Chalks

Design proposal

Rodrigo Benenson, Ricardo Niederberger

Draft of day
8th September 2004

Contents

1 Introduction

This text cover every defined aspect and propose a technical solution to implement the requirements. This is mostly a documentation of the inicial Chalks design.

Root concept: *Keep It Simple*, minimal complexity to acomplish the strictly necessary requirements.

2 Concurrent Editing

Direct implementation of the linear undo/redo algorithm described in Chengzheng Sun's works. This algorithm is the simpler aviable. It has a good compromise in the memory/cpu usage, with a some charge on the memory. Anyway as the objects managed are strings, the memory rarely grows more than a few tens of megabytes.

Chengzheng Sun's algorithm define three objects: the operations, the history buffer (HB), and the resulting text.

The operations are specific transformation over the text (insertion, deletion) originated from a specific version of the text. The history buffer is a buffer of all the operations applied over the text. And the resulting text is the consecuence of applying all the HB operations over the original text.

The concurrent editable algorithm define what to do with an new operations. The processing of the received operations will have effect over the HB and over the original resulting text.

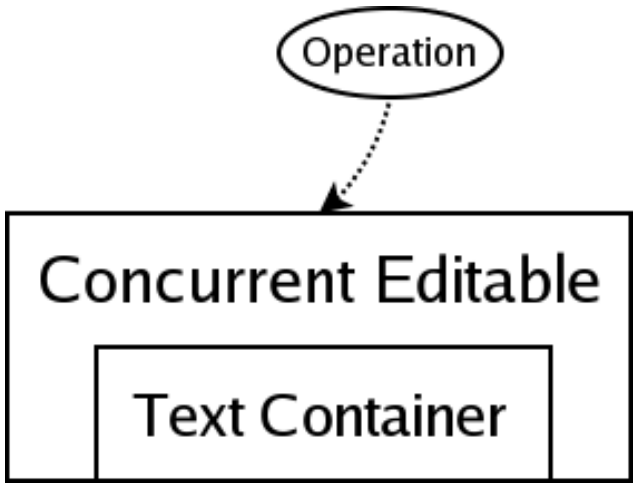

Figure 1: ConcurrentEditable, TextContainer and the Operations

Note: the actual WIP version does not separate in a clear enough way the TextContainer from the ConcurrentEditable object. I think this separation should be enforced to ease later integration with GUI

Using Python power the operations can be easilly represented as objects, following an easy api similar to Chengzheng Sun's notation.

In a similar way the Concurrent Editable algorithm can be implemented as an almost direct translation from the Chengzheng Sun papers to python code.

The `TextContainer` class is a simple one that contain an unicode string, and have methods for insertion, deletion of chars and the retrieval of the actual content.

3 Network

The definition and implementation of the networking system is the most difficult aspect of the software. Distributed bugs arises and issues not covered in the papers have to be afronted.

Esentially the network have to take care of three things:

- Sent operations have to arrive to every connected user
- If a new user login, everyone has to know about it
- If an user logout, everyone has to know about it

Every operation have a specific identifier. This identifier is the number of operations that the emitter site has received from other sites (including himself). This identifier define without ambiguity the *version* of the document over which the operation was generated.

To simplify the implementation instead of enumerating the sites (as in the paper) the sites are individualized by an unique id (ip+port). Then the operations are tagged with a dictionary of the kind `id1: ops_from_id1, id2: ops_from_id2, ...` (including site own id).

This approach becames less eficient if many sites just connect once, send an operation and then disconnect, but this is a rare case. This approach is less eficient than using an ordered list, but it is simpler and functional. It is simpler because we do not care about the order of arrival, the disconections and reconections, and when new users are added to the session the management is trivial.

3.1 Network topology

Following the KIS principle which is the simplest network architectures that does the job fine ?

The proposal is to use a "Connect to One, Accept N" topology, also known as a "simple tree". See the image 3.1 to get an idea.

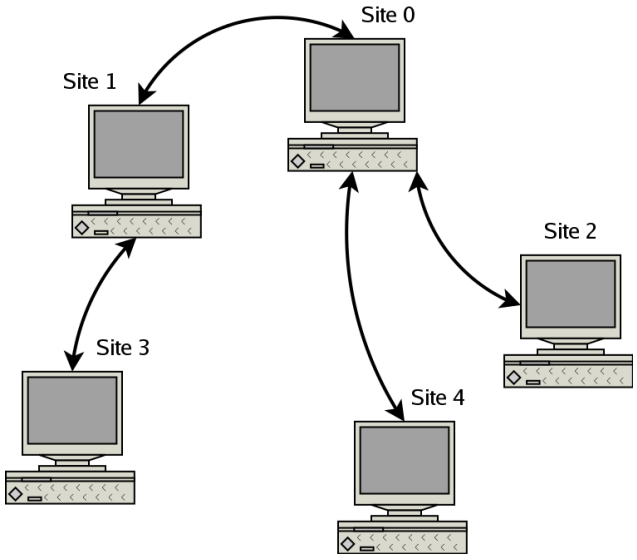

Figure 2: Connect to One, Accept N ilustrative topology

The concept of this topology is, very simple. One user only want to connect himself to one computer, so let's do that. If more than one user want to connect to the same computer let's do that. After the first connection no more connections are made.

This is topology is ineficient. In the figure, when site 4 edit the text, the operation will be relayed by site 0 and site 1, adding inecessary delays. This is true, but this topology is very simple implement. It has also an interesant advantage, it gives to the users the control of the network. In the figure, if sites [0,2,4] are in the same LAN, and sites [1,3], in another one, then the topology is not so bad. Also if an user is working in a slow connection (example, low SNR wireless conneciton), it would prefer to have only one connection to the nearest (in ping measure) machine. Having more than one connection add transfers costs to keep the TCP/IP links alive. If the users control the conection they can choose the computer with overload. A tipical user case will be simply N-to-One (central server receiving all the conections).

The tree topology is not only simple to implement but also simple to analyse.

When an site receive and operation it apply it locally and spread it to all the other known sites. As the topology is an acyclic graph, no special precautions has to be taken (to avoid repited reception).

Let's see what should happen when a user connect himself to a session.

- New user enter in the network

  An user has choosen a machine to enter in. This machine receive a new conection and has to send back, the actual text and the HB. We have to take care of the new operations that could be generated during the connection process.

  Chengzheng Sun's algorithm include a purge procedure to avoid to the HB unnecessarily big.

  At the first text editing the new user will generate an operation with a new entry in the tag. When a site receive it for the first time it will include it in the list of know sites. If a received operation does not include a know site id, obviously the generation has received zero operations from that site.

- An user quit the network

  No special precaution has to be taken when an user quit. Simply we have to disconnect from the parent node, and kill the childrens nodes. This will create a cascade that will close adecuatelly the branch.

Note: this could be an inconvenient, depending of the usage scheme. Maybe we could create a transaction system to invite the children nodes to connect themself to the parent node. But synchronisation issues arise ("what if packet arive to the parent while children-childrens are connecting them ?"). This is a non trivial topic that we should check. For a initial implementation branch die seems fine to me, but it is the desired behaviour ? (let's check the requiriment)

  HB purge should also purge sites info. If a site id does not appears more in the HB, then it should be deleted of the known sites list, thus shrinking back the packet sizes when no memory traces are keep in any connected site.

3.2 Protocol

`O`peration objects serialized and transfered via Twisted Spread.

- Well documented API
- Simple and eficient binary serialization (low bandwith usage)
- Well documented Serialization protocol (Banana)
- Fast serialization (C modules)
- Pythonic

3.3 API

Twisted

- Well documented API
- Full featured, spread, authentication, web services, diverse protocols, etc.
- Pythonic
- Easy to embed in the software distribution

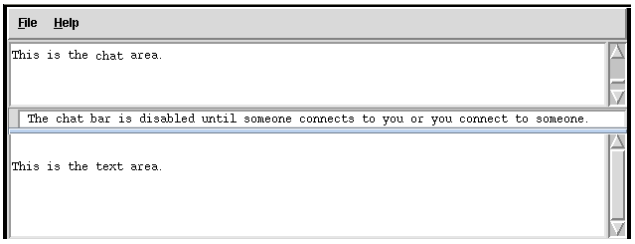

Figure 3: Text and chat areas in the same window

4 Gui

Show the text container, generate new operations over the text, allow sending chat messages to the other users.

The idea is to provide to the user all what it need to see in only one window, splitted in two areas: the text area and the chat area.

The chat area is used as a log window, to receive message, and have an input section to send messages. This input section can also be used to enter advanced commands (to enable/disable debug logs, to select special options, etc...). Different colors in the chat bar are used to differenciate diferent kind of messages.

The chat area is resizable (click and drag). If it is totally colapsed it transform itself into a status bar, that show the last line in the chat area.

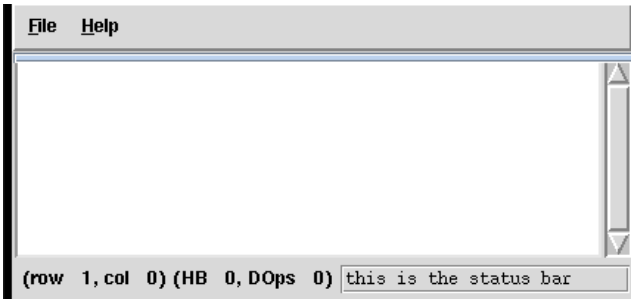

Figure 4: The chat area is colapsable, transforming itself in a normal status bar

In the text area color codes are used to differenciate the users entries. Also the local entry are marked in such a way that the local user know when a group of chars have been sent or not.

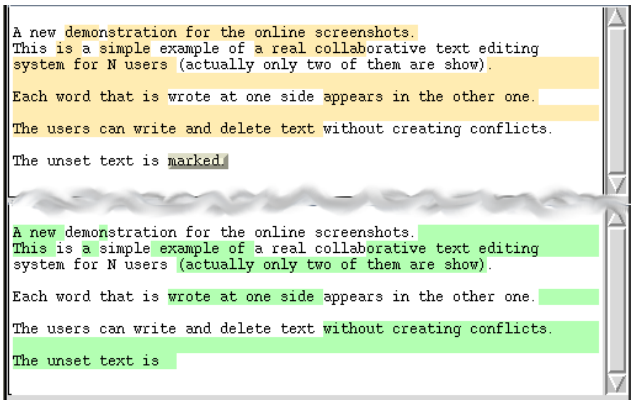

Figure 5: Text from other users are colored. Text not yet processed is raised

The color of the users is randomly choosen at the local site. The color generator choose in a family of smooth agradable colors.

4.1 Menus

The menus should be strictly minimal, siilar to NotePad ones, but with less options. The number of dialogs is also minimal, and they complexity very low.

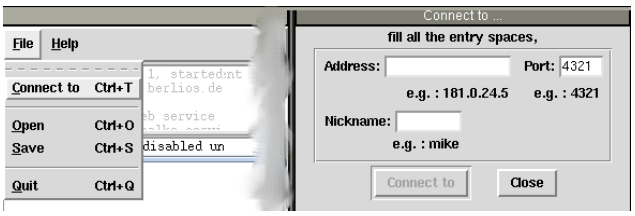

Figure 6: Simples menus and dialogs

4.2 API

Tkinter

Pros
- Easy to include in the python distribution
- Ligthweigth (? compared to wxwindows ?)
- Python default gui, crossplatform
- Good coloring facilities
- Documented, easy API
- Full Unicode support

Cons
- Manage text as a bidimensional object
- The gui is not similar to the OS default one (this may change in the future)

Note:the gui code implementation should be as well defined and separated as possible to allow future implementations using diferent gui engines (natives, scintilla, etc.)