VCRI: A groupware application for CSCL research

Jos Jaspers (j.jaspers@fss.uu.nl)
Marcel Broeken (m.h.broeken@fss.uu.nl)

Educational Sciences
Utrecht University
The Netherlands

Paper presented at the European Tcl/Tk User Meeting
Bergisch Gladbach (Germany), 27-28 May 2005

Abstract

One of the main research topics of the Educational Sciences group at Utrecht University is Computer Supported Collaborative Learning (CSCL). The CRoCiCL* project[1] is a research project, which focuses on CSCL and the effects of visualization of social aspects of collaboration processes in CSCL. The project started in September 2003 and will take about four years to complete. We have developed a groupware environment called VCRI, which enables users to collaborate and communicate. In this paper we will give an overview of the development history of the VCRI, its features, problems we encountered, our plans for the future and of course Tcl's part in all this.

Introduction

Secondary school students in The Netherlands – as a result of recent changes in the curriculum of the final years (the 'study house') – are doing increasingly independent research in preparation for college studies. The focus has shifted towards working actively, constructively and collaboratively, as this is believed to enhance learning. We have developed a groupware computer environment that supports these research activities that should fit well within this curriculum. The purpose of our research is to investigate the effect of the computer supported research environment and its tools on the final product through differences in the participants' collaboration processes.

* The CRoCiCL project (Computerized Representation of Coordination in Collaborative Learning) is funded by N.W.O. , the Dutch Organization for Scientific Research, under project number 411-02-121.

VCRI overview

The VCRI (Virtual Collaborative Research Institute) is a client-server based groupware environment providing a customizable tool-set. Currently there are about fourteen tools, ranging from a collaborative text processor (Co-Writer) to an instant messaging client (Chat). Any subset of tools can be used, to give users true flexibility. Adding new (third-party) tools is currently not supported but we are working on providing a clear framework for developers to make this possible.

One of the key ideas behind the VCRI is WYSIWIS (What You See Is What I See). Users share most tools. A shared tool is continuously synchronized and looks the same to every user. All users can edit the content of the tool, for some tools even simultaneously. The VCRI server is in charge of synching all tools. This mechanism of sharing gives the impression of real-life collaboration, even in cyberspace.

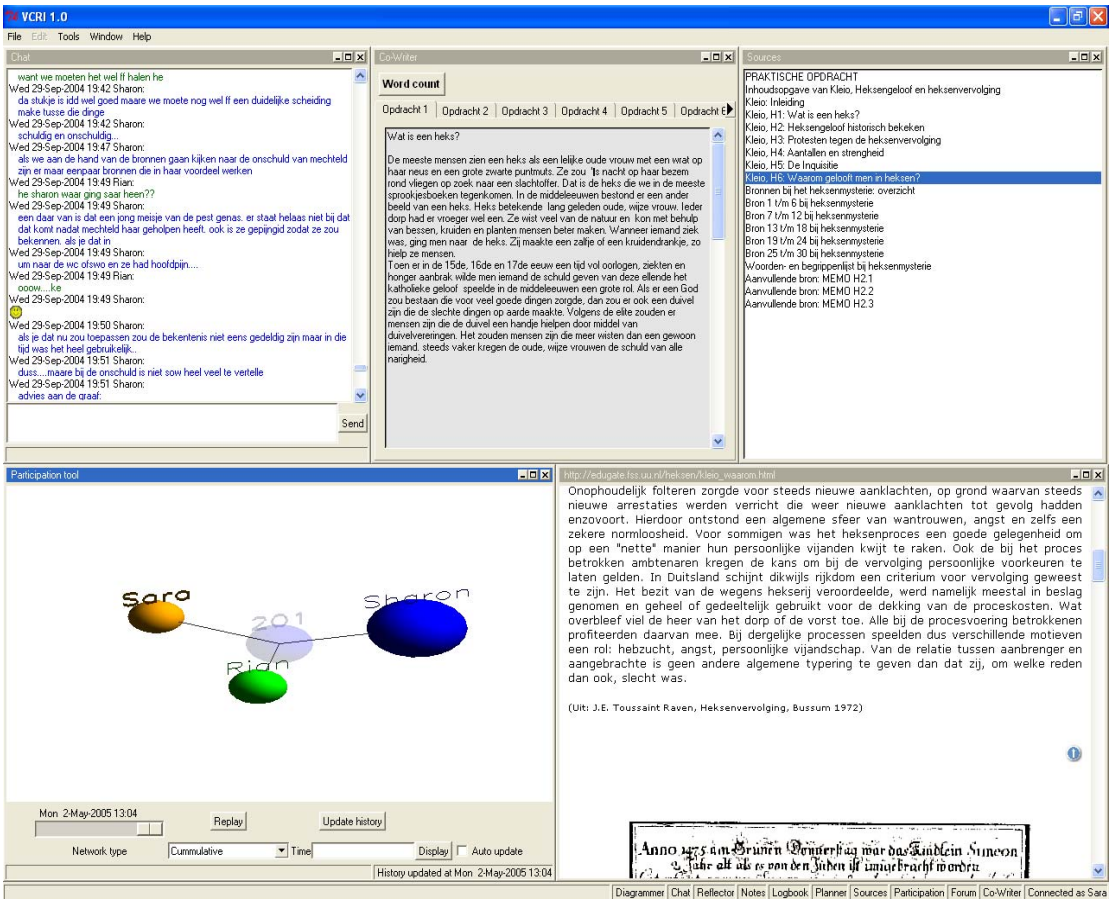

Figure 1 : The VCRI environment

Figure 1 shows a screenshot of the VCRI with some of tools. From left to right and from top to bottom:

- Chat: a synchronous communication tool
- Co-Writer: enables the participants to write the texts for the different assignments
- Sources: contains links to source materials for the assignments
- Participation tool: provides participants with a graphic display of their participation
- Source : a particular source was opened by the student

In this study, students have to collaborate in groups of three participants on a inquiry project about witches in medieval society based on historical sources. The research question focuses on the effects of participation awareness on the collaboration of the students. The main purpose of the VCRI is to enable students to collaborate on research projects, to help teachers to guide students while they are collaborating, and to enable researchers to collect data on the process of collaboration. Therefore all significant user events are logged by the server to enable our researchers to study the effects of our tools on the collaboration process. From the main log file, MEPA[2] files can be extracted. MEPA is an application for the annotation, coding and statistical analysis of verbal or nonverbal observational data or protocols, making analysis easier.

A bit of history

The VCRI is the result of years of developing different tools for CSCL research at Utrecht University. The first version was developed around 1995. This version was written in Visual Basic. One of the more surprising results was the feedback we received from our subjects. They actually liked the collaborative writing. One of the problems was the handling of socket communication. This proved cumbersome and the program was rewritten in Delphi (Pascal). This provided some improvement.

As we started on a subsequent project we sought an additional programmer to speed up the development of the next version. We found the company Equi4 willing to assist us. One of the first decisions was the choice of the programming language. Tcl was chosen for two main reasons:

1. Our application deals mostly with texts
2. Our application uses networking

This makes Tcl a natural choice. Development on the VCRI is driven by the research demands. These demands tend to change frequently, making it very important to use a flexible and interpreted programming language like Tcl. Its tight integration with Tk also makes it perfect for creating nice GUI's with little effort. Another key feature was cross platform availability, giving us the ability to run in almost every school. The fact that Tcl is open

source also was an important pro. Last but not least Tcl's clear and simple syntax makes learning Tcl easy.

Architecture

The VCRI is a client-server application. In principle, the client does all the heavy work while a customized TclHttpd[3] server distributes updates and saves the tool content. Distributing the work over all clients reduces the server's workload and optimizes CPU usage.
The client is kept as thin as possible, only providing a framework for communicating with the server and a login window. On user login the server queries a Metakit[4] database for the toolset and lets the client remotely source the corresponding files. This kind of remote sourcing, or dynamic loading, makes rapid development and bug fixes possible. Users can keep using the same client while still getting all the updates and bug fixes from the server every time they log in.
Client and server communication is http based. The client issues http requests with Tcl commands to be executed by the server. The server returns scripts to be executed on the client side as result for the http request. The main reason for using this RPC like communication is firewalls. Until recently, most schools which participated in our experiments used KennisNet, a government funded intranet. KennisNet blocked almost all ports (incoming and outgoing) and no exceptions were made. This prohibited a permanent socket connection. Since http requests and port 80 are always available, VCRI communication also became http based. The clients poll the server in regular intervals to exchange information.

Packages used

In developing the VCRI we've used some extensions and tools to provide the features pure Tcl/Tk was missing.

TclHttpd

This easily extensible and lightweight web server written in Tcl was perfect for our VCRI server. It provides a basic framework for client-server interaction giving us the room to focus on VCRI’s specific features. VCRI's use of http(s) for server communication also made TclHttpd a logical choice. At this time, our customized TclHttpd server runs on (SUSE) Linux, Windows, and Mac OS X.

Starpack

Schools are more willing to join when the effort on their part can be made as small as possible. An easy and straightforward installation is an important way to reduce that effort. Packaging the VCRI client as a Starpack makes installation equal to drag-and-drop and removing to delete. A Starpack is a Starkit plus a Tclkit runtime. *Starpacks are standalone executables, which run out of the box, making them even easier to distribute and use than Starkits.*[5]

Metakit

Earlier versions of the VCRI used plain files for storing persistent data. Increasing complexity and scale of the VCRI have made maintaining, archiving and tweaking these files more painstaking and placed a need for more structured and consistent data storage. We have decided to use a lightweight database for data storage.
We have chosen Metakit as our database because of its small size, efficiency, ease of use, and its easy to use bindings for Tcl (Mk4Tcl and Oomk[6]). Currently the VCRI server uses one Metakit database to store data, although in some cases plain files are still used.

TkHTML

The VCRI has a number of tools, which generate or display HTML. At first, we used OpTcl which is *a Tcl extension to provide connectivity to the resident hosts component model.*[7] Unfortunately, it only works for COM on Windows, which clashes with our cross platform objective. Another con is the inability to interact with OpTcl at a low level, handling events and manipulating the HTML content. Therefore, we decided to move away from OpTcl. Finally, we have chosen TkHTML[8], a Tcl extension written in C for rendering HTML-content. As a side note, the TkHTML project has recently been revitalized. CSS and XHTML support are among the project's main priorities.

TkOGL

Some of the tools offer 3D visualizations. Since Tcl/Tk lacks 3D capabilities we've used TkOGL[9] for this purpose. TkOGL is a Tcl extension written in C providing an interface to the OpenGL framework. It currently works on Linux and Windows but a Mac OS X build is also planned. TkOGL's most recent version (3.2) makes it possible to use OpenGL commands without (almost) any modifications.

[incr Tcl]

[incr Tcl][10] is a language extension for Tcl enabling object oriented programming by introducing the notion of classes, objects and other OO related terminology. By adding this extra layer of abstraction, it's easier to write and maintain large programs.

Problems

During VCRI development, we have come across a number of problems of which a couple will be highlighted in this paper. In the next paragraph, we will present some of our solutions to these problems.

Legacy code

As described in *A bit of history* the VCRI has come a long way since 1995. A lot of features and tools have been added but design wise things have stayed the same. This unchecked growth has lead to redundant and duplicate code, overlapping functionality and too many dependencies. At the beginning of the CRoCiCL project the decision was made to continue work on the current version of the VCRI instead of starting from scratch. The complexity of this legacy code has made it difficult to maintain and extend the code. Object oriented programming has turned out to be the solution to this problem.

User interface

The VCRI lacks eye candy. While this doesn't present a problem to those who are only interested in functionality, it's an important issue when designing software for teenagers, the target audience of the VCRI. In our experiments, it's crucial that the students enjoy working with the VCRI. An appealing user interface is key, especially for those users who have grown up with cool looking software. Another GUI related problem is the cross platform (and as a result non-platform) and inconsistent look of the VCRI, making it virtually impossible for users to make use of their knowledge of GUI conventions on their platform of choice. Unfortunately, lack of time and know-how makes it hard to fix this problem.

Platform dependencies

One of the main reasons for choosing Tcl is its cross platform availability. As development continued, shortcomings in Tcl/Tk's functionalities were patched using extensions. Not all of these extensions were 100% pure Tcl; as a result platform dependencies returned in the VCRI in the form of binary extensions such as TkHTML and TLS.

Different interests

Finally, there's been the problem of different interests among the project's stakeholders. Many times interests of users, researchers, and developers have conflicted. For example, adding a feature for research purposes can lead to unnecessary clutter (from the user's point of view). Furthermore, keeping everything as lean as possible makes logging all user events a bit tricky. Luckily, every one in the CRoCiCL team has at least a little experience in both research and programming, making it easier to resolve these conflicts by looking at the problem from different points of view.

Printing

One of our more recent problems is printing. For our application, we are looking for a printing solution to enable users to print the content of any VCRI tool window on any printer on any platform. Printing can be broken down into different sub problems. The first sub problem is to create a printable version of a tool's content. The other sub problem is letting the user (or the VCRI) select a printer and finally sending this printable version to the selected printer. Unfortunately, existing printing solutions only tackle one of these sub problems and are almost always platform dependent.

Solutions

In this paragraph we will present solutions to some of our problems mentioned in the *Problems* section. Please note that not all problems are discussed.

Object oriented programming

Some of the problems we mentioned earlier are closely related. The problems with complexity, redundancy and updating and using legacy code have to do with unraveling the relations between different chunks of code. Key ideas of object oriented programming are modularity and refactoring. By putting related code in one place (an object) interaction between different chunks of code is made simpler and more transparent. When all related functionality is provided by one object, the complexity of an application is greatly reduced. Instead of maintaining and adding code at different places, just one object needs to be changed when using an OO design. This also addresses the problem of redundancy because it's easier to recognize and fix redundancies.

Students and Tile

Another problem is the Spartan look of the VCRI, which cripples the user experience a bit. Since lack of time is still a problem, we have been trying to involve students in information sciences and interaction design. At the end of June, our application will be used as the subject of an assignment on user testing as part of the course usability engineering at Utrecht University. Hopefully, this will provide usable feedback on usability and look of our GUI. In the future, we hope to be a part of similar projects.
Tile[11], *an improved themeing engine for Tk*, is another possible solution for our GUI related problems. Tile can use so-called themes to provide a consistent look and feel. Themes consolidate all GUI options in one place, which minimizes the effort to create customized themes for different platforms, experiments or users.

The future

The VCRI will be used in at least two more experiments, one in September 2005 and one in September 2006. After these experiments, the VCRI will be released as free (and maybe open source) software for educational use. We're also planning to give third party developers the ability to extend the VCRI with new tools and functionality. Our aim is to release a product, which is easy to install (both client and server), easy to use, cross platform and fun.

Summary

Using Tcl to create the VCRI has proven to be a good choice. It's perfect for rapid prototyping, which has proved to be a pro in this research project with constantly evolving demands and requirements. The few shortcomings of Tcl we have encountered have been fixed by third party extensions. Being half way in the development of the VCRI in this project we can state that using Tcl has contributed a great deal to the result!

References

1. CRoCiCL: Computerized Representation of Coordination in Collaborative Learning, http://edugate.fss.uu.nl/~crocicl/
2. MEPA, http://edugate.fss.uu.nl/mepa/
3. Tcl Web Server, http://www.tcl.tk/software/tclhttpd/
4. Metakit by Equi4 Software, http://www.equi4.com/metakit.html
5. Beyond Tclkit - simplified deployment of scripted applications by Steve Landers, presented at the 2002 Tcl/Tk conference, Vancouver, http://www.equi4.com/papers/skpaper1.html
6. Object-oriented Metakit for Tcl, http://www.equi4.com/oomk.html
7. OpTcl, http://www2.cmp.uea.ac.uk/~fuzz/optcl/default.html
8. A HTML Widget For Tcl/Tk, http://tkhtml.tcl.tk/
9. TkOGL, http://hct.ece.ubc.ca/research/tkogl/tkogl/index.html
10. [incr Tcl] - Object-Oriented Programming in Tcl/Tk, http://incrtcl.sourceforge.net/itcl/
11. Tile: an improved themeing engine for Tk, http://tktable.sourceforge.net/tile/