Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Biomedical Informatics 38 (2005) 431–442
Journal of Biomedical Informatics
www.elsevier.com/locate/yjbin

Group-Slicer: A collaborative extension of 3D-Slicer

Federico Simmross-Wattenberg [a], Noemí Carranza-Herrezuelo [a], Cristina Palacios-Camarero [a], Pablo Casaseca-de-la-Higuera [a], Miguel Ángel Martín-Fernández [a], Santiago Aja-Fernández [a], Juan Ruiz-Alzola [b], Carl-Fredrik Westin [c], Carlos Alberola-López [a,*]

[a] *Laboratorio de Procesado de Imagen, ETSI Telecomunicación, Universidad de Valladolid, 47011 Valladolid, Spain*
[b] *Departamento Investigación y tecnología, Instituto Tecnológico de Canarias, 38009 Sta. Cruz de Tenerife, Spain*
[c] *Brigham and Women's Hospital Harvard Medical School, Department of Radiology, 02115 Boston, MA, USA*

Received 6 August 2004
Available online 26 March 2005

Abstract

In this paper, we describe a first step towards a collaborative extension of the well-known 3D-Slicer; this platform is nowadays used as a standalone tool for both surgical planning and medical intervention. We show how this tool can be easily modified to make it collaborative so that it may constitute an integrated environment for expertise exchange as well as a useful tool for academic purposes.

Keywords: CSCW; Telemedicine; Slicer

1. Introduction

Computer supported collaborative work (CSCW) studies how people collaborate with each other and the role that technology can play to help this collaboration succeed. Since its origin in the 1980s, computer networks have evolved to the point that, nowadays, CSCW applications can be developed in many more research areas than ever before. These advances in communication technologies have also allowed the implementation of more complex CSCW tools, so today one can find a great amount of applications based on this idea. In particular, medical informatics researchers can benefit from research findings and methodologies from the field of CSCW, in order to improve the abilities to build and deploy successful medical information systems based on collaboration [1].

The fact is, however, that although a wide range of medical CSCW applications can be found, most of them have been developed for very particular purposes and, as a consequence, they have been built from scratch. On the other hand, there are applications available for medical imaging and analysis that have been adequately tested by physicians and are routinely used in hospitals for both clinical and research purposes. Nevertheless, as of today, and to the best of our knowledge, these applications lack support for collaborative work.

This paper will show how one of these mentioned applications, the 3D-Slicer [2], can be extended to become collaborative so as to put together the advantages of a tool that is daily used by physicians and those that CSCW technologies can provide. Our extended 3D-Slicer, which we call "Group-Slicer," is still at an early stage of development but already provides collaboration facilities and a reasonable real-time operation to prospective users, as well as all the functionality that 3D-Slicer has.

* Corresponding author. Fax: +34 983 423667.
E-mail address: caralb@tel.uva.es (C. Alberola-López).

1532-0464/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jbi.2005.03.001

The paper is structured as follows: first, in Section 2, we describe the medical CSCW applications that we are aware of; as previously indicated, they seem to be developed to satisfy an identified need. On the other hand, clinically tested widespread image analysis applications have not been reported as having a collaborative module; this is what motivates us to build Group-Slicer, the technical details of which are discussed in Section 3. Section 4 is dedicated to results; first a sample execution is described; then, a quantitative analysis of traffic parameters using different protocols both in a local and a wide area network is carried out. This analysis draws direct conclusions about the real-time capability of the platform described. Section 5 concludes the paper.

2. An overview of existing collaborative medical platforms

In the recent past, several collaborative computer applications in the medical field have been reported; they all aim at creating a shared space in which physicians can interact as naturally as possible, with the ultimate purpose of reaching the best decision in every particular case. The first of these tools that we are aware of was reported in 1994, in the so-called Bermed Project [3] developed at the German Heart Institute at Berlin. This project describes an ATM-based collaborative environment for distributed patient data exchange. It also includes an audio and video channel, telepointers, and security issues.

Other applications are [4–6]. In [4], the authors describe an experimental implementation of a CSCW system built upon a PC/Windows platform, which is an example of a low-cost system suitable for adoption in a wide range of medical teleconsultation applications. The medical collaboration tool developed, "Teleworks," allows cooperative work on patient cases based on patient data folders consisting of selected diagnostic images, annotation text or patient history. The application takes care of transferring images in an encrypted form, to ensure confidentiality.

A telemedicine system is reported in [5]; this system is built on an asynchronous transfer mode (ATM) multimedia hardware/software platform comprising the following set of telemedicine services: synchronous cooperative work, high-quality video conference, multimedia mail, medical image digitization, processing, storing and printing, and local and remote transparent database access.

A collaborative extension to a platform dedicated to signal visualization and processing is presented in [6]. It allows geographically distant users to work in real time on electroencephalographic—EEG—signals. This application includes interesting functionalities, but is devoted only to (1D) signal analysis. It does not provide support for image visualization.

The results of the collaboration sustained between medical and technical teams in the development of a telemedicine platform for burn patients are presented in [7], which constitute an improvement in healthcare services for burn patients in emergency situations. With this platform, an in-the-field medical team can collaborate with an expert in burns, using information in personal data files, examination, JPEG compressed images, treatments, and messages. The platform is written in Visual Basic and runs only on Microsoft Windows. Regarding communications, the authors have developed an ad hoc protocol which allows the application to transfer images over a network. It is, however, specifically designed for burn patients.

Another teleconsultation system is reported in [8], which was specifically designed with real-time bidirectional remote control technology to meet critical teleconsultation application requirements with high-resolution and large volume medical images in a limited bandwidth network environment. With this type of consultation, both parties will be able to study the same image, manipulating synchronously on both local and remote sites including remote cursor, window/level, zoom, cine made, overlay, and measurement. To integrate the remote control communication with the medical image processing component, a medical imaging software has been created from the ground up, with communication facilities built-in. Although this method can result in a good performance since the communication protocol and the software architecture can be optimized for each other, this is a more complex option than adding communication functionality to existing medical imaging software.

There have also been developments of telemedicine systems based on the multiagent paradigm. Since telemedicine is grounded on communication and resource sharing, agents are suitable for its analysis and implementation. The most important one is perhaps the GUARDIAN system [9], where support is provided for collaboration among specialists to share data and knowledge. Another prototypical telemedicine application based upon the multiagent paradigm is reported in [10].

Also, telemedicine systems compatible with existing mobile telecommunication networks have been reported [11]. The system uses mobile telephones to transmit medical signals from sensors to a remote computer, where a doctor can remotely monitor a patient who is free to move around for sports medicine and for emergency situations.

An interesting (to our purposes) collaborative environment is reported in [12]; this platform is specialized in echocardiographic images. The author points out the need of a 3D model of the heart in order to be used as a *common artefact*, i.e., an object that allows all the physicians on-line to find their way in the set of 2D ultrasound slices; hence, the common artefact is used

here to fill the gap between the raw original echographic data and the heart in 3D.

A collaborative environment which deals with 3D data for segmentation, model creation, and visualization is reported in [13]. It is a graphical tool for collaborative image analysis and visualization of models created out of slices of volume data, and it allows a number of users to simultaneously and coordinately analyze medical images, create graphical models, navigate through them, and superimpose raw data onto the models. The application is intended to help physicians interpret data in the case that ambiguous situations may appear, by means of collaboration with other colleagues.

As of today, the need of 3D models in medical imaging is out of discussion; they not only serve to create a common ground among physicians, but they do provide an enormous amount of information both for diagnosis, planning, and intervention. Efforts in creating 3D graphical tools for advanced visualization and data manipulation which behave reliably even inside the operating room are numerous. Some of them are 3D-Slicer [2] from SPL [14], ANALYZE [15], MEDx [16], and MNI [17], just to mention a few. These applications incorporate powerful tools for segmentation, registration, and quantitative data analysis.

The majority of the applications described above have been created from the ground up and most of them are made for specific purposes and may not handle 3D models. Even though some attempts have been made to create collaborative environments that can deal with 3D data for segmentation, model creation, and visualization [13], it is clear that creating new tools from scratch to make them collaborative constitutes a non-worthtaking effort, since clinically tested tools are out there (some of them mentioned in the previous paragraph) and they could be extended—with a moderated programming effort—to work collaboratively. This is the approach we have chosen in this paper; specifically, we have taken the first step towards a collaborative extension of 3D-Slicer.

3D-Slicer is an open-source application for medical data processing. Its basic capabilities include visualization, registration, segmentation, and quantification of such data [18]. More importantly, 3D-Slicer has been tested by physicians at Brigham and Women's Hospital in real situations, it is routinely used for clinical practice, and can be considered as a standard de facto on its field. One of its main uses is to build 3D graphical models offline, so that these models are available in the operating room for the surgery process. In order for these models to be built, subtle details need to be segmented with human interaction. Many of these details need a second opinion based on previous experience. This is when collaboration plays a crucial role. In addition, once in the operating room the 3D model of the patient may need adaptation to the actual conditions. This is done through involved registration algorithms [19]; an external opinion is of great interest to evaluate the accuracy of these registration algorithms for the particular patient undergoing surgery. This is another issue concerning collaboration. Due to the specificity of all these procedures, it is clear that using the same tools, with no changes, and make them collaborative, is of paramount importance.

3. Design issues

3.1. Communications platform

In a previous work called diSNei [13], we developed a collaborative environment capable of real-time interaction over a network. This environment used a toolkit, called *Groupkit* [20], as a platform to support communications and session management. There were a number of reasons for which Groupkit was preferable with respect to other middleware solutions (see [21] for further information) but mainly, diSNei was implemented using the Tcl/Tk language, and Groupkit did exactly what was needed in Tcl/Tk, so it suited the authors' needs perfectly.

The problem was how to port the work and ideas from diSNei to 3D-Slicer. The 3D-Slicer is based (as diSNei was) on Vtk [22], a graphical library which provides a high-level interface to OpenGL and a pipeline mechanism to connect graphical filters. This library is implemented in C++, but provides a Tcl wrapper to instantiate and execute its methods. The rest of 3D-Slicer, user interface and event handling, is Tcl/Tk.

At this point, a straightforward choice was to use Groupkit for Slicer as well; however, the development of such a toolkit had stopped in 1999 and recent versions of 3D-Slicer, in constant growth, no longer worked with it. Some choices were considered to work around this situation, and eventually a decision was made in terms of developing a collaborative platform to support our particular needs without any other external dependencies. This platform (see Appendix A) basically does what Groupkit used to do (serve as a collaborative framework to Tcl/Tk applications), but focuses more on lower-level aspects of communications than Groupkit. This, in turn, allows us for a better control of the collaboration as well as to provide a high feeling of interactivity while still being as easy to use as Groupkit was. As a matter of fact, our collaborative platform provides an application programming interface (API) to Tcl, but has on the other hand been developed in C++, the same approach as Vtk uses.

3.2. Network architecture

The API[1] is based on the client/server approach. This decision was made for the simplicity these systems

[1] The functionality that will be described here is reflected in the functions listed in Appendix A.

provide over peer to peer networks, as well as for the need to maintain some data in a centralized repository. This scheme allows us to have an authority (the server) decide on potentially conflicting situations. Fig. 1 illustrates how virtual and real communications are carried out with this paradigm.

The server deals with IP addresses of clients, provides authentication mechanisms (see Section 3.3), and takes care of delivering messages to their destinations. Clients do not need to know about other machines but the server; they just know the users who have logged into the session and ask the server to send messages to one or to all of them.

To assist Tcl applications communicate with the server, a client library has been developed; this library provides interface functions that can be called directly from Tcl, thus hiding all the details of communications but the essential: users and messages. Only at session initialization clients must know about the server's IP address and port.

Overall, our collaborative platform consists of a server process that manages communications among clients and a client library, which acts as a bridge between the user application (3D-Slicer in this case) and the server. As it has been designed independently from Slicer, the platform can potentially be used to extend any other Tcl/Tk application to be collaborative.

Pertaining to network connections, all communications are done at socket level, but are conveniently encapsulated in C++ classes which provide an object-oriented sockets layer, as shown in Fig. 2. This way, a network connection can be thought of as an object which can be created, used, and destroyed almost the same way as a C++ stream (see Appendix B for an illustrative example). With this approach, we can transparently add optimizations (such as buffering) and error handling to the socket layer without altering higher levels of the platform; for instance, this layer allows us to easily switch between connection- and non-connection-oriented sockets (or even between Internet and Unix sockets if desired) with minimum changes to the source code. Indeed, we have working versions of our API which work over TCP and UDP, or through a SSH tunnel.

Bandwidth utilization is kept to a minimum, because large data transmissions are avoided while

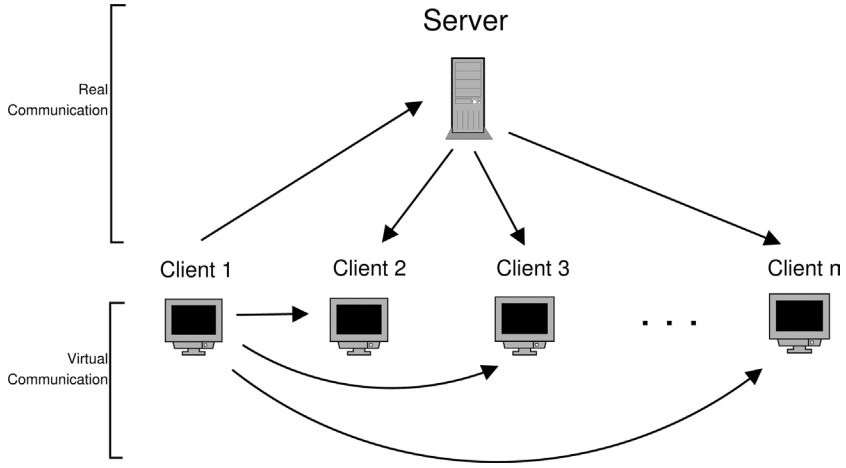

Fig. 1. Network architecture of the API. Clients virtually communicate with each other but real communication flows through the server.

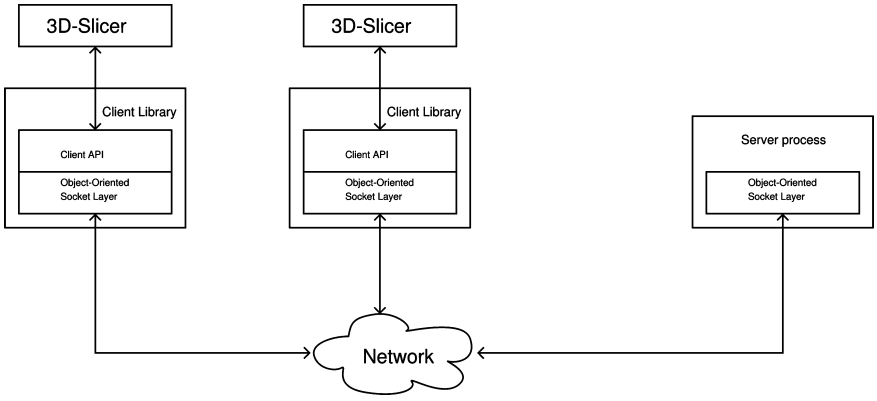

Fig. 2. Block diagram showing the client and server parts of our collaborative platform.

collaborating. Instead of sending rendered graphics over the network, which would certainly downgrade interactivity, events and user actions are captured and sent as messages to other clients which, in turn, react to the received events as if they had occurred on the local machine. See Section 4.2 for further details on this issue.

While initializing the session, a listening port is configured for use by both the server and each client (ports do not need to coincide) and they are thereafter used for the whole session. This serves the purpose of improving security (see Section 3.3) and it also makes it easier to configure a firewall if that is desired. Simply opening the configured port(s) for listening and enabling outbound connections is sufficient for the platform to work.

3.3. Authentication and user management

Users start collaborating by logging into a session. A user name and a password must be provided to the server, which is responsible for allowing or rejecting the user into the session. Usernames and passwords are stored in a XML file at the server machine, so as to remove dependencies with external database systems. Fig. 3 shows an example of users file. Mandatory fields are just 'login' and 'password' for each user, but any other information, such as the user full name (as shown in the example) can be stored in this file. If this stage is cleared, the user is assigned an identifier and is considered as registered in the session. This strategy is commonly used by web servers in maintaining private sessions for their users.

Obviously, this information must be known only by a unique authority that manages permissions to enter a collaboration session, hence the need for a centralized server. A peer to peer collaboration system would have needed, at least, any sort of shared storage which would have turned into a client/server system again.

```
<?xml version="1.0" encoding="UTF-8" ?>
<CKusers>
    <user>
        <login>crix</login>
        <passwd>mysecret</passwd>
        <name>Cristina Perez</name>
    </user>
    <user>
        <login>noemi</login>
        <passwd>anypass</passwd>
        <name>Noemi Garcia</name>
    </user>
</CKusers>
```

Fig. 3. A sample users file showing two users, namely, crix and noemi, along with their respective login names, passwords, and full names.

During the authentication process, clients store the IP address and the port where the authentication response came from. Similarly, the server keeps the original IP addresses of the clients as well as their assigned session identifiers. This is done as a step towards security: in any subsequent communication between the server and any client, the listening side (be it the server or the client) checks where network packets come from, and they are discarded if they do not match the stored IP address and session identifier (and port in its case). This has the disadvantage that a single machine can host one user at a time but, due to the nature of 3D-Slicer's user interface, it is unlikely that two or more users are willing to collaborate from the same machine.

Finally, we have not paid attention to encryption procedures; if this is an issue, communications may be tunneled through an encrypted protocol, such as SSH, which we can handle easily and with a moderately low increment in network load (see Table 1).

3.4. Implementation details

We wanted the platform to work in an easy way: the user in charge performs an action, and the results of the action have echoes both on the local computer and on the remote computer.[2] To achieve this, the straightforward approach is to duplicate the code lines that implement each specific action in order to implement the same action on the remote computer. By doing this, however, there would be a need of severe modifications in many files in the source tree. Fig. 4 shows the directory structure of the current public-domain freeware version of 3D-Slicer; most of the files located within folders with the *Tcl-* prefix of 3D-Slicer should be modified if this approach was taken, and thus, it would be impossible to extend this work to future revisions of 3D-Slicer because of the need to keep so many files up to date.

So one of the leading objectives of this collaborative extension to 3D-Slicer is that the original application should be minimally changed. Another decision made in the development of this extension was to keep the application independent from the added collaborative facilities. Accordingly, the extension has its own file structure, which can be plugged into the existing directory tree of 3D-Slicer, as shown in Fig. 4. The collaborative module is located in the new folder named *Tcl-cscw*, which contains all the functionality for communication, and the files with the necessary procedures to make 3D-Slicer functions collaborative.

[2] Our current version deals with only two users. However, this behavior can be easily changed to allow an arbitrary number of computers to collaborate.

Table 1
Measured values for various underlying protocols and networks, for 30 s of continuous movement on the Slicer's viewer

	Total packets	Avg. packet size (bytes)	Bytes/s	Total bytes	Round-trip delay (μs)	
					x_{50}	x_{95}
LAN, UDP	657	87.909	1941.514	57,756	590	753
LAN, TCP	3,455	64.355	7395.798	222,346	725	1136.9
LAN, SSH	2,325	102.135	8015.868	237,464	751	1825.6
WAN, SSH	817	92.840	2570.146	75,850	262,244	265,800
LAN, VNC	18,548	1139.009	747302.014	21,126,340	n/a	n/a

Values for VNC are included solely as a reference, as it uses other ways to communicate.

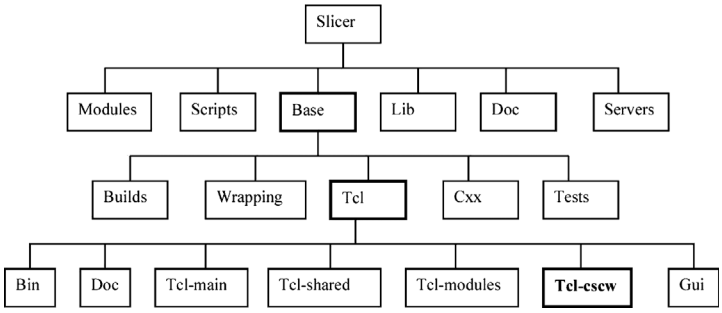

Fig. 4. Directory structure of 3D-Slicer. Highlighted boxes are those altered by our application. Note that folder *Tcl-cscw* is new (i.e., it is not included in the public-domain version of 3D-Slicer).

Instead of duplicating the code to perform actions on the remote computer, calls to functions have been written just below the code line that implements this same function locally. This is the main modification of the original code. The procedures for these remote functions are located inside the folder *Tcl-cscw*, so the original files of 3D-Slicer are only slightly modified with a call to another function. In these procedures, it will be checked whether there is a collaborative work or not, and in the case there is, all the actions made locally will also be executed remotely. An example of how the platform has been carried out is shown in Fig. 5.

As a result of this approach, the way 3D-Slicer works remains unchanged, and only some extra code has been added to the original one to integrate the collaborative module. The main advantage of this decision is the simplicity in the development of the extension and, moreover, physicians who use 3D-Slicer directly know how to use Group-Slicer; they do not have to learn how to use another application because it stays exactly the same.

With regard to the data used by the physicians, the application is designed bearing in mind that prospective collaborators have direct access to the same files; so our effort has been focused on authentication issues, as well as on communication issues and functionality.

Collaboration consists of having on screen the same information as the other collaborators. In order for actions to be coordinated, only one user (which has been termed *the one in charge* above) in the collaboration has access to the datasets and the GUI, while the others are only allowed to see the results of the actions of that user. These results do not have an appreciable delay, because only some lines of code are sent to the remote computer and the rendering of the scene is done in parallel at every machine. So, although real-time capabilities cannot be guaranteed, we can state that a sense of real-time operation is provided to the users.[3] An additional chat facility has also been included so that users can communicate by means of text messages at any time.

Any user has the possibility to ask for permission to get control of the application; we call it "asking for the token." When this happens, the user currently in possession of the token sees a flashing window on the screen and can either choose to keep the token or to release it. If the latter happens, the token goes to the user that asked for it, who is thereafter in charge.

As for the GUI itself (see Fig. 6), the original 3D-Slicer GUI has remained unchanged, and with this collaborative module 3D-Slicer can be used either collaboratively or individually. To add the collaborative extension, only one more window is needed. When users want to work collaboratively, they simply have to log in; if the log-in process is successful, the buttons needed for collaborative work will be added in this additional window.

[3] Section 4.2 shows quantitative values of this statement. A guarantee, however, cannot be given since communications depend on the network load at any time.

A

```
#-----------------------------------------
# ChangeLabel->Apply frame
#-----------------------------------------
set f $Ed(EdChangeLabel,frame).fApply
eval {button $f.bApply -text "Apply"
        -command "EdChangeLabelApply"}
        $Gui(WBA) {-width 8}
eval {label $f.lApply -text "Also apply by clicking on a label."}
...
```

B

```
#-----------------------------------------
# ChangeLabel->Apply frame
#-----------------------------------------
set f $Ed(EdChangeLabel,frame).fApply
eval {button $f.bApply -text "Apply"
        -command "EdChangeLabelApply;EdChangeLabelApplyRemote"}
        $Gui(WBA) {-width 8}
eval {label $f.lApply -text "Also apply by clicking on a label."}
...
```

Fig. 5. (A) Original code of 3D-Slicer which carries out an "EdChangeLabelApply" action. (B) The same code, modified to carry out the action remotely as well.

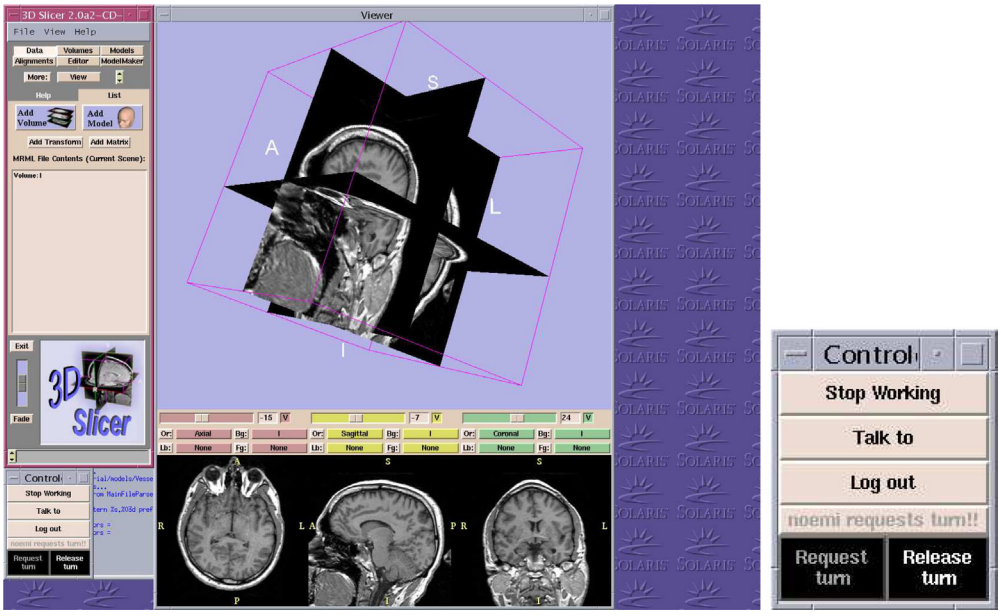

Fig. 6. GUI of 3D-Slicer. The window on the bottom left corner of the screen manages the token exchange procedures. On the right, a magnified view of this window.

4. Results

4.1. A sample execution

Fig. 6 shows the GUI of Group-Slicer; readers familiar with 3D-Slicer may soon notice that the GUI is the same of 3D-Slicer but for the small window in the lower left corner of the figure. This is the control window of the collaboration. In this figure we represent the control window seen by the user in possession of the token (the one whose "turn" is to control). At any time this user may release the token (see that button "release turn" is active). Other users will have this button inactive, while the button "request turn" will be active. In this figure only two users are collaborating. The user with the token may know that user *Noemi* is asking for the token because, in that case, the message window "Noemi requests turn" would flash. In addition, collaboration

Fig. 7. (A) A mouse click on the computer on the right. The user starts dragging a slider to move the cutting plane. (B) Effect of the drag action: displays always remain coherent and the transition between both instants is smooth on both computers.

may finish at any time by clicking on the button "Stop Working" and the chat window will pop by just pressing "Talk to."

Fig. 7 shows two snapshots of a sample execution. In particular, as for Fig. 7A the user on the right has just clicked at one of the sliders that select which slices are represented in the 3D model. Note that no one is using the computer on the left. Fig. 7B shows a second snapshot immediately after the user has dragged the aforementioned slider to some other point in the scale. Both computers update their user interface smoothly so that they are always coherent. Obviously, these snapshots do not show the transition between Figs. 7A and B but, as stated, it feels as smooth on the remote computer (left) as on the local one (right).

4.2. Measurements

It is commonly accepted in medical environments that collaboration generally improves diagnostics. However, it is not an easy task to measure the goodness of a collaborative application, since it heavily depends on subjective feelings of each user. In any case, it is clear that user satisfaction (in terms of action-response delay) affects the eventual success of a collaborative application. This is why we have focused on measuring quantitative traffic magnitudes as an indirect indicator of prospective success.

The specific magnitudes are the following: number of transmitted packets, average packet size, transmitted bytes per second, total number of transmitted by-

tes, and round-trip delay, the latter measured as the time between a query of the client and the response from the server reaching the client. For this kind of collaborative application, in which responsiveness and user interaction is a must, we propose these magnitudes to be measured at a worst-case working status (i.e., when network demands are maximum); so we have defined a 30-s interval in which one of the users is continuously rotating a graphical model in one of the clients. Measurements are taken within such an interval. As for the delay, we have observed a non-Gaussian distribution; therefore we report 50 and 95 percentiles.

Table 1 summarizes these values in four different cases. The first three (labeled with LAN) were measured on a 100 Mbps local area network (the two clients and the server were physically located in our lab in Valladolid, Spain) whereas the fourth one (labeled with WAN) shows a case in which the two clients were located in Valladolid and the server was located at Brigham and Women's Hospital Harvard Medical School (USA). All measurements, except for round-trip delay, were done with Ethereal [23].

As a reference, the same values were measured (whenever possible) using the well-known VNC application [24], which lets one or more users see a remote screen on their local desktops; this is not an actual collaborative application, but it may be used as such. We used the xf4vnc variant [25] because it includes OpenGL support, which is necessary to run Slicer.

The numbers shown in Table 1 correspond to the traffic exchanged between one client and the server for a period of 30 s of continuous movement of a 3D graphical model. At a glance, it is clear that UDP provides the best figures for interactivity. However, due to the nature of this protocol, it should only be used when packet losses are very unlikely to occur, such as on a dedicated line or a local area network; as a matter of fact, the figures for UDP indicate that collaboration is perfectly possible even through a 56k modem line (convert bytes/s to bits/s by multiplying by 8 in the table). On the other hand, TCP guarantees the delivery of all network traffic and, though its figures are worse than UDP (they are a bit tight for a modem line) this can hardly affect the user if the network is able to handle the extra amount of traffic (keep in mind that round-trip values are in microseconds). Note that the cause for the lower average packet size is TCP handshaking, which exchanges very short packets at every connection establishment. This also explains why the number of transmitted packets is so high compared to UDP.

The SSH protocol (which in turn works over TCP) can be used to tunnel network connections so as to pass through firewalls without the intervention of the administrator. Although our platform uses just one port for each client machine (plus one more for the server), it is not always possible or desirable to open up these ports, so we included the possibility to use SSH as a transport method. Table 1 shows that there is some overhead compared to plain TCP, mostly due to the encryption that SSH provides, but it remains perfectly usable if the network's bandwidth is enough to handle it. In this case, there are fewer transmitted packets (albeit larger) because SSH uses just one previously established connection, so there are not continuous handshakings. Also note the figures for SSH through a WAN (Spain–USA): there are fewer transmitted packets because we are measuring for the same period of 30 s and the roundtrips are several orders of magnitude higher than in a LAN. This, of course, results in a somewhat poorer sense of interactivity when compared to collaborating in a LAN (where, figures show, round-trip delays are totally unnoticeable for a human being) but the application remains usable nevertheless.

Finally, we compared our platform to VNC because it has proved very useful to work on remote computers, and it is sometimes used as a way to collaborate. VNC uses a different approach to show the contents of the screen on remote computers, which results in very large figures on Table 1. VNC is ideal for some tasks, but certainly not when large portions of the screen change very quickly in time, which is the case of Slicer.

5. Conclusions

In this paper, we have described a simple procedure to build a collaborative extension of the well-known *3D-Slicer* graphical tool. This extension is built upon the current public-domain version of the Slicer and actions are replicated in the remote computers by just sending the commands over the network, a design criterion which poses low bandwidth requirements. In addition, we have taken an approach that allows us to easily extend 3D-Slicer while modifying a minimum number of files in the original distribution.

Communications are taken care by means of an API we have also created. Quantitative measurements both on a LAN and a WAN indicate that the application allows for interactivity between users, a fact that is mandatory for collaboration.

Even though the application is still in its infancy and an effort is needed to validate it in terms of usability, we believe the application could be of great value in different areas of both clinical practice and academia.

Acknowledgments

The authors acknowledge the Comisión Interministerial de Ciencia y Tecnología for Research Grant TIC2001-3808-C02, to NIH Grant P41-RR13218 and CIMIT, and to the European Commission for the funds associated to the Network of Excellence SIMILAR (FP6-507609). The authors thank Dr. Raúl San José for his help, patience, and invaluable support at testing our platform between Valladolid and Boston, and Prof. Asensio for his useful comments in the revised version of this paper.

Appendix A. Application programming interface

We have built our own API to support Group-Slicer since Groupkit, a Tcl-based collaborative toolkit on which we built [13], has not been updated for quite a long time, so it would not work with the version of Tcl/Tk currently used by 3D-Slicer.[4] The API has been written in C/C++ for its network programming flexibility and power. For easier utilization of the sockets layer, the API uses our own Socket extension for C++, which makes them behave mostly as a standard buffered C++ stream (see Appendix B).

This appendix shows the main function calls of the API that we used when extending 3D-Slicer. Note that these few functions suffice to provide collaboration while keeping changes to Slicer to a minimum.

A.1. API function calls

The communication API was designed so that high-level primitives are as easy to use as possible, while still maintaining the efficiency that C/C++ can provide. Client function calls are invoked from the Tcl script which uses the API. Before starting to send and receive messages over the network, one must call the 'ck::init' function to initialize the library, and then register to the server via the 'ck::reg' command.

A.1.1. ck::init

This function initializes the communications with the server. It must be called before any other function.

SYNTAX: ck::init <server_host> [server_port [client_port]]

A.1.2. ck::register

This function registers the user in the current session. A valid login and password must be provided. This function must be called before messages can be sent or received, but after calling ck::init.

SYNTAX: ck::register <login> <password>

A.1.3. ck::unregister

This function unregisters the user from the current session. To start communicating again, a call to 'ck::register' must be made.

SYNTAX: ck::unregister

A.1.4. ck::users

This is meant to be a multi-purpose function which returns miscellaneous information about other users. As of today, only one subcommand is implemented, which returns a list containing the logins of the currently registered users.

SYNTAX: ck::users list

A.1.5. ck::to

SYNTAX: ck::to <destination> <Tcl_command> ...

This function is used to send commands to be executed remotely. This command is extensively used in the application and, in particular, in the example shown in Fig. 5.

Appendix B. C++ extension for socket programming

As stated before, our API utilizes our own sockets extensions to take advantage of object-oriented programming. Mainly, this means that one can think of a socket as an object similar to C++ Input/Output streams: A "send" or "receive" operation becomes a simple use of the operators << or >>. This greatly simplifies the use of inter-process communications in C++, as the underlying complexities (be it the network or the local filesystem) are removed. In addition, all communications are automatically buffered on input, so as to avoid worrying about the efficiency of making lots of system calls to retrieve information.

B.1. A simple example

For the sake of comparison, we show an example of a chargen client, coded with and without our Socket Extensions; on the left, a traditional TCP chargen client. On the right, our Socket Extension.

[4] The University of Calgary stopped Groupkit development in 1999. In March 2003, a new version of Groupkit was released independently, which can be downloaded from http://www.groupkit.org. At that time, our API was very much accomplished and, moreover, no new versions of Groupkit have appeared since then, to the best of our knowledge.

```
#include<netdb.h>
#include<unistd.h>
#include<iostream>
#include<sys/socket.h>

main() {
    struct hostent* remoteEnt;
    struct in_addr remoteIp;
    if((remoteEnt=::gethostbyname("myhost"))==0) {
        std::cerr<<"Socket error: "<<hstrerror(h_errno)<<'\n';
        exit(1);
    }
    remoteIp.s_addr=*((uint32_t*)remoteEnt->h_addr_list[0]);

    int s;
    if((s=socket(AF_INET,SOCK_STREAM,0))==-1) {
        perror("Socket error");
        exit(1);
    }

    struct sockaddr_in remoteAddr;
    remoteAddr.sin_family=AF_INET;
    remoteAddr.sin_port=htons(19);
    remoteAddr.sin_addr.s_addr=remoteIp.s_addr;
    if(connect(s,(struct sockaddr*)&remoteAddr,sizeof(remoteAddr))==-1) {
        perror("Socket error");
        exit(1);
    }

    char a;
    for(int i=0;i<1000000;i++) {
        if(read(s,&a,1)==-1) {
            perror("Socket error");
            exit(1);
        }
        std::cout<<a;
    }
}
```

```
#include<iostream>
#include<sock.h>

main() {
    try {
        inetStreamCli c("myhost",19);

        char a;
        for(int i=0;i<1000000;i++) {
            a=c.get();
            std::cout<<a;
        }
    }
    catch(sockException e) {
        std::cerr<<"Socket error: "<<e.what();
        exit(1);
    }
}
```

There is a clear gain of simplicity but, in addition, this variant is much more efficient due to the internal receive buffer, which greatly reduces the number of system calls made. Our tests show a mean execution time of 5.9189 s for the traditional version (0.0959 std. deviation, 102 samples) and 2.0667 s for the extended version (0.1032 std. deviation, 102 samples). While this can obviously be achieved with a more complex traditional program, we want to emphasize that the extended version is much simpler while its performance is not affected at all.

References

[1] Pratt W, Reddy MC, McDonald DW, Tarczy-Hornoch P, Gennari JH. Incorporation ideas from computer-supported cooperative work. J. Biomed. Inform. 2004:128–37.

[2] Gering DT, Nabavi A, Kikinis R, Grimson WEL, Hata N, Everett P, Jolesz FA, Wells III WM. An integrated visualization system for surgical planning and guidance using image fusion and interventional imaging. In: Taylor CJ, Colchester A, editors. Medical image computing and computer-assisted interventions. Lecture notes in computer science, vol. 1679. Berlin, Heidelberg, New York: Springer-Verlag; 1999. p. 809–19.

[3] Kleinholz L, Ohly M. Supporting cooperative medicine: the bermed project. IEEE Multimedia Mag. 1994:44–53.

[4] Makris L, Kamilatos I, Kopsacheilis EV, Strintzis MG. Teleworks: A CSCW application for remote medical diagnosis support and teleconsultation. IEEE Trans. Inform. Technol. Biomed. 1998;2:62–73.

[5] Gómez EJ, del Pozo F, Ortiz EJ, Malpica N, Rahms H. A broadband multimedia collaborative system for advanced teleradiology and medical imaging diagnosis. IEEE Trans. Inform. Technol. Biomed. 1998;2:146–55.

[6] Bouillon Y, Wendling F, Bartolomei F. Computer-Supported Collaborative Work (CSCW) in biomedical signal visualization and processing. IEEE Trans. Inform. Technol. Biomed. 1999;3:28–31.

[7] Reina-Tosina J, Roa L, Rovayo M. NEWBET: telemedicine platform for burn patients. IEEE Trans. Inform. Technol. Biomed. 2000;4:173–7.

[8] Zhang J, Stahl JN, Huang HK, Zhou X, Lou SL, Song KS. Real-time teleconsultation with high-resolution and large volume medical images for collaborative healthcare. IEEE Trans. Inform. Technol. Biomed. 2000;4:178–85.

[9] Hayes-Roth B, Larsson JE. A domain-specific software architecture for a class of intelligent patient monitoring systems. J. Exp. Theor. Artif. Intell. 1996;8:149–71.

[10] Mella V. Agents acting and moving in healthcare scenario—a paradigm for telemedical collaboration. IEEE Trans. Inform. Technol. Biomed. 2001;5:11–3.

[11] Woodward B, Istepanian RSH, Richards CI. Design of a telemedicine system using a mobile telephone. IEEE Trans. Inform. Technol. Biomed. 2001;5:13–5.

[12] Berlage T. Augmented-reality communication for diagnostic tasks in cardiology. IEEE Trans. Inform. Technol. Biomed. 1998;2:169–73.

[13] Alberola-López C, Cárdenes R, Martín M, Martín MA, Rodríguez MA, Ruiz-Alzola J. diSNei: A collaborative environment for medical images analysis and visualization. In: Anthony A. DiGioia, Scott Delp, editors. Medical image computing and computer-assisted interventions. Lecture notes in computer science, vol. 1935. Berlin, Heidelberg, New York: Springer-Verlag; 2000. p. 814–23.

[14] Brigham and Women's Hospital and Harvard Medical School. Surgical Planning Lab. Available from: http://www.splweb.bwh.harvard.edu:8000.

[15] Mayo Clinic. ANALYZE software. Available from: http://www.mayo.edu/bir/Software/Analyze/Analyze.html.

[16] Medical Numerics Inc. MEDx software. Available from: http://medx.sensor.com/products/medx/index.html.

[17] Montreal Neurological Institute. MNI software. Available from: http://www.bic.mni.mcgill.ca/software.

[18] 3D Slicer. 3D Slicer software. Available from: http://www.slicer.org.

[19] Maintz JBA, Viergever MA. A survey of medical image registration. Med. Image Anal. 1998;2:1–36.
[20] GroupLab. GroupKit: A Groupware Toolkit. University of Calgary. Available from: http://grouplab.cpsc.ucalgary.ca/projects/GroupKit.html.
[21] Toribios MA, de la Fuente JC, Dimiatriadis Y, Alberola-López C, Analysis and development of CSCW systems using toolkits. The case of collaborative diagnostics with echographical sections [in Spanish]. Actas de las Jornadas de Ingeniería Telemática, Jitel 2001, Barcelona, Septiembre 2001. p. 495–502.
[22] Kitware Inc. VTK, The Visualization Toolkit. Available from: http://www.vtk.org.
[23] Combs G, et al. Ethereal. Available from: www.ethereal.com.
[24] RealVNC. Virtual Network Computing. Available from: http://www.realvnc.com.
[25] Alan Hourihane. xf4vnc. Available from: http://xf4vnc.sourceforge.net.