USING MULTIMEDIA TO SUPPORT COOPERATIVE SOFTWARE DEVELOPMENT

Adérito Marcos and Christoph Hornung

Fraunhofer Institute for Computer Graphics
Wilhelminenstr. 7, 64283 Darmstadt, Germany
{marcos, hornung}@igd.fhg.de

Abstract

The aim of this paper is to propose a global solution to support Cooperative Software Development (CSD) through a multimedia environment. CSD systems have as main goal to enable several users connected over a network to work together in order to develop software products. They have to solve problems such as: coherence maintenance of the software project through the distributed system by managing possible conflicts between local versions of each group member and, above all, promote the necessary mechanisms for the inter-group awareness and integrity.

We introduce here the strategies of our own CSD prototype: a computer-supported cooperative work architecture for software development. It enables a group of developers (2 to 4), possibly located at remote places and connected over network, to develop software together. A cooperative multimedia editing environment is available for the whole Development Cycle, enclosing mechanisms of computer-conferencing (text, audio and video communications).

Keywords: CSCW, Cooperative Software Development, Multimedia, Consistency Conflicts, Computer-Conferencing.

1 Introduction

In the last years, the increased evolution on both network and workstation technology and also on multimedia user-interface metaphors, had enabled systems designers to present more and best concrete solutions on distributed environments. Following this, groupware and CSCW systems are being developed covering several specific cases of real group activities.

We deal here with the problems posed by Cooperative Software Development - a specific case of CSCW, which encompasses the questions of supporting the activities of a group of people that cooperates in order to produce a piece of software. We mean as Software Development those tasks directly related with code programming and software research implementations not expressively using CASE (Computer-Aided Software Engineering) technology.

The work in development and maintenance of software is typically alternating between tasks involving many persons and individual assignments [8]. This development work is performed following a Software Development Cycle (SDC) which includes activities like conceptualisation, design and specification, editing, integration of software components, debug, test, review, and others.

As a first step, the group has to conceptualise and decide guidelines and strategies to be used during the development process, and also divide implementation tasks and responsibilities among themselves. After this phase each developer will concentrate in his own tasks, and starts properly the development process.

It has been verified that almost all the implementation work tends to be done asynchronously and more or less independently, where each developer only carries out the task(s) assigned to him. However, points exist during the development when two or more developers want to collaborate by completing together a specific goal (sharing or not software object(s)), or simply exchange opinions about details on the software project [10].

Global meetings occur when the group needs to (re)consider together the software project state (e.g. test up results or redefine strategies). Then, all the changes achieved in the system by each developer appear to fit in an unique version, turning the system to the same global state. However, and due to the distributed characteristics of the development process, changes performed by one developer can conflict (*merging conflicts*) with the changes made by the other users in the system. This represents the main difficulty of CSD processes, i.e., support of consistency. A solution must be taken from negotiations and organisational protocol strategies [11]. We can easily devise how important (if not crucial) is allow efficient direct inter-user communications features in order to facilitate negotiations and promote the group work.

Accordingly, we have adopted a strategy following two guidelines: first, prevent *merging conflicts* from arising by avoiding absolute forms of parallel work; and second, support a strong awareness within the group by allowing mechanisms for a easy inter-group communications. In fact, when several developers want to edit in the same software object (but not in the same point), the object is divided among the candidates in strict reserved areas (not overlapping), for their own exclusive use. This permits a broad way of parallel work, in the sense that no different logical versions are created, but the same software component is being simultaneously changed. When contentious situations appear, referring to areas or even a whole object access, they can be solved by direct human (user-user) negotiations or asking the intervention of the group's moderator.

We propose in this paper a global distributed and multimedia environment to support CSD. The use of multimedia can improve CSD process in two different ways: to effectively support the necessary group communication links; and to enhance the expressiveness of the related information in the development cycle (see [13]).

We describe here the decisions and solutions adopted in our approach - a computer supported cooperative work architecture for software development. It includes a complete software development cycle (C, C++ programming dependent) and is implemented over a multimedia environment. It enables a group of developers (2 to 4), possibly located at remote places and connected over network (LAN or WAN (Internet and ISDN)), to conceptualise, decide strategy, edit and integrate software components, comment on, debug, compile, test, perform code inspection and review, and generate reports.

The system provides a strong awareness within the group by allowing the traditional paradigms of cooperative editing: personalised multiple cursors, WYSIWIS (What You See Is What I See), social roles, developers' identification, tele-pointing, multi-user interface, multi-user communication. The media available for communication are text, audio and video. The media available for editing in the several phases of the development cycle are text and graphics. The developers can exchange ideas

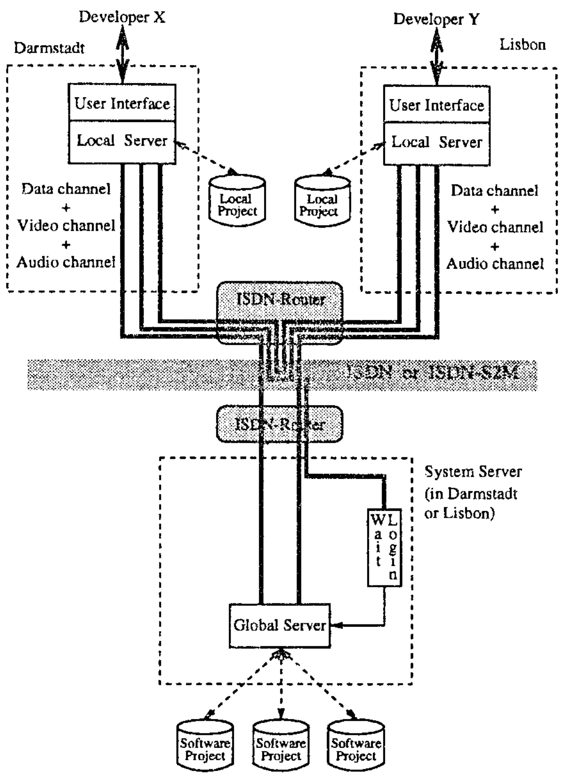


Figure 1: An overview of the CSD' main architecture (using ISDN-WAN).

about pieces of information in the editing phase by commenting on (doing public comments) in text, graphics and audio.

In this paper we firstly expose the global architecture and algorithm adopted. Next, we explain in detail the development cycle and finally we draw our future work directions and conclusions.

2 Global CSD Strategy

We mean as architecture the way the cooperative system is organised in order to enable the cooperative work. This organisation concerns about the distribution of the physical processes and files over the different machines where the users are located, as well as the way the communication is enabled. In context of CSD, the architecture concept also includes the organisation strategies of the software project.

On the other hand, the algorithm to support the CSD process encircles the strategies to control the information flow through the distributed system, concerning the issues to maintain global coherency.

2.1 System Architecture

CSD systems require special attention due to software consistency support. Indeed, CSD architectures should take in consideration where and how the software project data can be physically organised according to its parts already stated as consistent and stable (commonly used as source for the current and subsequent development), and also those components being changed (potentially inconsistent).

Usually, stable software components are saved as belonging to the current global project version, and must be independently preserved from inconsistent changes. Consequently, the centralised architecture seems to be the adequate solution.

On the other hand, most of the times changes being generated by each developer are temporal (or volatile), in the sense they are not yet integrated in the global version. Therefore, for efficiency reasons, developers' workspaces should maintain, more or less independently, these temporal changes in local copies.

In conclusion and following the above guidelines, we have adopted a hybrid architecture able to support a centralised control of global versions and also local structures for developers' workspaces (see Figure 1).

An external process (see the *Global Server* in Figure 1) synchronises all the developers' actions and is also responsible for the global software project management.

All the connections between the *Global Server* and the developers' processes and between the developers' processes themselves are made using Ethernet-LAN functions (TCP/IP) or ISDN-WAN [14].

Each user starts a process on his own workplace which establishes a communication link with the *Global Server* (which is started automatically if it does not exist). After this connection is established, the *Global Server* sends all the information concerning about the current software project (if it does exist) necessary for the new developer process to register himself has one more developer in the editing session.

Each developer process, supports several structures needed to sustain locally a version of the software project. Also updated copies of the editing state of the other developers are kept, which permits to pursue a strong awareness within the group.

The *Local Server* functionalities avoid the overload of the network by handling the traffic between developer process and *Global Server*.

The architecture also allows each process on the developer workplace to connect to another workplace via a text, audio and video channels. This allows inter-user communication without the *Global Server* control and consequently promotes the integrity and effectiveness of the group task.

The video communication uses the JPEG codec standard [6] for the video frames, and VideoPix hardware or IndigoVideo on Sun or SGI workstations respectively. For audio, we provide crossplatform usability, by using a common , intermediate exchange format and realising a number of on-line bi-directional converters supporting different code formats (see [16]). Finally, text communications follows an improved version similar to Unix/Talk feature.

Project Organisation

We mean as project organisation the way its involved entities are conceptually located and the kind of relation they have.

We define *software project* as a central head and a set of *components*. The head includes the *makefile* and the data referring to the project design and conceptualisation. The *components* are pieces of text or graphics, with or without logical or hierarchical relations, consisting parts of a whole, i.e., the project structure. These *components* can be:

- **Units**: the traditional C program elements (modules, headers and libraries). They are in practice the software components of the project environment.
- **Reports**: text or graphic files manually or automatically generated, including considerations about the development task.
- **Documents**: refers to any documentation file about the software project.
- **History**: text file automatically generated, which holds a logical narrative of the developers' actions along the development process.

The *software project* environment comprehends also entities such as compilers, debuggers or any tool assisting the development process.

In fact, our considerations about consistency are strictly related with *units* and their management. They represent the "physical" *software project* being produced.

Each *unit* has one creator and one or more developers. During the development process, successive versions for each *unit* are stated stable (or consistent) and integrated in the global *software project* version located in *Global Server.*

In order to avoid interference between individual developments, the system supports for each developer workspace, local versions of the *units* being changed. Each changed *unit* becomes "visible" to the whole system, only when is integrated (if no conflicts exist) in the global version. The management of all these different versions is performed by the *Global Server* process. It keeps a set of tables controlling the state of each machine/developer.

The Physical Workspace

The system physical workspace, concerning about manipulation of files, is based on a fixed tree of disc directories. It integrates several software versions, specific tables for consistency policy, compiling or debugging and files for general purposes. At start time, this tree is automatically generated (if does not exist) on the related machine or simply updated with the last project version (if the last *Global Server* machine was different). All disc accesses are performed using a path related to the "well-known" fixed tree, and no dependent references to the machine Network File System (NFS) are considered. This strategy allows the independence of the CSD process through different NFS.

The tree structure contemplates the principle of global and local versions, by maintaining subtrees respectively for global data (*server directory*), and one for each active machine in the system (*local directories*) (see Figure 2).

Each one of the *local directories* keeps the development state concerning the related machine. It includes for each *Developer*, a subdirectorie holding his software version.

This scheme allows the *Global Server* to store independently the global version and also each *Developer* process to have locally the software state of the other users. If a *Developer* decides to compile his or another version, then uses the contends of the respective developer entry under his own correspondent machine directory.

When the *Global Server* starts on a different NFS, considering the last session, then it is automatically updated by receiving the contends of *server directory* located on the other NFS/machine. In the case of the same NFS, the updating is not needed,

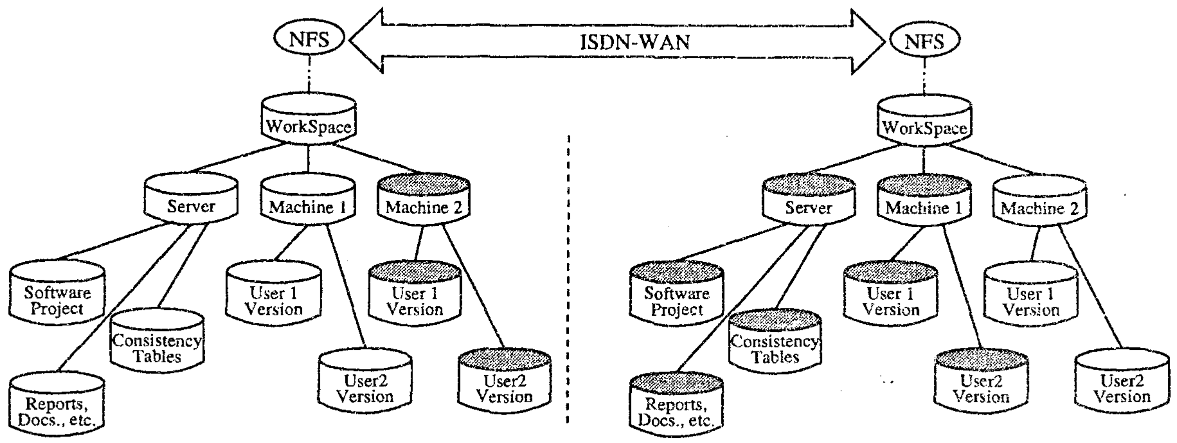


Figure 2: The physical system workspace when two Developers are active in two different machines and working in different NFS. The filled directories represent structures not used. There are two server workspaces, but only one is active, located on the machine/NFS where the first Developer started the system. Notice, that each one of the workspaces represents the structure used when two users are working in the same NFS (but in different machines).

i.e., the tree of directories is a single one visible from all machines. Only one system workspace exists per NFS.

During the *Developer* login time, the *Global Server* sends him the current global project version, even if starting on a different NFS, in order to update his local correspondent workspace. This guarantees late users to enter the work at the current development state.

2.2 The Cooperative Algorithm

It is easy to devise that different architectures adapt better to different algorithms. For example, centralised architectures adapt better to Client-Server algorithms and replicated architectures to Order algorithms.

We have adopted a hybrid solution, incorporating concepts from Client-Server (see [1]) and also Order algorithms (see [7]) and adapted to the system architecture. On one hand, there is a *Global Server* process to perform the management and on the other hand, by means of the *Local Server*, the user process gets more "intelligent" helping in the synchronisation problems.

As the development actions are performed by each of the developers, they are transmitted to the *Global Server* which re-transmits them to all the developers (including the one who originated it). There is no direct communication for development actions between the users in the system. The *Global Server* receives and dispatches the users' requests using a FIFO rule. Therefore, this guarantees mutual exclusion and serialisation of the users actions. There exists no parallelism in the answer time [9, 14].

The *Global Server* has locally a set of consistency tables containing the current development state of the whole software project. Access conflicts, updating

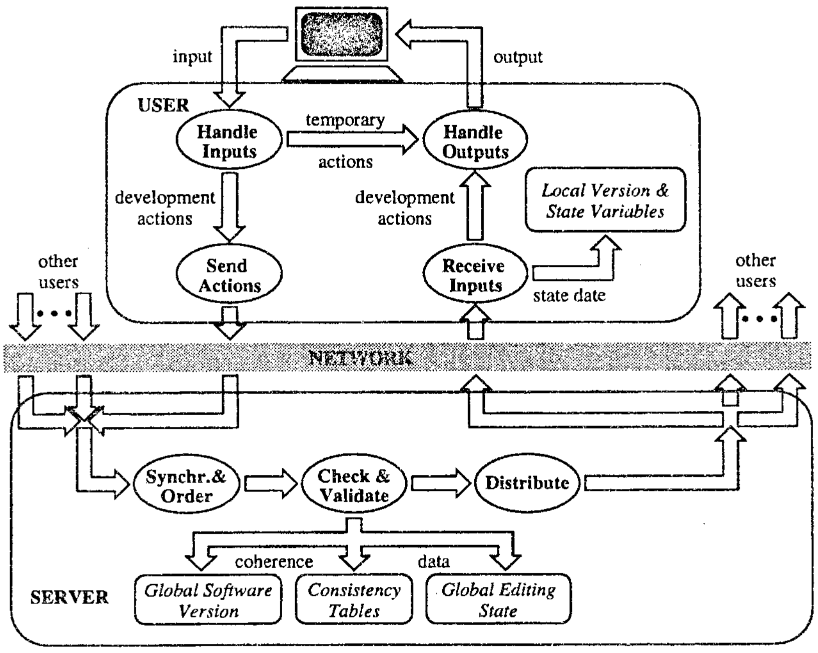


Figure 3: A global overview of the cooperative algorithm.

requirements, or general coherency violations, are checked using the consistency tables.

By receiving the actions from the *Global Server* each user process executes them either by changing the development state or by outputting a result (see Figure 3). The user actions are only definitively executed after receiving the *Global Server* answer (agreement). During the development process, the local versions are being successively updated by receiving, through the *Global Server*, the changes (stated consistent) from the other developers. Actions such as local debug or test, even passing through the *Global Server*, are always executed using the contents of the local workspace. Global test or debug demands, for purposes of coherence, the integration of all the local changes to the same global version.

The *Global Server* process is the only one with real access to the global version of the software project being developed and each one of the user processes has only a copy of it (local version). Precisely, the consistency maintenance of these copies and also of the global version, is the main goal of the algorithm.

3 Cooperative Software Development Cycle

The development of software is necessarily a set of cyclical tasks, following a common goal, i.e., the production of a "satisfactory" package of software. These set of tasks, commonly named as the Development Cycle, represent in practice the system interface to the users. The Multimedia mechanisms/metaphors play here a decisive role by enhancing the expressiveness and a best manipulation of the involved information, and consequently improve the editing environment of the Development Cycle. The media used in our system during the Development Cycle are text, graphics (raster and 2D) and audio. For the inter-group communications there are three media channels: text, audio and video.

The software development when performed by individual or small groups of developers usually follows more or less the traditional "four-steps" cycle strategy (conceptualisation, code editing (programming), debugging and testing). Moreover, we have verified this strategy adapts quite well to cooperative environments, where certainly, we need to take into account the specific requirements of the transition from individual to group work.

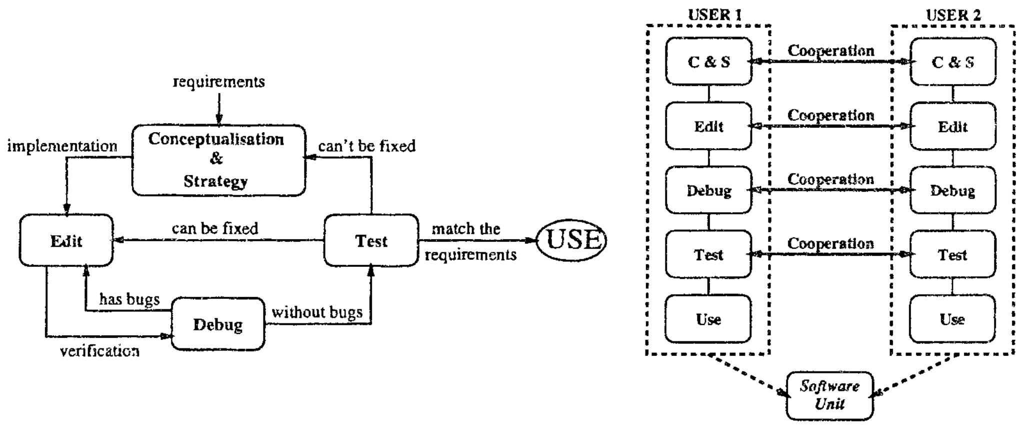


Figure 4: The *traditional* software development cycle performed by an individual developer(left), and the Cooperation role in the cycle (right).

Accordingly, we have defined a Cooperative Software Development Cycle (CSDC) as having four inter-connected phases:

- **Conceptualisation and Strategy** - the cycle's first step, where the group decides together the strategies for the project development.
- **Editing** - in this phase (we call it programming step) each developer performs his task as decided in the Conceptualisation step, and can be done alone or together. It encloses the environment of creating and changing project components with the necessary editing policy mechanisms.

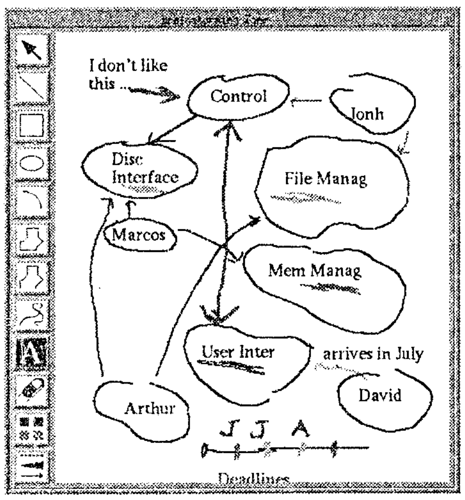
  Tasks as review or inspections, are not properly independent development steps, and consequently are included in the editing environment.
- **Debugging** - it is the common compiling and syntax error fixing task, and can be done with other developers' help.
- **Testing** - after producing executable results the developer should test them, with or without the group collaboration.

On the other hand, as the cooperative development is a task performed by a group of people, it must consequently include some kind of inter-coordination in order to promote the work performance. One response to the problem is the definition of social roles. The social roles we have adopted are Moderator - chairs the session, Developer - one of the participants in the team that actively contributes on the development process. The social roles of each of the participants in the meeting is defined at the login time or during the *Conceptualisation and Strategy* phase.

Developers can also assume another role in the system, i.e., as *Expert*. The *Expert* is any active developer in the system, who has some understanding on specific areas of the project. The developers can ask him opinions about conceptual things or send him an error report while looking for help. An *Expert* affects a special importance in the Debugging phase, where difficult syntax errors can be more easily solved with colleagues' help.

3.1 Conceptualisation and Strategy

We define *Conceptualisation and Strategy* as the first project step where the group' members decide together things as: Conceptual guidelines, global resolution strategies and tasks distribution. From here, the development responsibilities are divided among team members and each one should promote the work in his specific part. This phase represents the convergence of the whole group and consequently comes to be extremely important be supported through efficient tools for inter-group communications and group awareness.



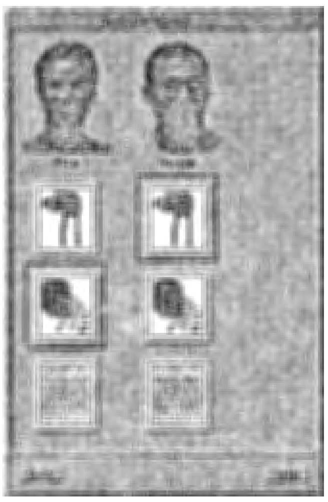


Figure 5: An example of a common drawing area (raster graphics) used for brainstorming (left) and of a window used to establish the communication channels among users in the group (right).

We provide *Conceptualisation and Strategy* in our system by a set of brainstorming tools that includes:

- a common drawing area (raster graphics) where all the users can sketch simultaneously. Each one sees instantaneously the other' inputs;
- communication channels (video, audio and text media);
- a 2D-graphic editor for drawing up more specific schemas;

In the drawing brainstorming area all the actions from other users occur with their personalised cursor, which allows the individual recognition.

3.2 Editing

As we referred before, the editing phase encloses the environment to manipulate project *components*. A *component* is a piece of text or graphics that can be edited by one or more developers.

Two specific cooperative editors (text and graphics) support the *components* editing. They include strongly the WYSIWIS and multiple cursors paradigms. The system also presents a global environment where the user can edit at same time several *components* and organise them (see Figure 7 (top-left window)).

The manipulation of *units* as software objects, represents the central goal of the editing phase, and must be considered under the global consistency strategy adopted. Indeed, only two editing modes are permitted:

turn taking - only one user can edit at a given time, i.e., the *unit* is locked.

split & combine - allows simultaneous user editing. Even so, each user has a reserved area for his own strict use and cannot interfere in the other users' parts. Anyway he is completely aware of the other actions (see Figure 6).

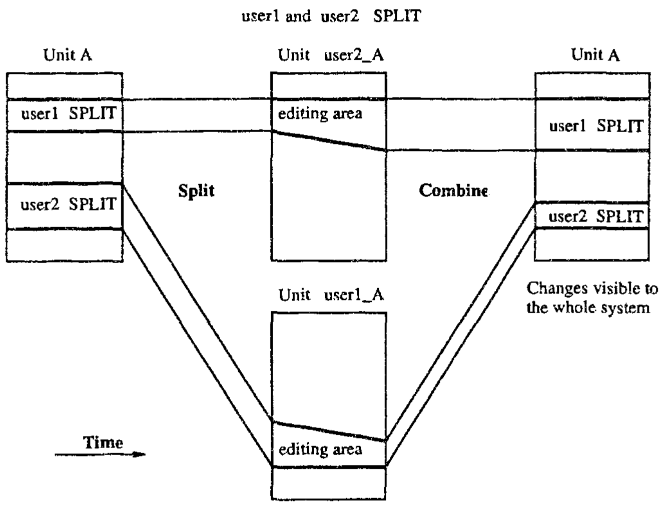


Figure 6: Creation of two alternatives units and their merge. Notice, no absolute parallel work is performed. The editing areas are always in not overlapping positions.

The *turn taking* mode concerns about the exclusive use of *units*. In fact, this mode encircles the possibility of the other users waiting indefinitely for the lock to be released. In such case, direct user-user communications can be used to find out an agreement, or as a last resource, ask for the Moderator intervention who has authority to break the lock and free the *unit*.

The *Split & combine* mode demands a merge mechanism in order to integrate the various changes. However, as no absolute parallel development exists and also no physical "collisions" are generated, consequently the merge function is reduced to a simple copy of the changed areas to the original *unit* (see Figure 6). Human intervention can always be used to repair difficult merge cases, usually coming from logical dependencies between changed parts from different users.

When the Developer leaves a whole *unit* or simply a reserved area, the related changes are sent to the *Global Server*, integrated in the global software version and finally distributed to all the other developers local versions in the system.

Code Review

Code Review is the inspection and analysis of source code *units* by developers who are knowledgeable in the application domain and programming environment. Code reviewers analyse individually the *unit* to be inspected, looking for coding errors, portability problems, violations of coding standards, etc. Thus, review is mainly based in commenting on software source code. These review comments are fragments of information referring to specific parts of the text code, and holding reviewer considerations.

Our system supports public comments and private annotations. They can contain textual, graphical and voice information. Their function is based on the traditional hypermedia paradigm - they can be accessed by following a link when clicking on the area. The comments are immediately distributed after their creation to all users. They are then common knowledge to the group and can be further edited by anyone [13]. Annotations are appropriate to privately generate ideas, which may subsequently be exposed to the group.

3.3 Debugging

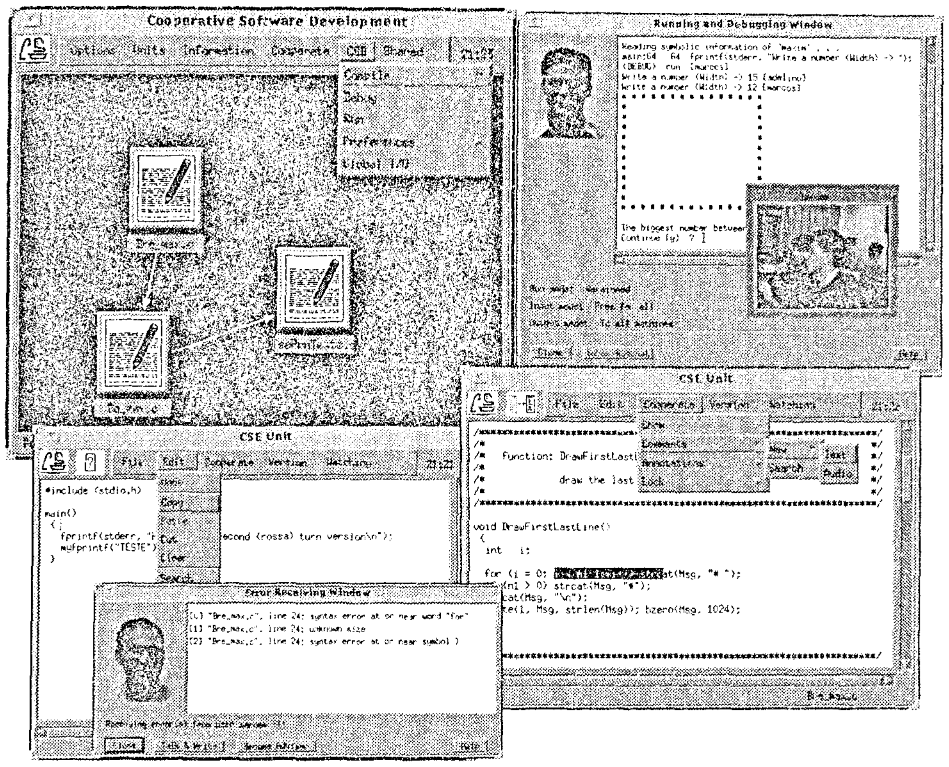
We define debug as compiling and syntax bugs fixing. In most of the cases, it is a closed individual task. The cooperation can appear in all the situations where a developer needs to ask for someone else's opinion about details such as errors or software characteristics. Our environment takes that into account by permitting developers to send errors or a whole debugging environment such as windows or *units* to an Expert. The Expert can be any of the other users. Therefore, a sub-group or all the group can follow out and help in the debugging process of a member.

The cooperation occurs also in the manipulation of a specific debugger tool (the current prototype version supports only the debugger *dbx*). Several developers can perform inputs and receive outputs in a shared interface, following the paradigm of WYSIWIS. This aims to maintain a "on line" discussion over the related local or global version being debugged.

3.4 Testing

Testing is essentially the assessment of the current results achieved on the *software project*. This process can be made locally or globally. In the first case, consists of a private test where the developer uses his local project version in way to evaluate his own changes. The global test happens when the group decides to integrate all the local versions and observe the whole aspect of the project. Then, one of the machines is adopted to hold the running process, which is usually the one of the developer who requested the global test.

Testing globally involves a shared interface and the control of the several input/outputs coming from the group. Consequently, the principle of WYSIWIS is



```
Reading symbolic information of 'maxim' . . .
main:64   64  fprintf(stderr, "Write a number (Width) -> ");
(DEBUG)  run  [marcos]
Write a number (Width) -> 15 [adelino]
Write a number (Width) -> 12 [marcos]

The biggest number betwee
Continue (y)  ? ]
```

```
/*
/*   Function: DrawFirstLastL
/*
/*             draw the last
/*

void DrawFirstLastLine()
 {
  int   i;

  for (i = 0;             at(Msg, "* ");
     (n1 > 0) strcat(Msg, "*");
     cat(Msg, "\n");
     te(1, Msg, strlen(Msg)); bzero(Msg, 1024);
```

```
#include <stdio.h>

main()
 {
   fprintf(stderr, "F          econd (rossa) turn version\n");
   myfprintf("TESTE")
 }
```

```
[0] "Bre_max.c", line 24: syntax error at or near word "for"
[1] "Bre_max.c", line 24: unknown size
[2] "Bre_max.c", line 24: syntax error at or near symbol )
```

Figure 7: An overview of the CSD system. The Units organisation environment (top left), with two open Unit editors (bottom right and left), "*Expert*" window (bottom left), the debug window (top right), and also video channel window (top right).

strictly followed, permitting as much as possible awareness within the group. The input flow can be controlled by two modes:

- *token-ring*: only one developer (who has the token) can perform input.
- *free-for-all*: all the group ' members can enter inputs.

The *token-ring* mode, refers to the classical token strategy, i.e., only one developer has the turn to perform input. The turn can be given or lost to another user.

When testing with *free-for-all* mode, all the developers can perform inputs, and the *Global Server* takes the responsibility to solve possible conflicts. In fact, a strategy of global stamps is used, i.e., each developer process has locally a total sequence order, referring to the inputs accepted by the *Global Server*. This sequence is the same in the whole system, and reflects the inputs serialisation performed by the *Global Server*. When the *Global Server* process accepts an input, it firstly informs all the developers (including the owner) which was the accepted input and who is the owner, and only after dispatches the input request.

This scheme permits to sustain coherently the sequence of input/outputs through the system.

4 Future Work

Even though the prototype offers a complete Software Development Cycle, we still have a number of research directions underway.

One of these directions that we want to pursue is the so-called version management, which permits simultaneous editing of different (time/space) versions of units and consequently a best evaluation of the whole software project. This represents one of the most important points to be completed as a future work.

On the other hand, we are already doing the first steps in what we call our priority goal - a generic multimedia cooperative environment able to integrate non-cooperative applications. It comprehends a complete multimedia computer-conferencing top level structure which allows people to share their own tools and environments and enforces a best continuity of the cooperative framework.

Another important point concerns about the support of remote software packages demonstrations and consulting. It has to combine: forms of logical representation of editing actions, multimedia, hyper-organisation of objects and transfer mechanisms over network, to enhance an interactive generation of presentation sessions. Also we have under consideration the use of ATM and "mobile" technology for communication proposes.

Finally, we want to improve the cooperation mechanisms and integrity of the system, doing it gradually able to be used as a general platform to support Software Engineering in its several tasks.

5 Conclusion

In the last years, more and powerful cooperative systems have been developed following the evolution on technology and performance of both network and multimedia workstations platforms.

One of the group' activities being supported cooperatively is Software Development. In this paper we introduce a global distributed multimedia environment to support Cooperative Software Development. An architecture and algorithm to support the CSD task, were explained, and also the Development Cycle. These have enclosed issues such as: the way the software project is organised through the distributed system, or which kind of strategy was used to perform the software development process in its several steps.

Acknowledgements

We thank Prof. Dr. J. Encarnação for the opportunities given, especially A. Santos for the useful suggestions and discussions, and finally the student J. Rossa for the help in the implementation. This work is partially funded by a CIENCIA scholarship (BD/2663/93-IA).

References

[1] Greenberg S., Roseman M., Webster D. "Human and Technical Factors of Distributed Group Drawing Tools" Proc. of the Workshop on Real Time Group Drawing and Writing Tools, CSCW'92.

[2] Harrison W., Ossher H., and Sweeney P., "Coordinating concurrent development", In Proc. of CSCW'90 (1990), ACM.

[3] Hornung Ch., Jaeger M., Santos A., Tritsch B., "Cooperative Hypermedia: an Enabling Paradigm for Cooperative Work", The Visual Computer: an International Journal of Computer Graphics, (in press).

[4] Ishii H., Kobayashi M. "Integration of Inter-personal Space and Shared Workspace: ClearBoard Design and Experiments", Proc. CSCW'92.

[5] Johnson P., Tjahjono D., "Improving Software Quality through Computer Supported Collaborative Review", Dept. of Information and Computer Sciences, University of Hawaii, January 1993 Honolulu.

[6] "JPEG Technical Specification", Joined Photographic Expert Group ISO/IEC, JTC1/SC2/WG8, CCITT SGVIII, August 1989.

[7] Lamport L. "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of the ACM, July. (78).

[8] Magnusson B., Asklund U., Minör S.,"Fine-Grained Version Control for Cooperative Software Development", Tech. Report No.LU-CS-Tr:93-112, Dept. of Computer Science, Lund University, 1993 Sweden.

[9] Marcos A. "Cooperative Editing of Static Images and 2D-Graphics in CoMEdiA", in technical report FIGD - 92i014.

[10] Marcos A., Hornung Ch., "A Global Solution to support Cooperative Software Development", as submitted to European Symposium on Programming, Edimburgh, Scotland.

[11] Narayanaswamy K., Goldman N., " *"Lazy"*Consistency: A Basis for Cooperative Software Development", in proceedings of CSCW 92.

[12] Neuwirth C., Kaufer D., Chandhok R., Morris J. "Issues in the Design of Computer Support for Co-authoring and Commenting", Proc. CSCW'90.

[13] Santos A., Tritsch B., "Using Multimedia to support Cooperative Editing", in proceedings of EUROGRAPHICS'93, Barcelona, September 1993.

[14] Santos A., Marcos A., "An Algorithm and Architecture to support Cooperative Multimedia Editing", in proceedings of 4th Workshop on Future Trends of Distributed Computing Systems, Lisbon, September 1993.

[15] Santos A., Marcos A., "CoMEdiA: UMA FERRAMENTA PARA A EDIÇÃO COOPERATIVA DE INFORMAÇÃO MULTIMÉDIA", proceedings of V Portuguese Conference on Computer Graphics, Aveiro, Feb. 1993.

[16] Tritsch B., Hornung Ch., "Cooperative Multimedia on Heterogeneous Platforms", Proc. Dagstuhl Workshop on Multimedia System Architectures and Applications, 1992.