed fashion, more powerful (and potentially dangerous) commands (as described by Borenstein, 1992) will need to be provided for remote use. Our current command set lies somewhere between these two extremes, but the advantage of our approach to the overall system design is that the local administrator can decide on the level of support that will be provided, and thus provide the appropriate command set to support this style of management.

Currently, MESSIE does not provide direct support any form of *structured* documents - its model is that a document consists of a set of sections (files), and it is entirely up to the authoring teams to decide whether and how a document should be partitioned into sections. This is in line with our attitude that the system should be as policy-free as possible. However, it may be desirable to also provide some document structuring commands, whereby a user can request, for instance, an entire document. The structuring command would then utilise document meta-data to return the entire document in one piece (rather that as distinct sections). It would also be possible to link such a tool to a text formatter, and return, for instance, a PostScript version of the entire document including diagrams. At the present, we consider such tools outside the scope of what is intended to be a policy-free minimal system, but there is nothing to stop a user group from deciding on a policy, and extending the functionality to by adding such commands.

7 Conclusions and future work

MESSIE has the potential to be used as a general collaboration tool. Although much effort is currently being put into synchronous collaboration tools, many forms of collaboration do not need to be very tightly coupled. It is currently used by the (geographically very much distributed) Executive Committee of the British HCI Group (A Specialist Group of the British Computer Society) as a document store and organizational memory: committee members submit and request PostScript and RTF files for printing letterheaded paper and mailing labels, templates for forms and letters, etc., as well as use it for collaborative production and editing of minutes and policy documents.

It has been suggested that MESSIE be used for collaborative authoring of software. Another project to which we aim to apply it is ESPRIT project MICE (Kirstein, 1992) to handle the booking of resources for full-scale video conferences between a number of European sites.

In general, MESSIE may be suitable for many tasks that require loose collaboration, but where the overhead involved in porting synchronous software to all possible remote systems is too great.

Acknowledgements

MESSIE was designed and implemented as part of the RACE CAR project, funded by the Commission of the European Community. The authors would like to thank Jon Crowcroft, of the Department of Computer Science at UCL, for helpful comments and suggestions, and the UK CSCW Special Interest Group and the London Unix User Group for feedback given on earlier presentations on the system. Finally, a tribute to the groups of authors who used MESSIE over the last year - the observations we collected, and the feedback given during the trial period very much shaped the design and implementation.

References

Baydere, S., Casey, T., Chuang, S., Handley, M., Ismail, N. & Sasse, A. (1993): "Multimedia Conferencing for Collaborative Writing: A Case Study." In Sharples, M. [Ed.]: *Computer Supported Collaborative Writing.* Berlin: Springer. pp. 113-135.

Borenstein, N. (1992): Computational Mail as Network Infrastructure for Computer-Supported Collaborative Work. In *CSCW'92: Proceedings of the Conference on Computer-Supported Collaborative Work* (Toronto, Canada, Oct. 31 - Nov. 4, 1992). New York: ACM. pp. 67-74.

Chalfonte, B. L., Fish, R.S. & Kraut, B. (1991): Expressive Richness: A Comparison for Speech and Text as Media for Revision. In *Proceedings of CHI, 1990*, (Seattle, Washington, April 1-5, 1990), New York: ACM. pp. 21-26.

Grudin, J. (1990): Groupware and Collaborative Work: problems and prospects. In Laurel, B. [Ed.]: *The Art of Human-Computer Interface Design.* Reading, MA: Addison-Wesley.

Kirstein, P. T. (1992): Piloting of Multimedia Integrated Communications for European Researchers (MICE). *Proceedings of the Second Packet Video Workshop*, Vol. 2 (Research Triangle Park, NC, Dec. 9-10, 1992). MCNC.

Neuwirth, C. M.; Chandhok, R.; Kaufer, D. S.; Erion, P.; Morris, J. & Miller, D. (1992): "Flexible Diff-ing in a Collaborative Writing System". In *CSCW'92: Proceedings of the Conference on Computer-Supported Collaborative Work* (Toronto, Canada, Oct. 31 - Nov. 4, 1992). New York: ACM. pp. 147-154.