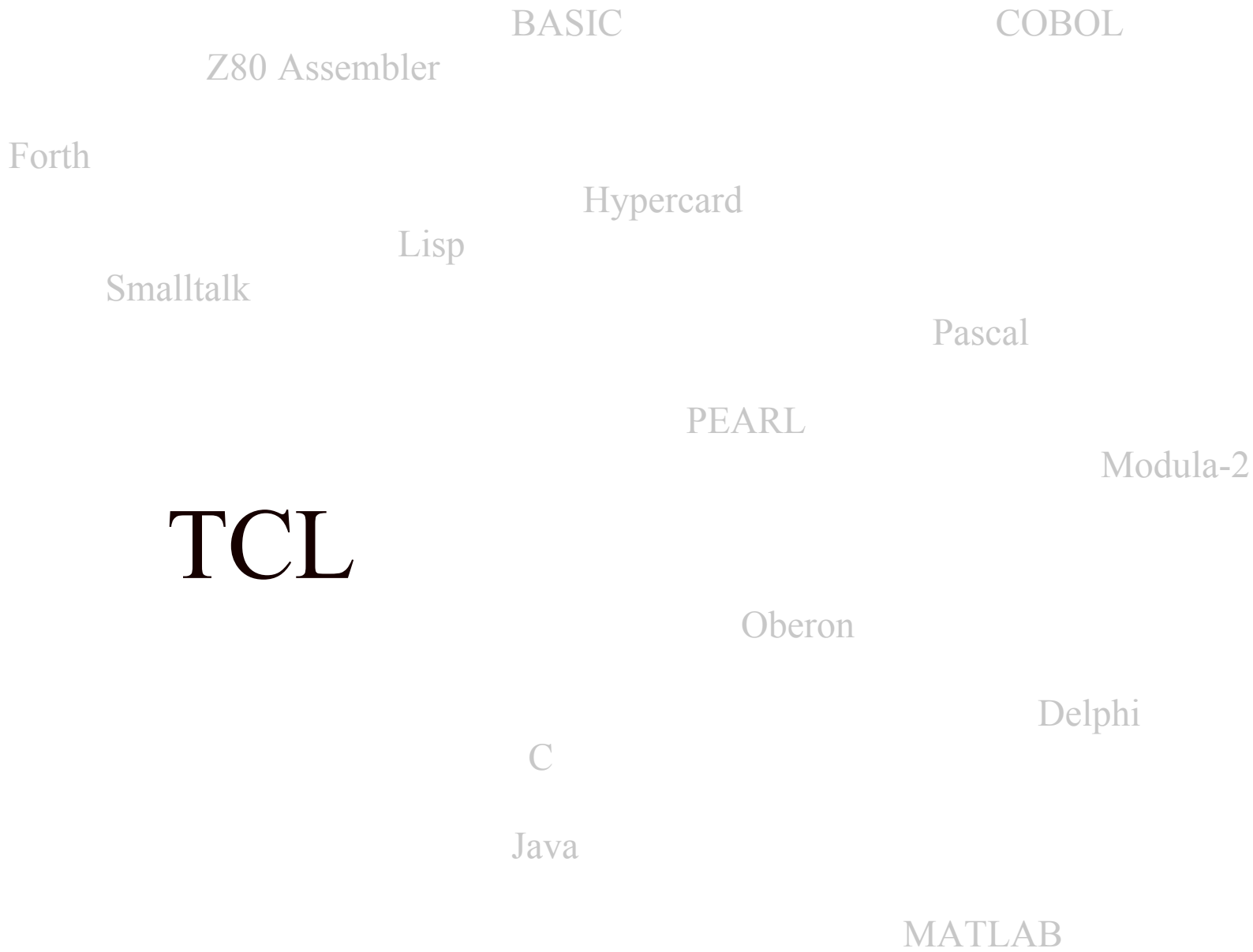
Presenting eti

- How i learned to stop worrying and created megawidgets -

Axel Nagelschmidt http://axn.dyndns.org


BASIC
COBOL
Z80 Assembler
Forth
Hypercard
Lisp
Smalltalk
Pascal
PEARL
Modula-2
TCL
Oberon
Delphi
C
Java
MATLAB

How it all began ...

Started programming on a calculator TI-59

Built the Sinclair ZX-80 from a soldering kit after a holiday in England

Advanced to ZX-81, ZX Spectrum, Atari ST 520, Macintosh SE

Studied Computer Science and Mathematics in Erlangen

Worked for several institutes and clinical departments during years of study, designed PC Card for measuring tasks with DMA
Started business to assemble PCs according to customer requirements

Used CP/M, GEM, DOS, Windows, Mac OS, Minix, IRIX, Solaris, Linux, Mac OS X and other systems

Together with a friend founded a computer company positioned in trainings and consulting, mainly on java and OO techniques
Preparing course materials, administrating Solaris and Linux machines for classes and developers, first contact to TCL

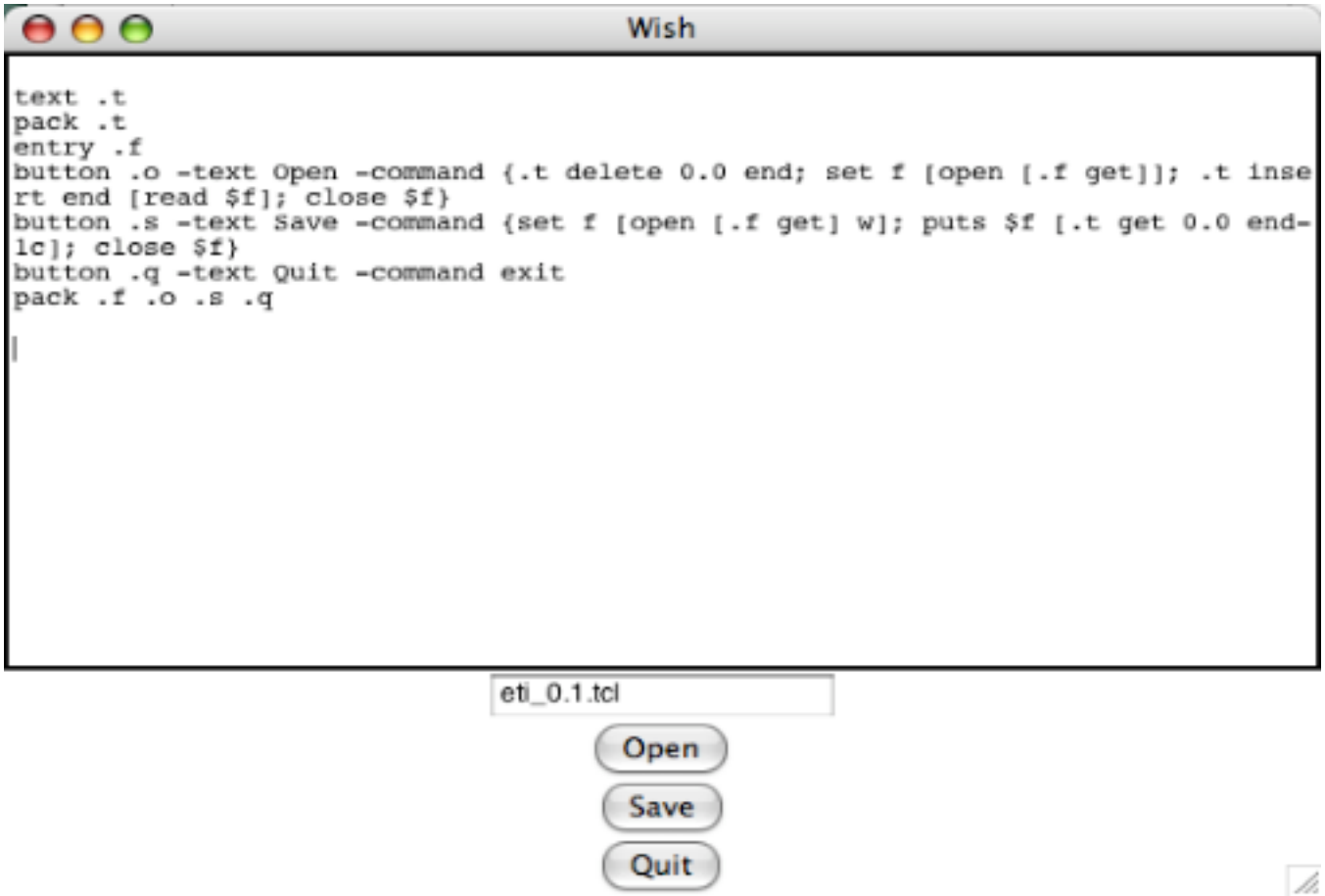
Working since 9 years in a company for biomedical technology, as developer, database developer and Solaris administrator

Today running Ubuntu on PCs, Mac OS X on several Macs (PPC), Solaris on some SUNs, being forced to use Win-XP at work

Until now met John Mc Carthy, Niklaus Wirth, Douglas Adams (SIGGRAPH Keynote in New Orleans 1996), happy to meet more experts here !

Writing a complete editor is possible after one month of learning TCL!

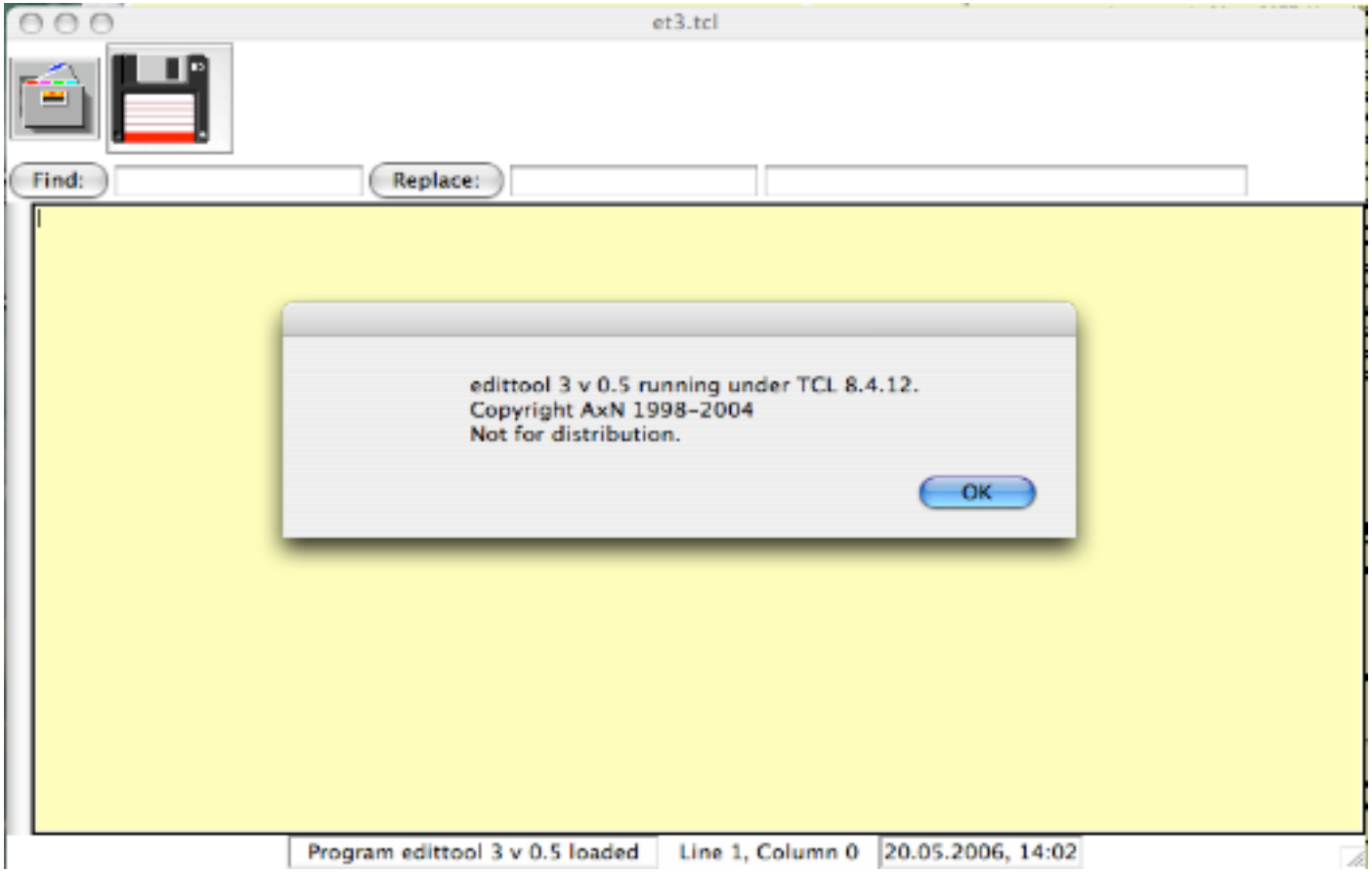
I wrote something like this around the year 1997:



eti 0.1, nearly feature complete

Even better version 0.3:

Has menubar (not shown here, screenshot from Mac)
Options to save / save as / open / close file, toolbar, find and replace
Options to change font family / size / background colour
Status bar with status info, cursor position and clock
Can mail it‘s content
Ported to Solaris (because no change necessary)
No widespread use, probably because never deployed



Started database programming with TCL and oratcl, network programming cross platform

Found TCL be able to connect to a database very easily

Created an online transaction processing application, processing tens of thousands messages per day from cardiac pacemakers

Needed to write a multiplexing front end to an SMS sender/receiver that allowed only one connect from a client

Added monitoring tools to watch processes, checks for availability of hosts, DB listeners, Webservers, disk space, load ...
Running several instances of tclhttpd on production and test servers to watch processing and resources, lightweight, easy to use

Using tequila to synchronize processes and have a look at process figures from several clients

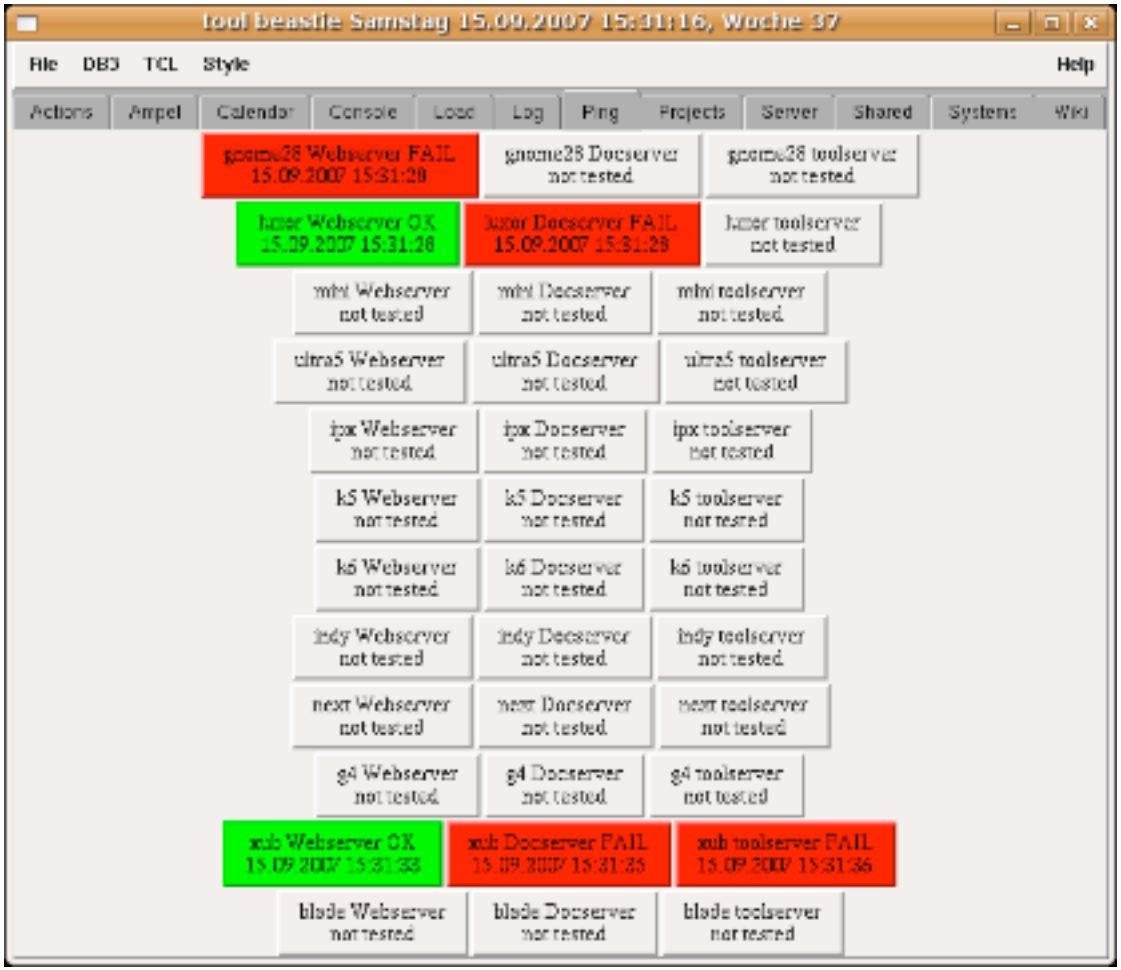
Also created more and more tools for me and collegues

- simple application
- single window
- some buttons and entries
- database connection
- monitoring tools
- text transformation
- database repair tasks
- XML parsing and creation of TCL and SQL scripts from description of data modules

Each of these sharing many properties with other tools, having it‘s own method for persistant storage like **.**files, ini-files etc.

A tool is born ...

Collected these into a common tabnotebook, without any interaction between the modules yet:



Usage of object orientation seemed to be a good idea:

- ◇ Have a mini editor for any kind of text. Derive HTML editor, TCL editor etc. from this.
- ◇ Have a set of classes needed for graphic programs, derive different graphical drawings / simulations from this.
- ◇ Specify a set of common used actions to include into the menu or a toolbar
- ◇ Have multiple instances of the same type, so can use a notebook with several tabs
- ◇ incr TCl offering classes and incr TK offering megawidgets like notebook seemed natural

But (1) ...

- ◇ incr TK seemed to be more complicated than necessary
- ◇ incr TK seemed to be slow because of code overhead

Solution I:

- ◇ create one superclass drawing the toolbar, the notebook and the status bar
- ◇ create one superclass in incr TCL (not TK!) for all instances (called etimodules)
- ◇ have each etimodul draw it‘s own frame
- ◇ inherit superclass etimodul to take care of common needed methods

But (2) ...

- ◇ creating multiple windows with notebooks needed another superclass
- ◆ inheritance for widgets was not really used
    - ◇ tcledit or diffedit could not really inherit the class miniedit, different count and layout of widgets
- ◇ learned about tile and increasing / ongoing support for Mac OS X

Solution II:

- ◇ recreated some widgets using tile components
- ◇ added menu to change style to any existing tile theme
- ◇ some usage of mkWidgets, because better creation of megawidgets was anticipated

But (3) ...

- ◇ mixing iwidgets with tile widgets was no good idea
- ◇ mkWidgets usable for megawidgets, but not really an OO system
- ◇ mkWidgets not being supported any longer?
- ◇ not much time to try something new

Then ...

- ◇ visited the 5th european TCL User meeting (and organized the 6th one)
- ◇ got more involved using tequila, thus making shared apps an easy possibility
- ◇ learned about integrating OpenGL to create apps using 3D graphics
- ◇ learned more about tile, the notebook api has some differences
- ◇ learned more on creating fullfeatured apps
- ◇ learned more on mixing text and graphics to create integrated learning experiences
- ◇ learned more on tablelist and it‘s integration into tile

Other influences happening in this time:

- ◇ More usage of Java and especially Eclipse at work
- ◇ Our company introducing Lotus Notes, me visiting a developers course
- ◇ Seeing that TCL can control much of windows through twapi
- ◇ .NET and mono appearing in widespread use, new IDEs anjuta and monodevelop
- ◇ Applications becoming more complex, each one developing their (own) makro language
- ◇ Seeing that TCL can integrate calls to graphic libraries like ImageMagic or OpenGL
- ◇ Understanding that integration in big consistent frameworks like Oberon or Squeak makes usage much easier
- ◇ Anticipating to learn that TCL can integrate calls to control OpenOffice.org
- ◇ Learning about snit and it‘s ongoing support and integration with TCL 8.5
- ◇ Writing more applications as helpers or for database access, needing a usable and robust framework to do so
- ◇ Expecting user interfaces to allow users much more choice and flexibility - ”graphical programming”
- ◇ Not wanting to learn a new P*-language every year
- ◇ Seeing that TCL is here to stay, easy to use, easy to teach or use as description language, cross platform

=> Decided to base framework and apps on TCL, integrating good usage examples into an usable IDE

(OK, did so before with Pascal, Oberon and Delphi ...)

Moving from incr TCL to snit

Topic	incr TCL	snit
Header	itcl::class <name>	snit::widget <name>
Inheritance vs. Delegation	inherit <upperclass>	delegate <method> to <otherclass>
Building widgets	frame $w, delivered as argument	set w $win, frame already created
Reference to instance	$this	$self
Calling own methods	<method>	$self <method>
declaring method	method <name> <args> <body>	method <name> <args> <body>
Destroying instance	itcl::class delete object $this	destroy $self
Overwrite class definition	no, only after deletion of class	yes, like the TCL way
Inherit methods	yes	no, but delegation helps

Rewrote about 5 major classes and the core of the framework in about 3 days.

Need to rewrite toolbar, and some handy megawidgets used from mkWidgets to snit.

First results seem to be very promising.

Skeleton of an etimodule:

```
snit::widget somepage {

        typecomponent super
        delegate method * to super
        typeconstructor {set super [etimodul %AUTO%]}

        variable w

        constructor args {

                # create the modules widgets
                # initialize variables

        }

        method start {} {
                # perform actions, start timers etc.
        }

        method onshow {} {
                # perform actions when module is shown again
        }

        method onhide {} {
                # perform actions when module is hidden
        }
```

```
	method f9 {} {
		# refresh, reload actions when requested to do so
	}

	method clone {} {
		# create a clone of the same type of module
	}

	method close {} {
		# stop timers, free resources, close this module
	}


	method whatever {} {
		# do something
	}

	# ......


}
```

All methods not implemented in a specific etimodule are called in the super class by delegation.

Some of them are empty, others do the easy and safe default action, e.g. clone, aboutme

A digression on network and monitoring:

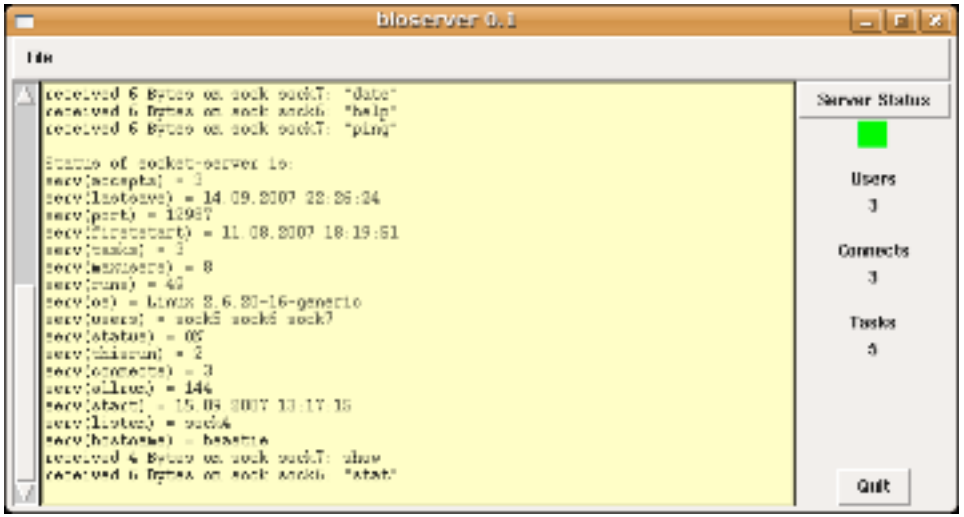
Needed a small TCP/IP connected client / server tool for cross platform monitoring and maintenance applications

Synchronous **reliable** remote calls to hosts or services that are down lead to sometimes intolerable delays in client tools

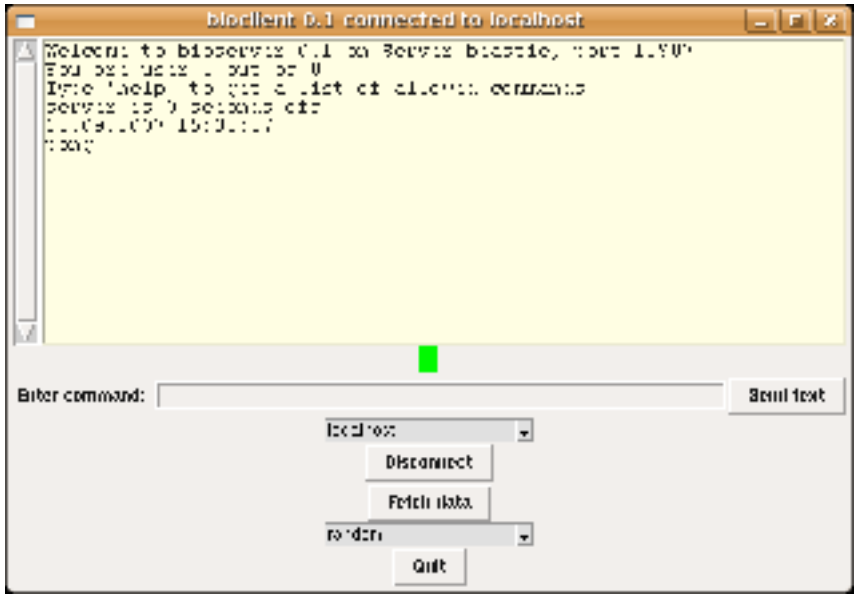
Until now we could live with tequila for process synchronization, carefully dividing into modules which may encounter delays and those that need short response times

Some problems remain, will explore more about this later. Maybe UDP can help?

Also using tclhttpd modules as application server, after integrating SQLite database into the webserver.



<==



Webclient with SQLite database should make deployment of application and upgrades very easy ...

```
#!/usr/local/bin/tclsh


# idea from my wiki 147, thanks to DKF
# stripped down, no interp, working and tested

package require Tk
package require http

proc http_source url {
        set token [http::geturl $url]
        set script [http::data $token]
        http::cleanup $token
        eval $script
}

http_source http://deployment.local/tclapps/initapp.tcl
```

(Remember the first version? 7 lines of code is all that is needed, the possibilities are endless ...)

Upgrading and migrating through tests is a reliable way ...

Also a good way to document bugs and show when they are fixed

- [x] `tcltest::test achterbahn test2 -body {math3::achterbahn 39} -result 34`
- [x] `tcltest::test fibonacci test1 -body {math3::fibonacci 14} -result 377`
- [x] `tcltest::test fibonacci test2 -body {math3::fibonacci 19} -result 4181`
- [x] `tcltest::test roman test1 -body {math3::roman 25} -result XXV`

Lately had to follow strict QM standard procedures, parts of TCL application now certified for medical usage in clinical studies

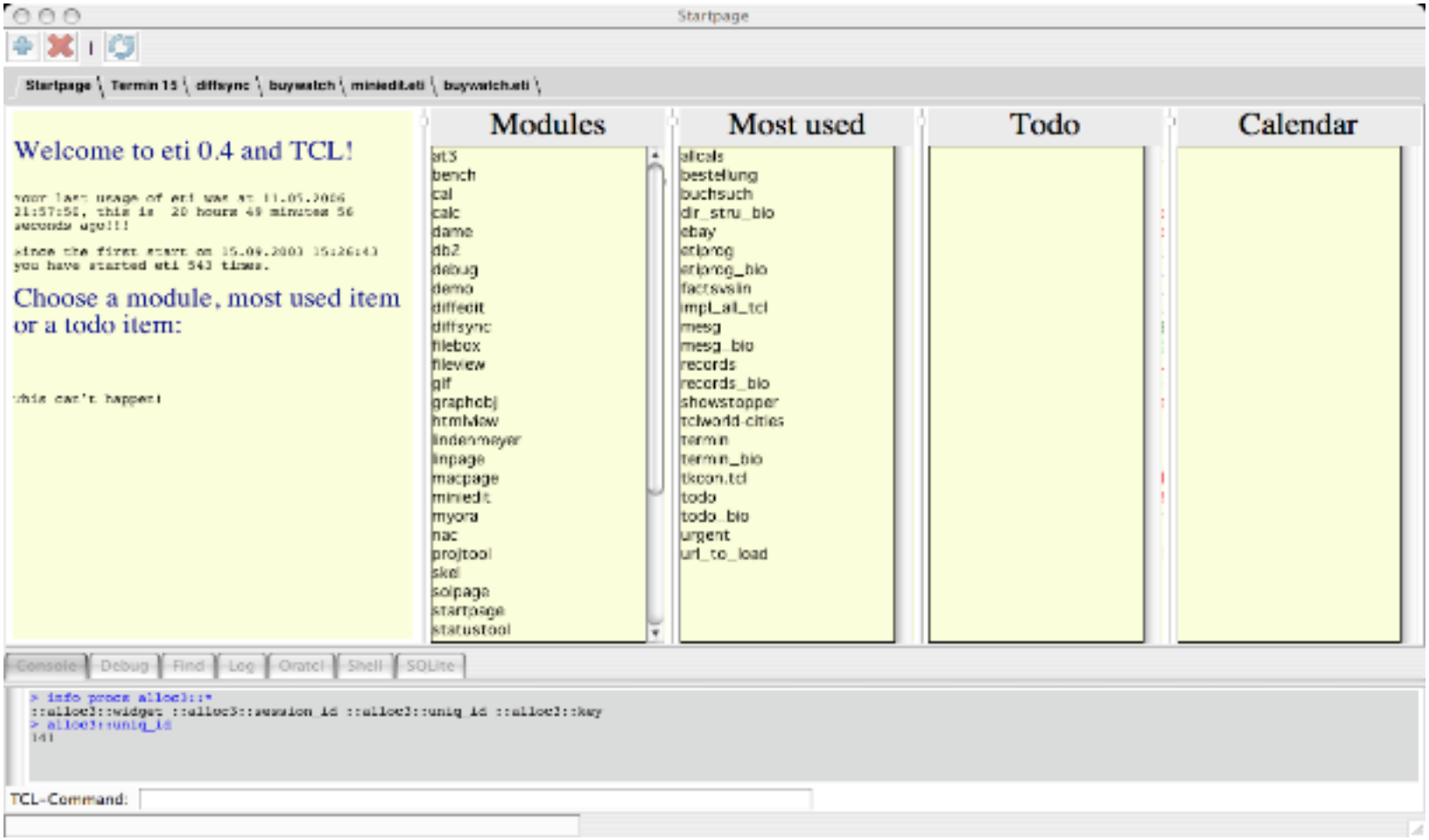
Requirement No.	Description of requirement
REQ-628-003.012	The new module shall be called routines3 and conform to the standard calls for etimodules. It shall be placed in the etilib2 module library.
REQ-628-003.016	The module shall have a method demo to start a small demo showing it's features.
REQ-628-003.018	The module shall have a method to reliably predict the weather for the next 20 days.

Test-driven development helps to structure requirements into code and to prove that all usecases have been considered:

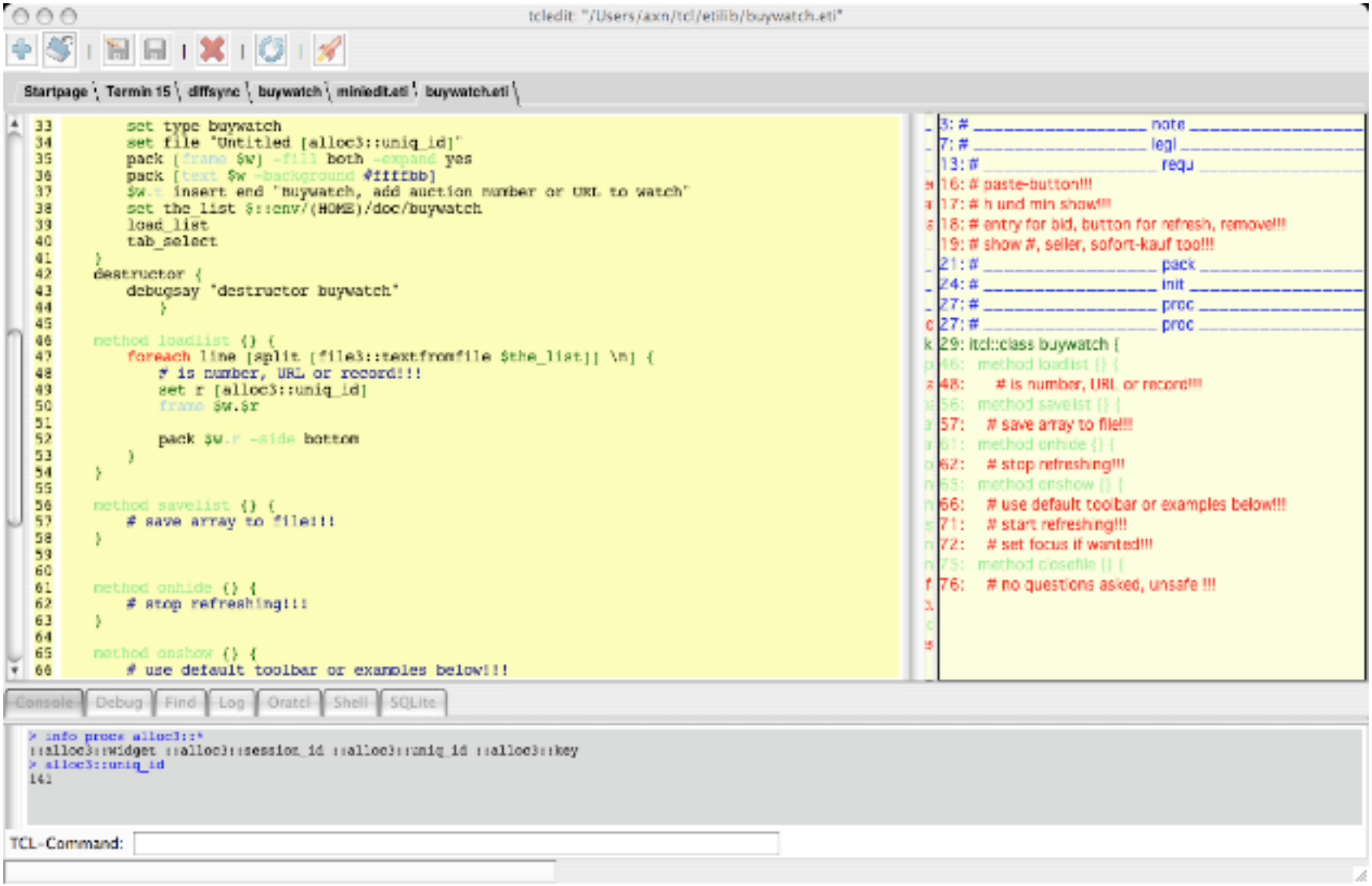
- [x] `tcltest::test routines3 exists -body {file exists $tclpath/etilib2/routines3.tcl} -result 1`
- [x] `tcltest::test routines3 demo -body {routines3 .a; expr {[lsearch [.a info methods] demo] >= 0}} -result 1`
- [ ] `tcltest::test routines3 test2 -body {# not so easy to write the test} -result {# the programmer will implement whatever he thinks is correct}`

Examples of etimodules:

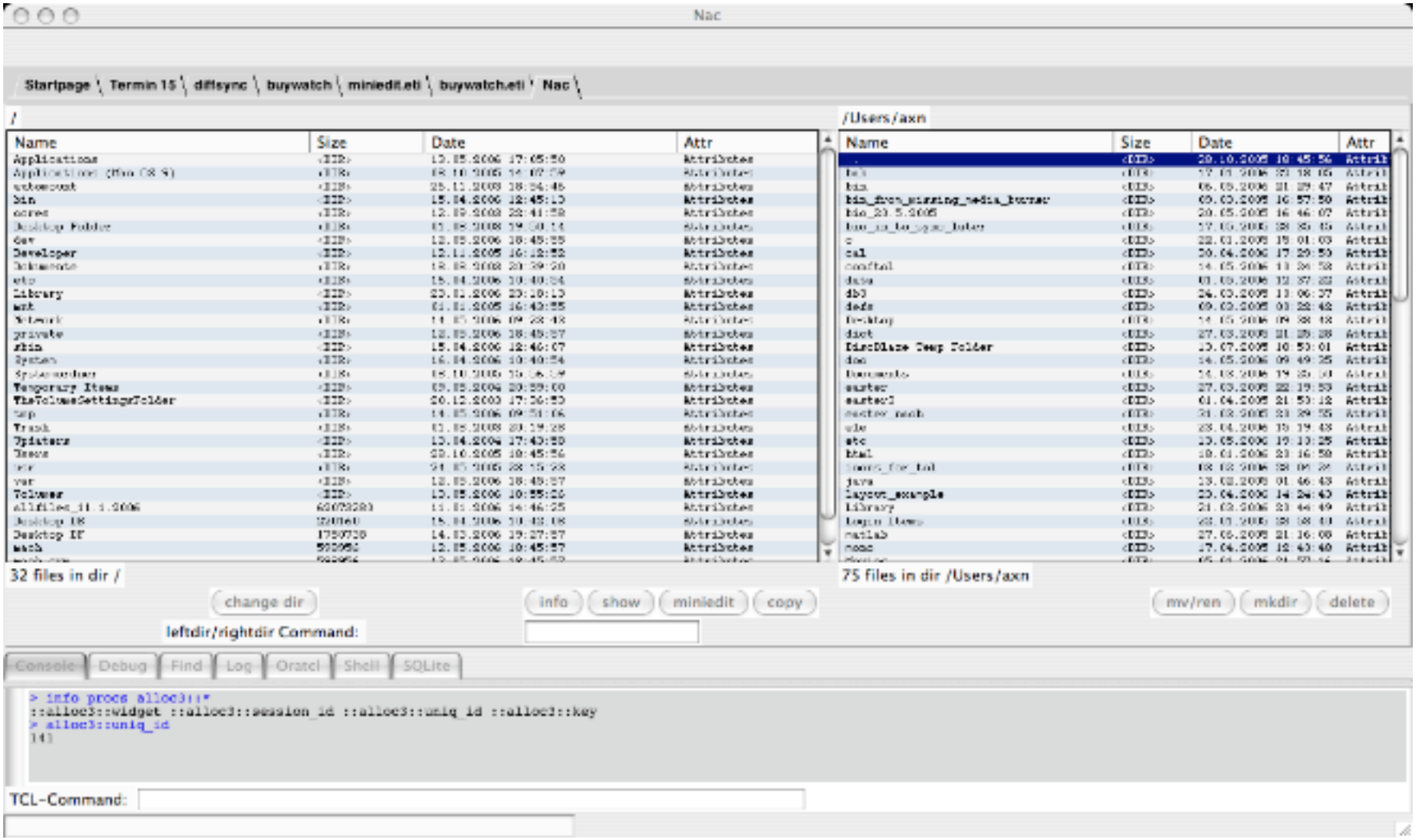
Startpage, allowing to start other modules, showing most used documents, todo and calendar
Older framework for helper applications (terminal, consoles for sqlite and oratcl) shown below



tcledit, editor with syntax colouring based on ctext
Allows working in projects and libraries, highlights requirements and proc / method headers



nac, (not another commander), because this is my main productivity tool
Using tablelist, but need more intelligent bindings for keyboard operation and cleaning of layout



Using a database in the application, using the application in the database ...

Many possibilities with metakit, starkits etc. exist

Coming from Oracle, i found SQLite the most easy method to use a lightweight though very powerful database in my apps

Already had a personal wiki inside of eti, using structure of this to migrate the project into the database, shown in demo now

```
sqlite>
sqlite> .tables
addr   cal    code   prefs  todo   wiki
sqlite> .schema wiki
CREATE TABLE wiki (id id, item string, text string, first date, count integer, last date);
sqlite> .schema code
CREATE TABLE code (id id, type string, value string, name string, first date, count number, last date);
sqlite> select distinct type from code;
app lib module
sqlite>
```

Still using cvs in filesystem, but will move versioning into database soon, have code archive on other servers in net.

There is still more to come and more todo ...

TCL 8.5 should be ready soon (or may be it is already?)

Check tcl modules (see TIP #189) and convert my libraries to new format

API of etimodules has to be frozen at some point for deployment

Update webserver and have code repository there

Solve some problems with timeout waits on TCP/IP

Have a single application framework for easier deployment of apps, perhaps using freewrap?

Follow development of Eclipse, Lotus and the Komodo Project ...

info, to show information about the application
Thanks again, this is really a tool made possible by the community!