CS3283

GUI Programming

Hugh Anderson
Department of Computing Science,
School of Computing
NUS
hugh@comp.nus.edu.sg

Preface

The FOLDOC[1] dictionary of computing defines a GUI as:

> *(GUI) The use of pictures rather than just words to represent the input and output of a program. A program with a GUI runs under some windowing system (e.g. The X Window System, Microsoft Windows, Acorn RISC OS, NEXTSTEP). The program displays certain icons, buttons, dialogue boxes etc. in its windows on the screen and the user controls it mainly by moving a pointer on the screen (typically controlled by a mouse) and selecting certain objects by pressing buttons on the mouse while the pointer is pointing at them.*
>
> *Though Apple Computer would like to claim they invented the GUI with their Macintosh operating system, the concept originated in the early 1970s at Xerox's PARC laboratory.*

The *official* NUS description of CS3283 is:

> *This module aims to teach the nuts and bolts of GUI programming. At the end of the course, students will acquire practical knowledge in Windows programming and techniques of programming interactive systems. Topics include Windows programming, Motif, Tcl/Tk programming.*

An interesting aspect of this course is that there is some emphasis on graphical visualization methods. These notes are just an expanded set of overheads - you are asked to read supporting papers, and maintain an active interest in GUI design and implementation. You may find the latest copy of the notes at

http://www.comp.nus.edu.sg/~cs3283/ftp/cs3283.pdf

It is my intent in this course to give lots of examples of different GUIs - successful and otherwise, and to show the methods used to develop these interfaces.

[1] The Free-On-Line-Dictionary-Of-Computing is found at http://wombat.doc.ic.ac.uk/foldoc/index.html.

Assessment: The proposed assessment may be modified slightly if the need arises, but currently is as follows:

Assessment			Weighting	Grade
Assignments				35%
Ass1	Design	Group	10	
Ass2	Design, prototype	Individual	20	
Ass3	GUI implementation - Swing/MFC	Individual	40	
Ass4	GUI/Visualization implementation - Tk/Java3D	Group	30	
Tutorials				5%
Mid-term		Closed book		10%
Final Exam		Open Book		50%
Total marks				**100%**

Textbook: Unfortunately, no single textbook adequately covers the material presented in this course, so I am providing a set of notes, which may be supplemented by readings from:

1. The User Interface Concepts & Design, Lon Barfield, Addison-Wesley (1993)
2. The JFC Swing Tutorial - A Guide to Constructing GUIs, Kathy Walrath & Mary Campione, Addison-Wesley (1999)
3. Tcl and the Tk Toolkit, John K. Ousterhout, Addison-Wesley (1994)

Tools: You may wish to download and install the following toolkits, copies of which are found at the course web site at http://www.comp.nus.edu.sg/~cs3283/ftp.

1. JDK Version 1.3
2. The Cygwin development system, including Tcl/Tk and debuggers
3. The Netbeans IDE, the glade GUI builder, and so on

Topics to be covered: During the course, we cover

- Fundamental GUI concepts (1 lecture)
- Design and programming techniques (3 lectures)
- Cross platform GUI development using Swing and Tcl/Tk (6 lectures)
- Visualization techniques (2 lectures)

Enjoy the course!

Contents

Chapter 1

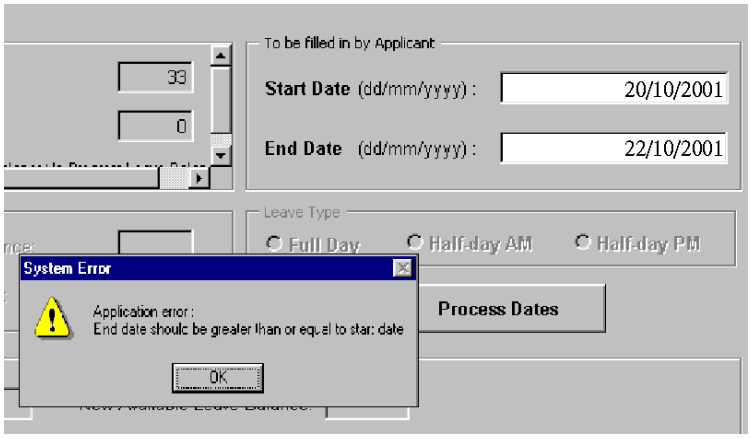
GUI concepts

The user interface for software has changed over the years. Early user interfaces were text based, and normally had a fairly simple interrogative style - prompting the user to provide needed information in a fixed order. The modern GUI provides for complex interaction between the user and the application, and often relies on shared concepts or metaphors.

GUI programming is about the conceptualization, design and implementation of that part of a software application which is concerned with user interaction.

1.1 How not to do GUI

I have decided to start with an example from here at NUS - the leave system for staff in the department. Notice the dates I have entered, and the error message[1]:



The question is ... what did I do next?

[1] It says: "**End date should be greater than or equal to start date**".

What do we learn from this? I think two points stand out:

1. Try out your applications before delivering them.
2. Ensure that error messages are precise, and indicate the next step.

During this course, I hope to give useful examples of interfaces and techniques - but equally important is to see these poor examples - *how-not-to-do* things. How about this one:

Often a single poor example can remind you of things to avoid - a picture of a poor GUI reminding you of a whole range of things.

1.2 General rules of GUI

A key point to understand as we begin our investigation of GUI development is that effective GUIs owe more to effective psychology than to effective programming. A graphical user interface is not just windows, icons and so on - it also includes an abstract view - not visible, but understood by the user. A successful GUI will have no clash between this view of the user interface and the more concrete one involving icons, buttons and so on. The term ergonomics might be used here - we hope for a correlation between the physical and conceptual ergonomics.

Another key point is that humans are not equipped to handle multiple things at one time, and this leads us to try to keep interfaces simple and uncluttered.

Humans are particularly good at navigating systems which have some analogy to things they know - for example the use of the desktop metaphor is well established and works well in most cultures. Icons are also useful, but shouldn’t be abused.

To summarize:

1. Ensure correlation between What-u-c and What-u-think
2. KISS
3. Analogy, metaphor and icons

Always remember to include the U in GUI.

1.2.1 Do's and don'ts

The following tips are gleaned from various sources, and are provided to help you know what you *should* do with your GUIs:

- **Do follow standards** - for example **quit** is the bottom item on the leftmost menu and so on. In general people prefer new applications to follow established standards.
- **Do be predictable and responsive** - events should have an immediate response, and should operate in well understood and predictable fashion.
- **Do be flexible** - the interface should admit a wide variation of allowable sequences, rather than forcing a particular ordering. An undo facility is often helpful here.
- **Don't forget the user** - it is important to consider the pyschological, physical, and social attributes of people when designing user interfaces for them.
- **Don't forget the machine/environment** - but watch for nasty implementation details showing up in user interfaces.
- **Dont assume things** - like "*the user will know how to ...*" - provide informative help and clear actions.

1.3 Types of applications

Not all applications benefit from a GUI. Consider the area of embedded control systems (such as lift controllers, washing machine controllers) - these systems certainly have a user interface which requires careful design, but not necessarily a GUI. In this course, we will not consider this sort of interface.

However, there are many application areas that do benefit from a GUI:

- **Immersive applications:** - games, medical imaging, avatar based CSCW and so on.
- **Office and business applications:** - for use by anyone.
- **Interactive control systems:** - flight systems, remote control and so on.

In addition, a newer application area involves the use of visualization to examine large data sets:

- **Data mining:** - delving into some set of data.

Finally, there is the use of GUI in WAP enabled devices and on PDAs. This is a specialist topic, which will not be covered in this course.

1.4 Native environments/platforms

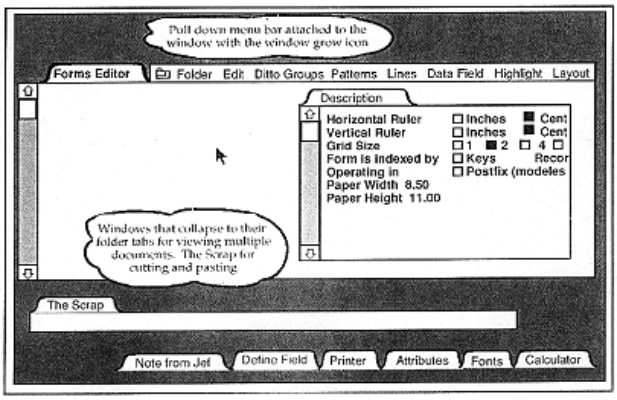
Early user interfaces were primitive things which barely disguised the underlying machines - nearly always text-based, and fixed in operation.

Contrary to popular public opinion, “Windows” is not the only GUI window system. The Macintosh system, and the UNIX X window system both predate Microsoft’s GUI system, and each have interesting features that have yet to be added to “Windows”. For example the UNIX X window system allows a clear separation between the display and the processing, whereas “Windows” display and processing must be on the same machine.

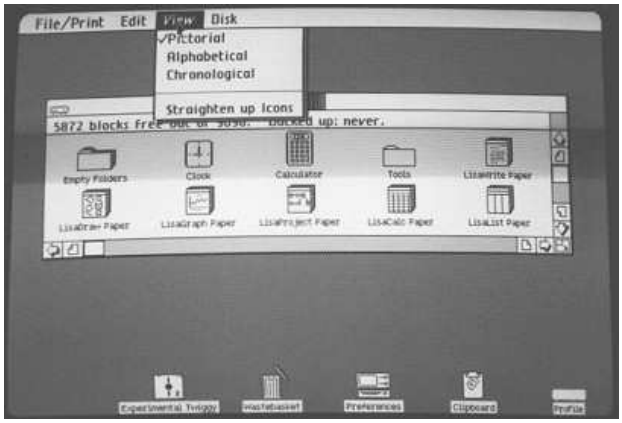
By far the most common GUI windowed environment on the desktop is the one found in Win95/98, and our programming API for it is known as Win32.

1.4.1 MacOS

It is interesting to see the development of the Macintosh window system from an early prototype in 1979 through to its current incarnation - MacOSX.



Note that this first prototype used folder tabs in a way that is no longer done, and did not appear to use icons. Within a year, the first commercial Macintosh system had the following interface:

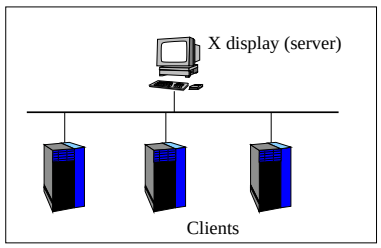


Macintosh operating systems have a single *user*, single *display* orientation, although the latest version of the OS is built on top of a multi-user (UNIX) kernel. The latest version has some interesting features - most notable is the application docking bar *magnification* feature:

1.4.2 X

The X window system[2] is a sophisticated and well developed system which allows for multimedia software and hardware components to be distributed around a network. At its simplest level, it allows a program and its display to be on different computers.

The architectural view of X is a little peculiar. The designers view the display as central to the system, and the software running on the display is called the X-server:



From the diagram, we can easily identify three essential components of X:

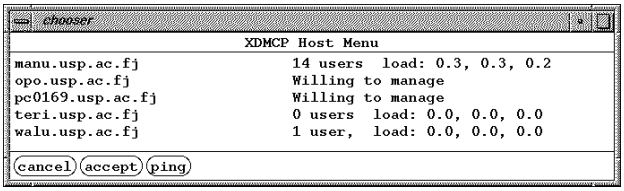
1. **The X server** - providing the high resolution graphical display(s[3]), keyboard and mouse.
2. **The X protocol** - providing standard communication methods between the distributed software components.
3. **X clients** - programs with graphical display output, running on (perhaps) many machines.

[2]The system is called X, or the X window system. UNIX weenies insist that it is **not** called X-windows!
[3]Mechanism, not policy!

There are two other components:

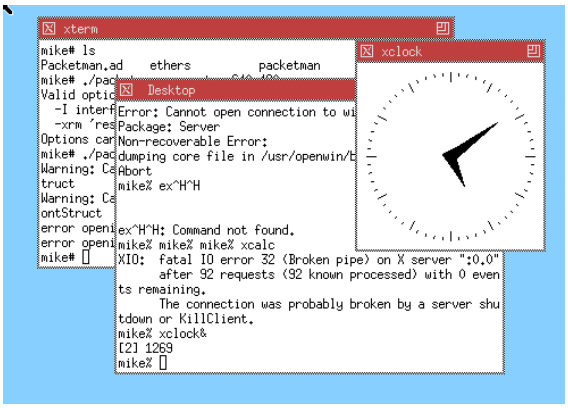
- **The Window manager(s)** - providing decorations around each window.
- **The Display manager(s)** - providing access to the system.

The display manager controls access to displays. The diagram shows a simple display manager allowing selection of one of a number of hosts. When you select a host, you are presented with a login window for that particular host.

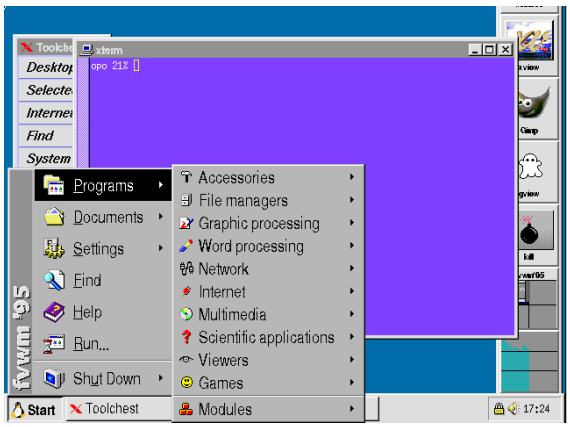


or

There are of course other display managers, and login windows. There are also many different window managers:



or



1.4.3 Win32

Win32 is the 32 bit successor of the Win16 API - the original Windows Application Programming Interface. Win32 is a generic name for 4 (slightly) different APIs - providing standard function calls for accessing the GUI, file system, processes and so on. The Win32 API on Win95 is a subset of those on WinNT, so applications written for Win95 should be portable to WinNT. The reverse is not always true, but most WinNT applications can run on Win95/98.

The normal way for you to access Win32 functions is by using a precompiled library from a C program. C programmers include a set of header files, and applications link at run time to the Win32 DLLs.

The API for Win9X has three sections:

- **KERNEL:** - the low level kernel services in user32.dll.
- **GDI:** - Graphics Device Interface - drawing and printing in gdi32.dll.
- **USER:** - User Interface controls, windows and messaging services in kernel32.dll.

In Windows NT these services are kernel calls.

1.5 Non-native environments/platforms

The following systems can be used to provide a consistent environment that is independant of the host operating systems:

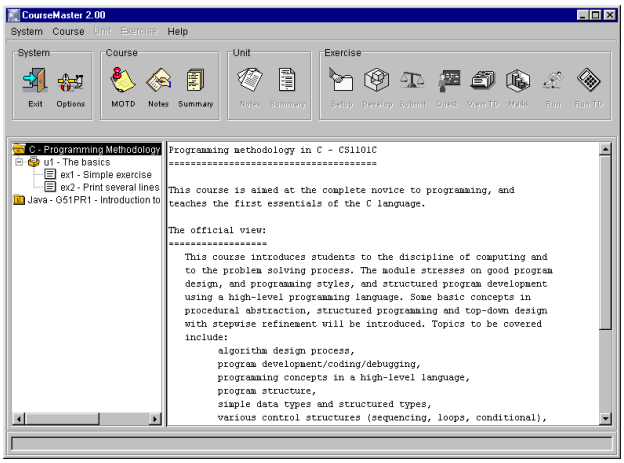
- Java/Swing
- Web browser interfaces
- Thin client systems

Each is discussed in turn in the following sections.

1.5.1 Java

Sun Microsystem’s development of Java has always been done with portability issues in mind. The JVM (Java Virtual Machine) is available on all platforms, and runs reasonably well on them. There are some unresolved issues - particularly in the areas of security, fonts and efficiency - for example the Blackdown distribution of Java for Linux is reputed to be about ten times slower than Sun’s in some areas of its use.

However - despite this - it is relatively easy to write a portable application - for delivery either as an applet in a web page, or as a standalone application. The **swing** windowing toolkit is the Java API for GUI development.



The all-java standalone application shown above is used for automated laboratory assessment of programs.

1.5.2 Web browser interfaces

The first web servers provided static pages of hypertext and images, but fairly quickly, demand led to the specification of a standard for active page generation - known as CGI - the Common Gateway Interface.

CGI specifies how to pass arguments to a program on a server as part of the HTTP request. The program might then look up a database before generating some HTML to pass back to the browser. A CGI program can be any program which can accept command line arguments - Perl is a common choice for writing these programs.

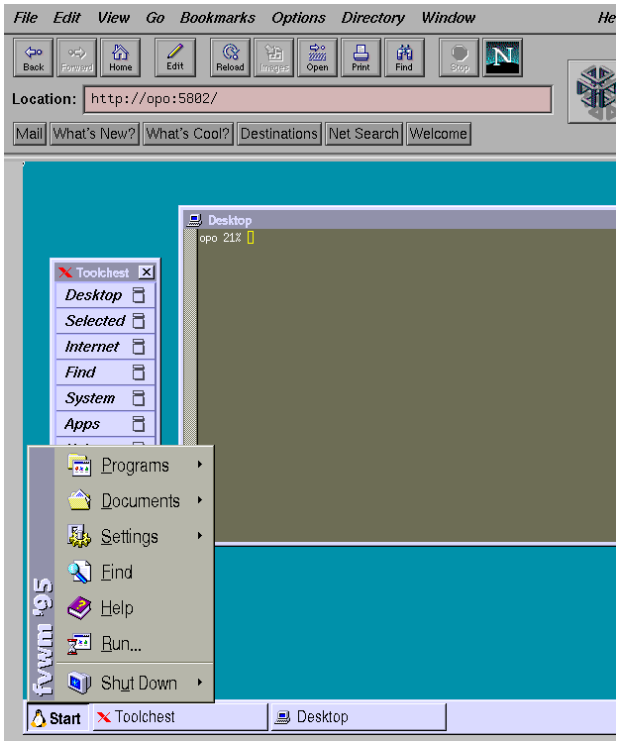
You should be aware that poorly constructed CGI scripts can result in security problems for the server, and also that there is normally a process overhead for each script started. More recently there have been various other web/interface techniques - for example the use of:

1. Java applets, to allow processing at the browser,
2. PHP (a server-side, cross-platform, HTML-embedded scripting language), or
3. ASP (a scripting environment for Microsoft Internet Information Server in which you can combine HTML, scripts and reusable ActiveX server components).

In this course we will look at the use of Java in this role.

1.5.3 Thin client systems

A relatively recent development involves moving the X-server (or equivalent) from the machine with the display to a larger machine, and then using a smaller computer to actually display the data. Here is a thin client written in Java, running as an applet in a web page:



1.6 Widget sets

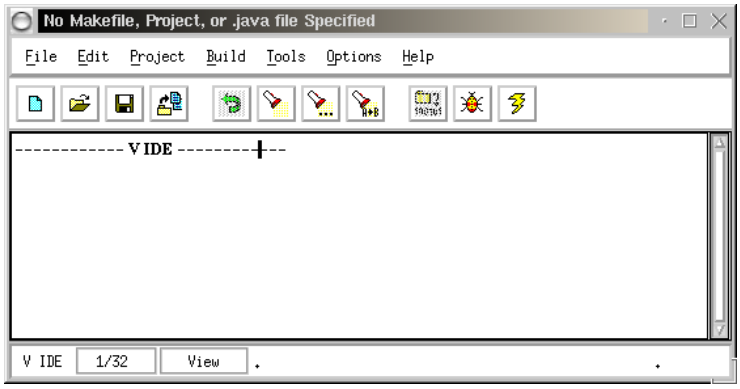
One definition of a *widget* is:

> *[possibly evoking "window gadget"] In graphical user interfaces, a combination of a graphic symbol and some program code to perform a specific function. E.g. a scroll-bar or button. Windowing systems usually provide widget libraries (sets) containing commonly used widgets drawn in a certain style and with consistent behaviour.*

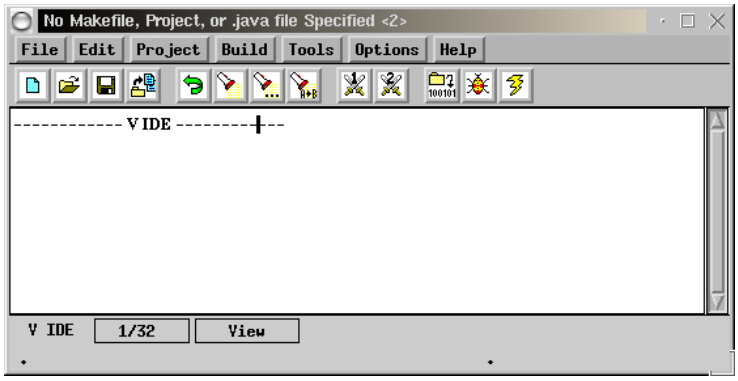
When we use different widget sets, our applications have a slightly different look-and-feel.

The following two screenshots are of the same application - the first linked with the Motif widget set, and the second with the Athena Widget set - notice the differences in the look of the two applications.

Motif:



Athena:



The ICS widget databook has a series of useful widgets to extend the basic Motif set, including ones for bar graphs and so on.

1.7 Summary of topics

In this module, we introduced the following topics:

- Rules of GUI
- Types of applications
- Windowing/GUI environments
- Widgets

Questions for Module 1

1. List three rules of things-to-avoid when developing GUI applications. For each rule, give an example which demonstrates the problem.
2. State one way in which an application written for the X-window environment may be different from an application written for a Win32 environment.
3. Find one example of a GUI application with a clear use of metaphor. Describe the application and the metaphor.
4. Research: What is a DLL?
5. Research: Why is java considered more secure for use in a distributed environment than (say) C?
6. Research: What is the principal use of PHP?

Further study

- The hall of shame:
  http://www.iarchitect.com/mshame.htm.
  This link appears to be dead, but a search of the Internet reveals copies of parts of it, and other similar web sites:
  http://www.umlchina.com/GUI/Tab.htm.
  http://www.umlchina.com/GUI/Controls.htm.
  http://www.umlchina.com/GUI/Termino.htm.
  http://www.umlchina.com/GUI/Misplaced.htm.
  http://pixelcentric.net/x-shame/.
  http://pixelcentric.net/x-shame/moz.html.
- A brief history of HCI:
  http://www.comp.nus.edu.sg/~cs3283/ftp/BriefHistoryOfHCI.ps.gz.

1.8 Sample Assignment - design

This is just a sample assignment. Assignment 1 for this semester's CS3283 will be distributed in class.

Task:

- Identify a GUI application which you think you can improve.
- Design an improvement.
- State how you would *test* the improvement.

Deliverables:

- A title page containing your name(s) and matriculation number(s).
- A two to five page document containing
  - A description of the application - perhaps with screenshots.
  - A description of that part of the application that needs improving, clearly stating *why* you think it needs to be changed.
  - A description of the improvement that you would make, clearly stating *why* you think this change would be better.
  - A testing methodology for the change that you want to make.

Note that this assignment does not require you to implement any change, just to describe one that you would make.

Chapter 2

Design

Before investigating more detailed processes of design, it is worthwhile to consider more general issues related to design. For example, please remember that we often don't bother designing small things (a snack before lunch, the seating arrangement at the dinner table), but for large things, we insist on prior design (an HDB apartment building, a bridge...). Successful design approaches an art form, involving partially understood balancing tricks with many competing constraints. The more design that you do, the better you get at it, but it is hard to discover the points that result in a successful design.

User Interface (UI) design has one identifying characteristic that separates it from other design areas - the principal concern is with the *user* of the system, not the constraints of the hardware. This leads us to a common mindset for a UI designer - the UI designer must primarily consider the *human factor* when designing systems.

2.1 How not to design

Consider the following UI for searching for property listings. It has some *upsetting* qualities.

In addition, you might also consider the utility of regular expression pattern matching for files (compared with the point and click interface). Why is **"ls *.c"** better than point and click?

In summary, I think very good rules to keep in mind are to:

- Avoid doing things just because you know how to do them.
- Make your designs be driven by requirements.

2.2 The design process

The design process involves both

- specification of the behaviour of a product, and
- specification of the detailed techniques used to implement the product.

In each area, there exist a range of tools and techniques that can benefit any software product, although there is no clear agreement[1] on which methodologies should be used. Having said this though, it must be emphasized that design *should* be done, and it should mostly be done *before* implementation. (I say mostly, because experience shows us that the design often undergoes an iterative phase, where the design changes as more and more of the implementation is done.)

2.2.1 Role of designer

A software designer cannot operate in isolation. The software designer interacts with people (the users and implementers), and as well has a responsibility to tie designs back to specific requirements (from the original analysis), and specific constraints (from the implementers, and users).

The design must be a readable, understandable, implementable document.

To achieve this, the designer uses *abstraction* extensively, at many different levels, and must be prepared to argue *for* the use of a particular abstraction. *The design of graphical interfaces is no different.*

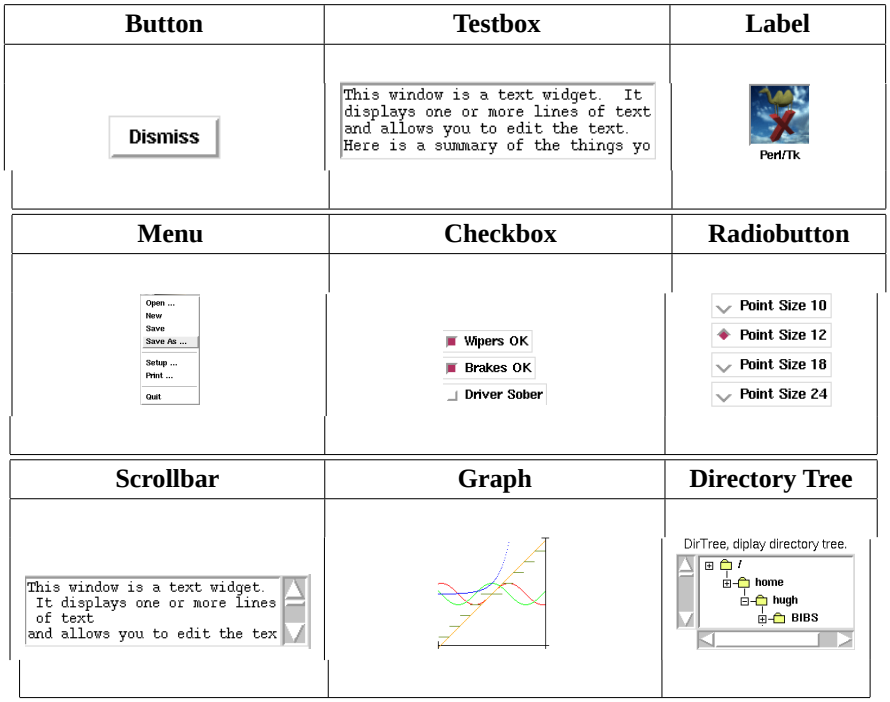
The base abstraction found in GUIs that does not appear elsewhere is the *iconic* abstraction - something is called *iconic* if it has some likeness to what it denotes. The simplest use of icons is when we represent a text file on disk using an icon that looks like a sheet of paper:

[1] By contrast, many other engineering disciplines *do* have generally accepted techniques to be used.

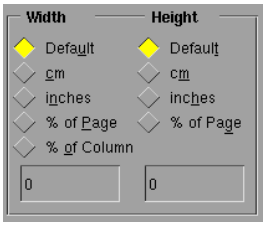
Here there is a clear relationship between the icon and the text file. We also have higher level abstractions - for example, the desktop and wastebasket metaphors. The designer needs to become familiar with successful abstractions like these, so that they can be used and so that new abstractions may be evaluated.

2.2.2 Building blocks of user interfaces

Beginning with the visible items, we have a range of widgets from the very simple iconic ones (such as the button widget), through to more complex ones:



In addition, we have invisible components - for example we use container widgets to construct more complex interfaces from a group of simpler ones:



Finally, you should remember sundry GUI components such as cursors, fonts, and colours, and well understood GUI actions such as - drag-n-drop, cut-n-paste.

2.2.3 Tool support, use cases and modelling

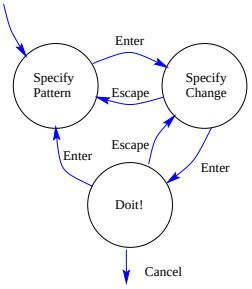
In general, the designer somehow imagines and proposes common scenarios[2] for the use of the software, and

1. checks to see if the scenarios are *consistent*, and *complete*,
2. tries out the scenarios on people to see if they work,
3. tests the scenarios and attempts to quantify their behaviour.

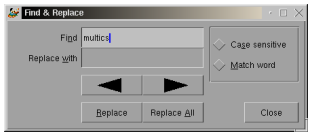
There are a range of tools we can bring to bear on these design problems. For example:

State-diagrams: - Used to specify and check the behaviour of a user interaction.

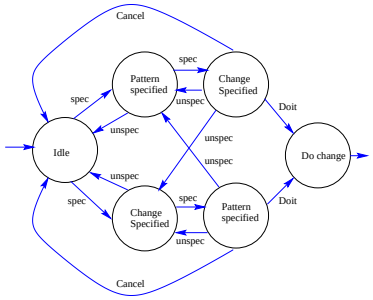
A simple older-style user interface for *find-and-replace* might involve first asking for the pattern of text to find, and then for text to replace it with. A *state-diagram* would look like this:



Note the states, and the labelled transitions. Consider this GUI-style *find-and-replace*:



The state-diagram for this might be quite complex - perhaps something like this:

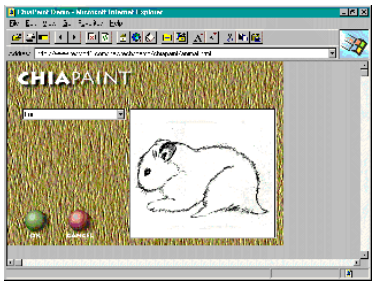


[2] Scenarios=Use_cases. Use_cases=scenarios.

Note that in this state diagram, the states are different, and have different meaning - as are the transitions - which are no longer single key presses - they may now involve complex functions. This focus on detail related to the state of a dialog is not trivial. There is a well known example of a poorly constructed dialog, that contributed to the death of cancer patients in the US see [4].

Modelling: - Used to demonstrate the UI, without actually implementing the *core* software.

Dan Bricklen's demo program (a demo copy is available at http://www.brickin.com/) is worth looking at for modelling a user interface. There is an amusing demo called **chiapaint**.



It is also relatively easy to model a new UI using Tcl/Tk.

2.2.4 OO technology and design

The principle features of OO technology [5] are as follows:

1. Abstraction,
2. Information hiding,
3. Inheritance,
4. Polymorphism, and
5. Genericity

The inheritance and polymorphism features of 00 technology have supplied a mechanism for creating/updating and maintaining effective software libraries. These libraries contain generally useful classes instead of parts of old projects, and it is a *librarian's* duty to ensure the generalization[3] of the classes.

Once a software library is in place, we can look in the library to find what we already have, and what is 'close enough'. For example the 'people' class may already exist, and we may decide that a generic 'combiner' class is close enough to 'booking area' to warrant its use.

[3]For example: classes 'airline', 'booking', 'person', 'flight', 'batchmode' rather than class 'batchmode_airline_booking_system'.

The next stop is to detail the features of each of the new classes. All of this is design, and is almost effortless - if we start detailing features and find it is 'all wrong', we can just step back to re-arranging/factoring the classes.

2.3 GUI specification and design

GUI design has to meld four possibly conflicting elements:

1. Software model - the structure of our data and overall architecture of the software developed during the normal system design process.
2. User profile - the types of targetted users of the product, with their specific characteristics.
3. Product perception - the mental image developed by an end user in relation to the use of the GUI product.
4. Product image - the specification of the GUI - screenshots, descriptions or specifications of it's behaviour.

In general, a GUI is successful when the product perception matches the product image.

Pressman's [6] principles for general software specifications need some modification for visualization and GUI specification. We need not, for instance, concern ourselves with *"the context in which the software inter-operates with other system components"*. Our concern is to:

> Develop a functional and behavioural response specification in terms of its cognitive aspects.

The functional and behavioural response specification is turned inside-out from a normal *software* specification. With a *software* behavioural model, we start with an analysis of states, events and actions, and specify the expected views as a result. With GUI specification, our orientation is to start with the views, and specify the states, events and actions associated with those views.

2.3.1 A basis for GUI specification and design

One of the most characteristic elements of many GUI programs is the use of the event-driven software architecture. When the designer adopts this paradigm, the GUI program is viewed as a series of response routines for particular events.

In addition, the software may require asynchronously running components. An implementation may use a number of threads for the asynchronous tasks, along with a set of event response routines. For example, a word processor may asynchronously spell-check a document, underlining questionable words.

A possible outline structure for a GUI design document might be:

1. User requirement
2. Environment
    (a) Software constraints
    (b) Other constraints
3. Interface design
    (a) Overview
    (b) Interface description
        i. Prototype screens
        ii. Functional specifications
        iii. Behavioural specifications
4. Testing methodology

Note the example of a design document along these lines in Appendix A.

2.3.2 Formal GUI design

Some aspects of GUI design can be easily formalized. For example, in Section 2.2.3 we saw the state-diagram used to define and describe the interaction behaviour of a user interface. Z, a specification language, has been used to *formally* specify complex GUI interactions. There are supporting Z tools which can then automatically test the specification for completeness and correctness with respect to some more abstract specification.

More details may be found in [3], and the handout [1], found at

http://www.cs.virginia.edu/~jck/publications/zum.97.pdf

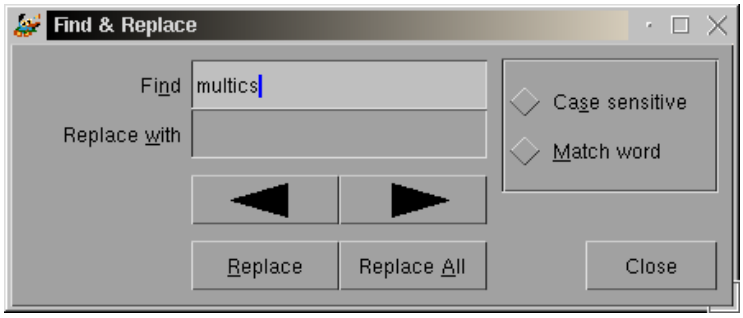
It describes the use of formal specification tools and notations in constructing the interface to a nuclear reactor.

2.3.3 Examples of GUI designs

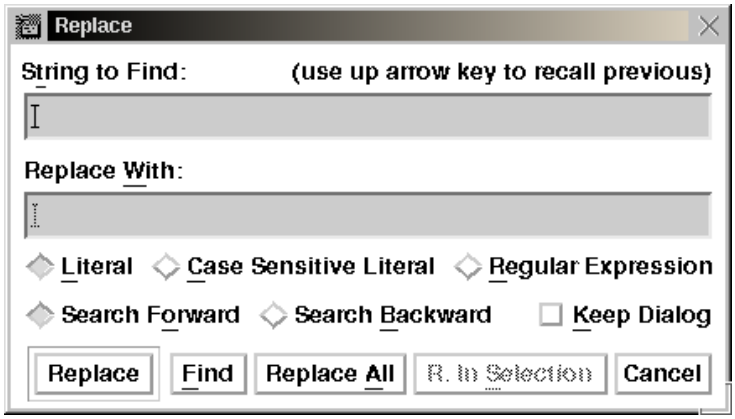
Here are some examples of different designs for similar things, with some brief comparative comments:

Dialog boxes for find-and-replace:

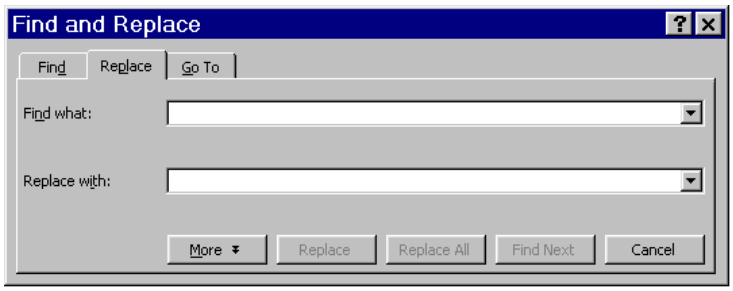
This dialog box in the **LyX** word processor was confusing the first time I tried it:



This one is from **nedit**. The check buttons are a bit confusing, but the up arrow recall of previous strings works well.

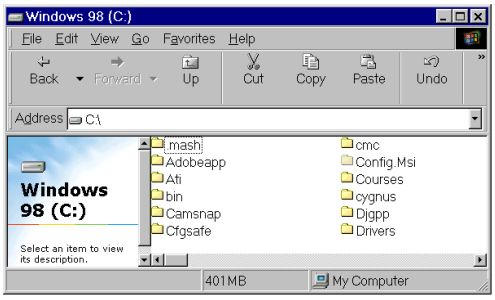


This is the **Word** dialog box.

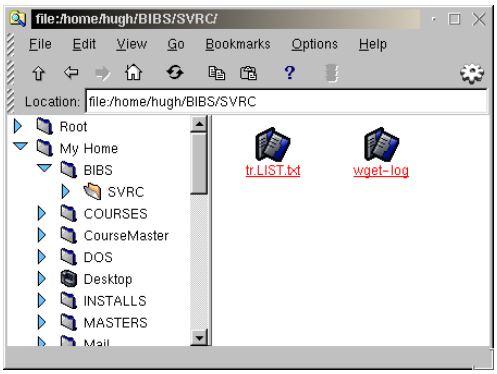


File system navigation:

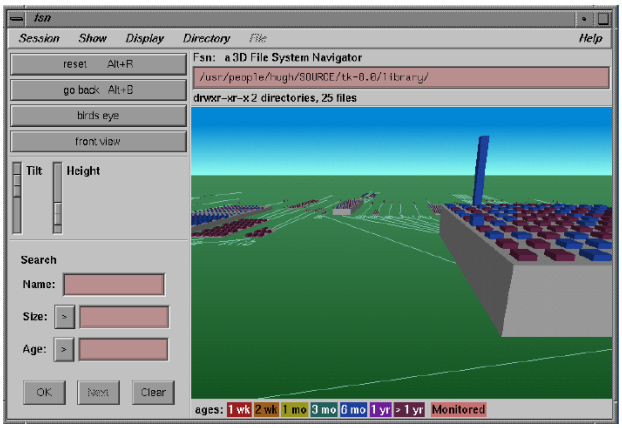
The familiar **win98** file manager borrows the basic concept from MAC file managers, and is quite easy to use.



A more explicit directory tree style file manager. The expanding arrow tree list on the left is a nice feature.



SGI have a (freeware) file manager called **fsn**, which briefy appeared in the movie “Jurassic Park”. It has a large computational overhead, but is fun to use.



2.4 3D vizualization specification and design

Visualization design has a similar structure to GUI design - a difference being the focus on the use of *analogy*.

2.4.1 A basis for visualization specification and design

Eick [2] proposes the following guidelines as a basis for engineering effective visualizations:

1. Focus the visualization on task-specific user needs.
2. Use a whole-database overview display.
3. Encode the data using colour, shape, size, position.
4. Use drill-down, filters and multiple linked views in a direct manipulation user interface.
5. Use smooth animation to show the evolution of time varying data.

With visualization specification, our orientation is again to start with the views, and specify the states, events and actions associated with those views. There is an example of design along these lines in Appendix B. Here is a possible outline for a visualization specification:

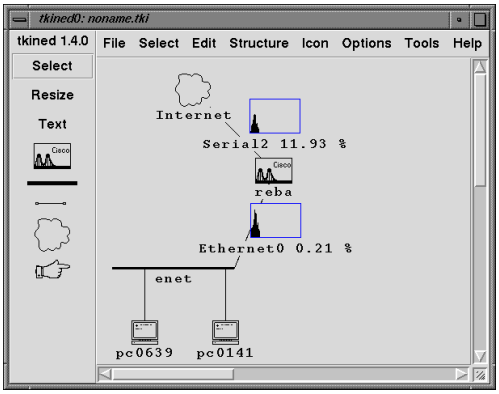
1. User requirement
2. Environment
   (a) Software constraints
   (b) Other constraints
3. Interface design
   (a) Overview
   (b) Interface description
      i. Drill-down and other displays
      ii. Encoding
4. Testing methodologies

2.4.2 Examples of visualization design

There are many examples of data visualizations, and I have just taken some from the world of network management - starting from simple graphical displays through to 3D images.

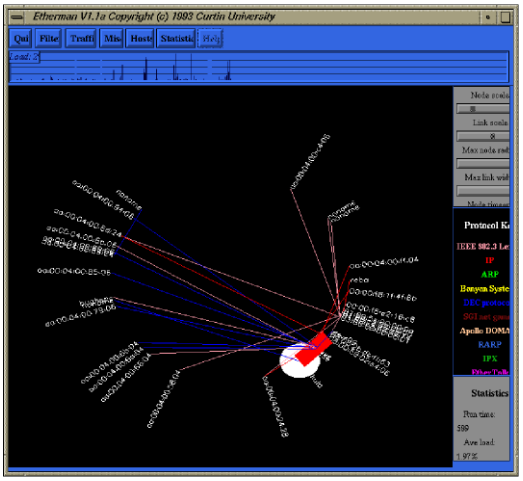
Graphs and diagramming:

Tkined[4] is a freely available SNMP management station. It centers around an effective graphical network editor which can be used to diagram a network.



Unusual display - compact visualization:

Etherman is a medium sized monolithic application which runs on a UNIX host.

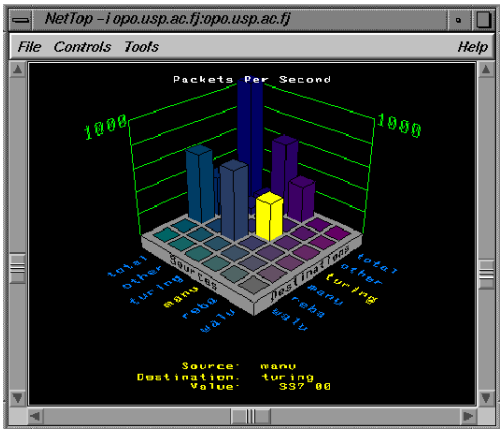


When running, **etherman** collects and displays graphically the ethernet traffic on the directly connected network. In the figure the display shows several hosts communicating with a range of protocols. A host to the bottom right of the display is generating a lot of traffic.

Etherman uses a visual metaphor associated with an easily understood model involving fluid, tanks and fluid flow. It gains leverage from human cognition of the behaviour of simple physical models.

[4]http://wwwhome.cs.utwente.nl/~schoenw/scotty/

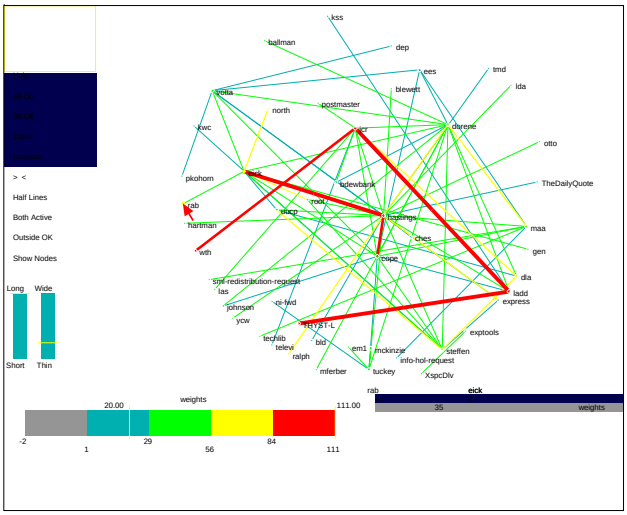
3D graph:



Nettop is a graphical display from SGI which indicates network traffic flow between systems. The display presents 3D bar graphs of network traffic. It can show the top sources and destinations of traffic on the network, or it can show the sources and destinations of your choice. It can also show the traffic on nodes, each with its filter.

Abstract 3D view - SeeNet:

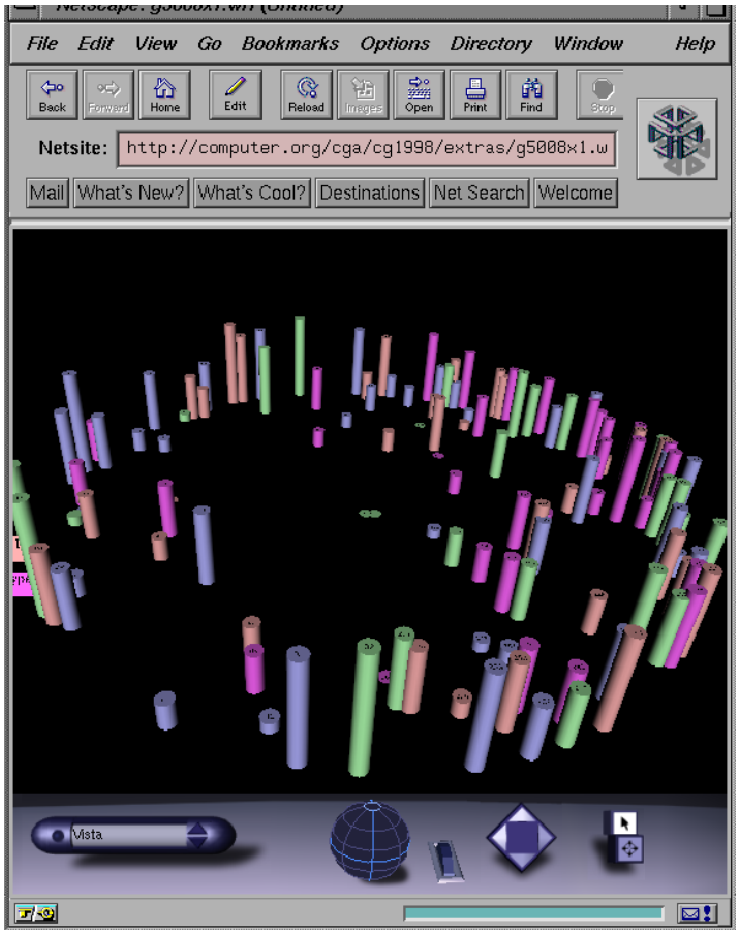
SeeNet was developed as part of a continual research effort in network data analysis at AT&T.



SeeNet is in daily use by AT&T engineers, and has many user interfaces - including this spring-tension 3D model, shown above. This figure shows an analysis of e-mail usage, and indicates that the user at the center (*Hastings*) is the e-mail *hub* of the department.

Abstract 3D view - Flodar platter display:

The **flodar** (Flow Radar) system [7] was developed at the National Security Agency (NSA) for continuous monitoring of large numbers of NSA servers. The designers have used a web based system for the display, interrogating a database that collects data asynchronously from remote agents.

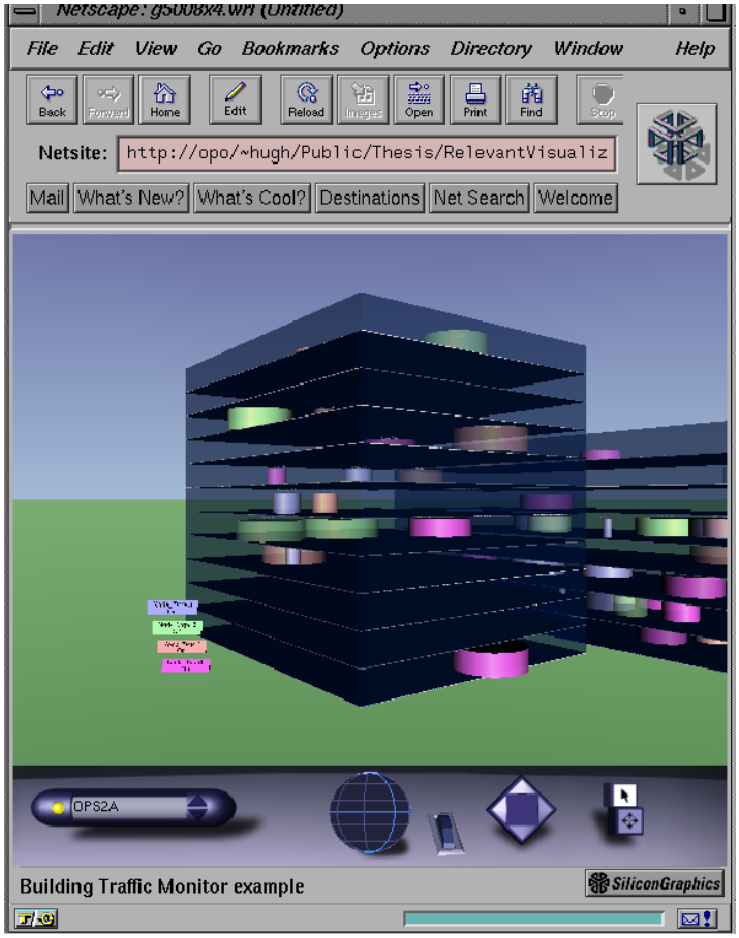


In the platter display, over a 24 hour period, cylinders representing servers move to the center of the platter. When a server signals, its cylinder moves to the outside of the platter. In this way servers that require attention move to the center of the platter.

The principal use of **flodar** is to alert operators to servers that have not signalled the database in a long time.

3D world-view:

In this example (again from the flodar system) we see a *building/locational* **view** - the servers are represented by cylinders. When the server signals the database, the cylinders are made nearly transparent. As the servers age, they become more opaque.



2.5 Summary of topics

In this module, we introduced the following topics:

- The designer's mindset
- Specification and design, tools and methods
- Examples of successful designs

Questions for Module 2

1. Give at least four other widgets not mentioned in section 2.2.2.
2. Give one other well-understood GUI action not mentioned in section 2.2.2.
3. Differentiate between radiobuttons and checkboxes. When would you use a radiobutton? When would you use a checkbox?
4. Describe how you might attempt to evaluate two competing designs.
5. Research: Study a visualization application which has an abstract view - evaluate the abstraction - is it successful or not?
6. Consider the 3D graph shown in the nettop application. Can you think of another use for this visualization?
7. Briefly outline a specification for a GUI application intended to manage a room booking system at NUS.
8. Briefly outline a specification for a visualization application intended to manage the flow of containers through the port in Singapore.

Further study

- Visualization:
  http://www2.iicm.edu/ivis/ivis.pdf.
- Formal specification:
  http://www.comp.nus.edu.sg/~cs3283/ftp/ObjectZToSpecifyWebInterface.ps.gz,
  http://www.cs.virginia.edu/~jck/publications/zum.97.pdf,
  http://www.comp.nus.edu.sg/~cs3283/ftp/SurveyOfUILanguages.ps.gz.
- Pressman [6] on UI design pp.395-406.

2.6 Sample assignment 2 - design/prototype

Task:

Your task is to develop the design of a GUI interface for a system for room booking at NUS. The system should provide for logging in, selecting a room or choice of rooms, a timeslot or choice of timeslots, submitting a request and displaying the results.

Deliverables:

- A title page containing your name and matriculation number.
- A five to ten page design document containing
  - A brief summary of the user requirement, and environment
  - An overview of the interface design
  - A detailed description of the interface design, including
    - Prototype screens
    - Functional specifications
    - Behavioural specifications
  - A testing methodology for the interface.

Note that this assignment does not require you to implement the application, just to design one.

Chapter 3

GUI application architecture

Modern GUI applications may be composed from a number of different software components. For example, a GUI application may access remote databases, or other machines, or it may be standalone. In this section we characterize some of the common software architectures for GUI applications.

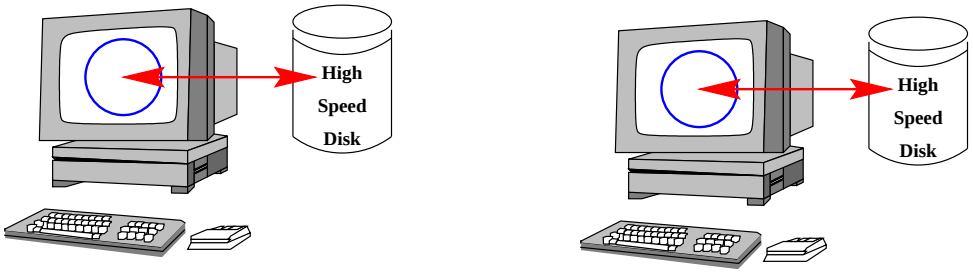
3.1 Architecture of GUI applications

We can categorise GUI applications in many different ways, but the overall communications architecture is of interest, and helps us select tools and development strategies. One classification is:

- Standalone
- Shared file
- Shared database
- Web based
  - Simple
  - Scripting
  - Java

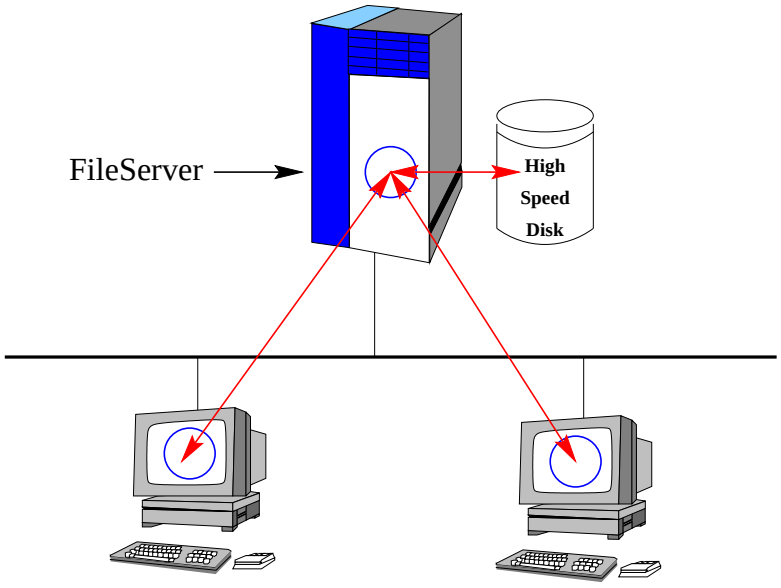
3.1.1 Standalone

A standalone GUI application runs on the user's PC, and reads and writes a local disk for files or (in a very general sense) databases. Note that Microsoft Access programs are normally of this sort.



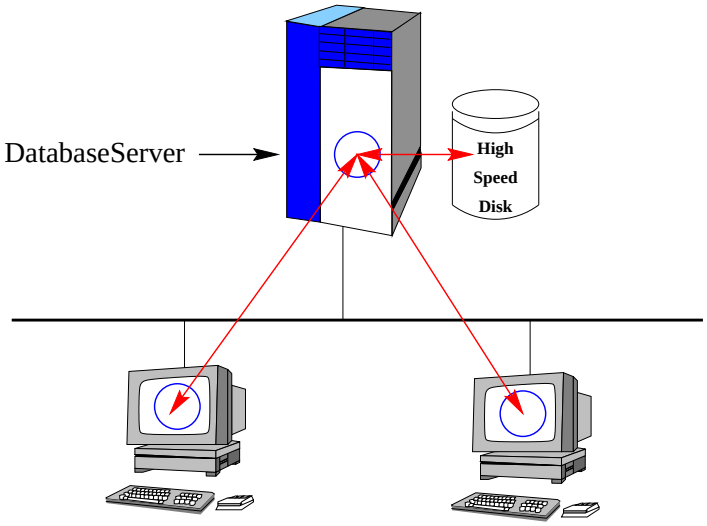
3.2 Shared file

A GUI application runs on the user's PC, and reads and writes a shared disk for files or (in a very general sense) databases. Note that Microsoft Access programs may be of this sort, but that Access files are generally not shareable.



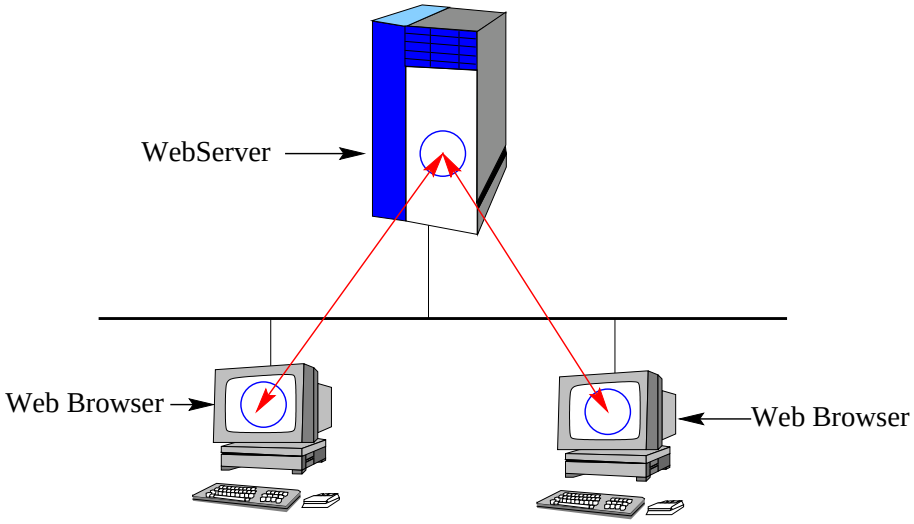
3.3 Shared database

A GUI application runs on the user's PC, and reads and writes a local database. Note that when your application uses a database like Oracle or Microsoft SQL server, it is likely to be of this sort.



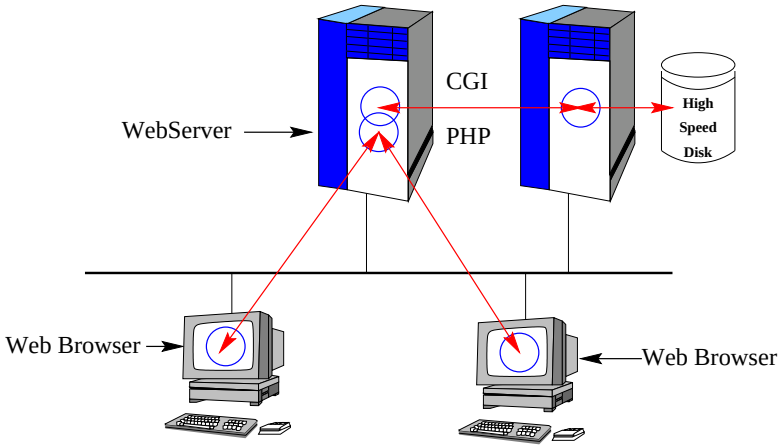
3.4 Web server applications

A VERY simple GUI application might be constructed using a series of interlinked web pages found on a server, and relying on a web browser on the client PCs. Help documentation applications are sometimes of this sort.



3.5 Web server with active scripting

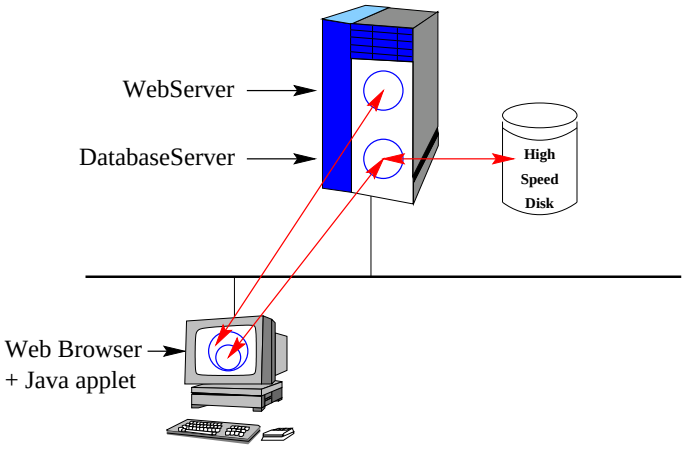
A more complex GUI application might be constructed using a series of interlinked web pages found on a server, and relying on a web browser on the client PCs. However, by using the Common Gateway Interface, or some other scheme of server-side scripting, we can return a program-derived web page to the application. CGI and PHP are both examples of this technique.



3.6 Web server with Java applet

An even more complex GUI application might be constructed using a series of interlinked web pages containing Java applets. The advantage of this, is two fold.

1. The processing load on the web server may be reduced.
2. The Java applet can directly[1] communicate with a database server.



[1]Note that there are some security concerns here.

3.7 Summary of topics

In this module, we introduced the following topic:

- GUI application architectures

Questions for Module 3

1. Characterize each of the programs you have written since you started studying at NUS.
2. Find an example of a site which is using PHP/MySQL with a large database. Give the URL, and a brief note on the site (size of database, type of user interface...).

Further study

- Pressman

Chapter 4

First steps in GUI programming

As we will discover, there is no one standard for GUI programming, although the techniques that you learn in one standard are generally transportable to another. In elementary programming styles, there is a single thread-of-control, which we can examine by reading the main(). This code determines how a user is expected to interact with the program. This general program architecture is satisfactory for small programs with simple command-line user interfaces.

However, graphical user interfaces have a much more complex thread-of-control. For example, at any time we may have a number of possible events about to occur - the user may ask for the window to be minimized, or resized, or click a button, or select a menu, or ...

Our programs must respond to each of these events. The normal way to do this is by restructuring our programs as a group of functions - each of which responds to an event. These functions are called *callbacks*.

Our GUI mainline code looks like this:

CODE LISTING **GUICode.c**

```
#include <any GUI header files needed>

int
main ()
{
    RegisterAllCallbacks ();
    LoopForever ();
}
```

An interesting area of GUI programming is the development of abstract windowed environments, where we program using an abstract API. These systems often allow the development of software which can be compiled for any environment.

4.1 How not to do GUI programming

Don’t do it the hard way!

4.1.1 Direct calls to the X API

It is possible to write applications that use the X APIs directly. These programs tend to be long (that is - they have a lot of source lines). Here is a simple application:

The source is as follows:

CODE LISTING **xaw1.c**

```
#include  <stdio.h>
#include  <X11/Intrinsic.h>
#include  <X11/StringDefs.h>
#include  <X11/Xaw/Command.h>
#include  <X11/Xaw/Paned.h>
#include  <X11/Xaw/Label.h>

void
quit_callback (widget, client_data, call_data)
     Widget widget;
     caddr_t client_data;
     caddr_t call_data;
{
    exit (0);
}

main (argc, argv)
     int argc;
     char *argv[];
{                                   /* main */
    Widget parent;
    Arg args[20];
    int n;
    Widget pane_widget, quit_widget;
    Widget label_widget;

    /* Set up top-level shell widget */
    parent = XtInitialize (argv[0], "Xaw1", NULL, 0, &argc, argv);
    /* Set up pane to control whole application */
    n = 0;
    pane_widget = XtCreateManagedWidget ("pane",
                                         panedWidgetClass, parent, args, n);
    /* Set up command widget to act as a push button */
    n = 0;
    quit_widget = XtCreateManagedWidget ("quit",
                                         commandWidgetClass,
                                         pane_widget, args, n);
    /* Set up a callback function */
    XtAddCallback (quit_widget, XtNcallback, quit_callback, (caddr_t) NULL);
    /* Set up label widget */
    n = 0;
    XtSetArg (args[n], XtNlabel, "This is a label.");
    n++;
    label_widget = XtCreateManagedWidget ("label",
                                          labelWidgetClass,
                                          pane_widget, args, n);
    /* Map widgets and handle events */
    XtRealizeWidget (parent);
    XtMainLoop ();
}
```

On a UNIX system we can compile this in the following way:

```
gcc -o xaw1 xaw1.c -lXt -lXaw
```

This programming technique is only included to demonstrate the underlying graphics primitives.

4.1.2 Direct calls to the Win32 API

It is also possible to write applications that use the Win32 API directly. These programs also tend to be long. We start with this program:

CODE LISTING **SimpleWin32.c**

```
#include <windows.h>

int STDCALL
WinMain (HINSTANCE hInst, HINSTANCE hPrev, LPSTR lpCmd, int nShow)
{
    MessageBox (NULL, "Hello, Windows!", "Hello", MB_OK);
    return 0;
}
```

This small example may be compiled using CYGWIN as follows:

```
gcc -oSimpleWin32 SimpleWin32.c -mwindows
```

And it produces the following application:

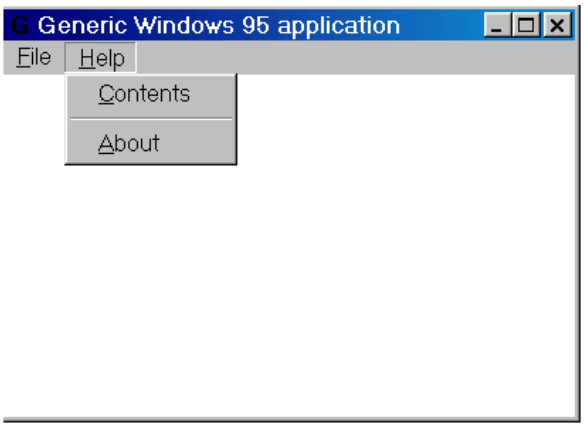
Another more complex example shows the flavour of raw Win32 programming, although I have removed about 380 lines for clarity:

CODE LISTING **BiggerWin.c**

```
#include <windows.h>
#include <string.h>

int STDCALL
WinMain (HINSTANCE hInst, HINSTANCE hPrev, LPSTR lpCmd, int nShow)
{
    HWND hwndMain;              /* Handle for the main window. */
    MSG msg;                    /* A Win32 message structure. */
    WNDCLASSEX wndclass;        /* A window class structure. */
    char *szMainWndClass = "WinTestWin";
    memset (&wndclass, 0, sizeof (WNDCLASSEX));
    wndclass.lpszClassName = szMainWndClass;
    wndclass.cbSize = sizeof (WNDCLASSEX);
    wndclass.style = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = MainWndProc;
    wndclass.hInstance = hInst;
    wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wndclass.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW);
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
    RegisterClassEx (&wndclass);
    hwndMain = CreateWindow (szMainWndClass, "Hello", WS_OVERLAPPEDWINDOW,
                             CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                             CW_USEDEFAULT, NULL, NULL, hInst, NULL);
    ShowWindow (hwndMain, nShow);
    UpdateWindow (hwndMain);
    while (GetMessage (&msg, NULL, 0, 0)) {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

The full source code and a makefile is available at http://www.comp.nus.edu.sg/~cs3283/ftp/generic.tgz.

When this is compiled, we get this:



Unfortunately, the full source code totals over 400 lines of C, and still doesn't actually do anything! If you are interested in learning more about Win32 programming, there is an interesting tutorial at http://www.winprog.org/tutorial.

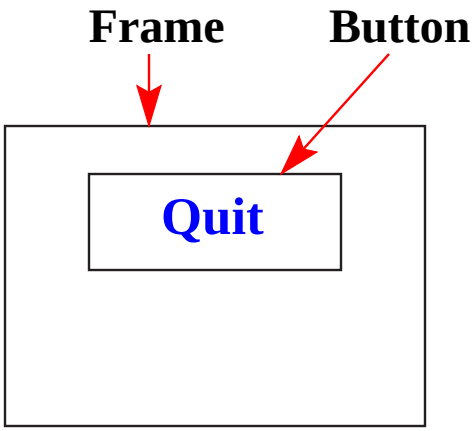


Figure 4.1: Nested GUI components

4.2 OO GUI toolkits

If we can reduce the size of code, we can be more assured that our code is correct. We are already using one mechanism to do this - we call Win32 or C functions which do quite complex things such as the call to **CreateWindow()** in the Win32 code above. A view of this might be that we are *hiding* the difficult parts from our application.

However, this sort of functional hiding has some flaws - our programs must maintain various sets of data representing our GUI environment, and can easily cause errors. Object-oriented technology provides a better mechanism, hiding the data from our programs - so that the only way in which the programs can modify the data is by using “approved” routines.

Unfortunately, there is no one object-oriented standard for GUI applications, leading to a fragmented situation - there are literally hundreds of commercial and free OO GUI framework/toolkits, with no one clear leader in the market place.

4.2.1 Event handling

When using OO GUI toolkits, the visible GUI part of the application often contains nested components as in Figure 4.1.

Each component may be the current focus of an event, and may have an event handler. If the inner component does not handle the event, then the event is passed up to the containing component.

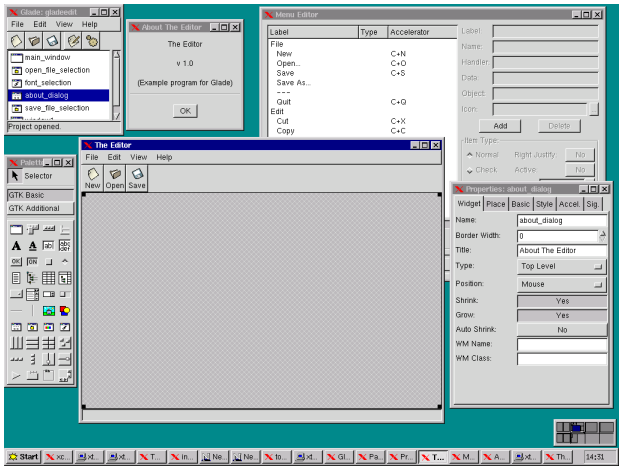
This general structure maps easily onto OO software architectures, when the innermost components are specializations (i.e. inherit from) the outer components. If you overide the event handling method in the inner component, then it will handle the event. If not, the parent component method handles the event.

4.2.2 GTK+ and glade

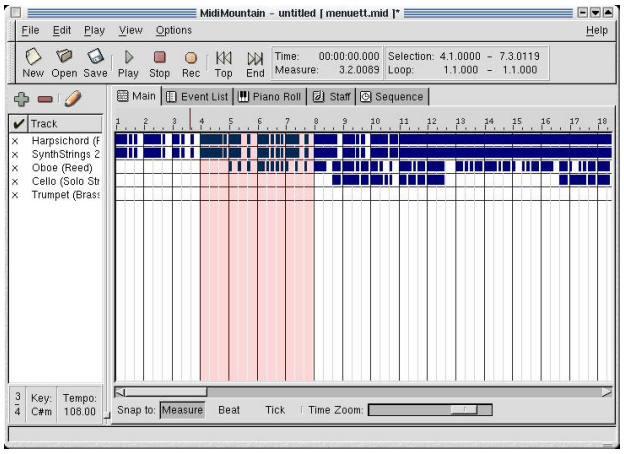
GTK+ is a multi-platform toolkit for creating graphical user interfaces, originally designed for the X Window System. However, by compiling using the CygWin GNU compiler, it is possible to use GTK+ on Win32. GTK+ is free software and part of the GNU Project. GTK+ has an object-oriented architecture for maximum flexibility, consisting of the following component libraries:

- GDK - A wrapper for low-level windowing functions.
- GTK - An advanced widget set.

One of the parts of the GTK+ distribution is a GUI builder called **glade**, which can build user interfaces very quickly.



Here is an example of an application built using **glade**:



4.2.3 MFC

The Microsoft Foundation Classes are an OO toolkit used to access Win32 system calls, and especially to construct GUI applications. A DLL contains the code for MFC, and is normally linked at runtime.

The base class is CObject, and all MFC classes inherit from this class.

One characteristic of MFC programs is the use of Hungarian (prefix) notation for variable names. It is common to see MFC program variables prefixed with type identifiers. For example:

- **dLocalMax** is a double variable
- **iLocalMin** is an integer variable.

4.2.4 Java/Swing

Originally the graphical toolkit for Java was AWT, the Abstract Windowing Toolkit. It is fairly primitive, and the new Swing toolkit provides much more functionality. AWT is native code, with a Java API, but Swing is implemented on-top-of AWT.

Swing components inherit from **java.awt.component**, and the Swing classes that are similar to AWT classes are prefixed with the letter "**J**". For example, the AWT **Button** class is renamed **JButton**. You can mix-and-match AWT and Swing components.

Java/Swing may be used in two distinct ways:

1. Producing a standalone application.
2. Producing an applet to run within a web browser.

One of the features of Swing is that it implements a pluggable look-and-feel. The look-and-feel can even be changed dynamically.

4.3 Web interfaces

It is possible to develop sophisticated applications with a web-based interface. We might divide the methods into the following categories:

- **Server-side dynamic pages** - using (for example) the CGI (Common Gateway Interface) to execute small programs or scripts on the server. This method is very common, and programs are typically written in perl.

- **Server-side scripting** - using (for example) PHP3 pages, and a specialized server which can interpret the PHP.
- **Client-side scripting** - using (for example) Javascript, and an interpreter on the client machine.
- **Client-side applets** - using (for example) a Java applet, precompiled and executed on a JVM interpreter on the client machine.

We will look at some of these methods later in the course.

4.4 Scripting languages

Scripting languages which can produce GUI interfaces are relatively easy to use. An effective strategy for building GUI applications is to write the GUI part in a scripting language, and to write the core 'difficult' part in C.

4.5 Summary of topics

In this module, we introduced the following topics:

- Programming styles to avoid
- Event driven architectures
- OO toolkits
- Web-based systems
- Scripting languages

Questions for Module 4

1. In the code listing on page 36 are a series of library calls to the **libXaw** library. In which call does the program spend the most time?
2. In this same code listing, draw the relationship between **parent**, **pane_widget**, **quit_widget** and **label_widget**.
3. In Figure 3.1, if a mouse was clicked over the **Button**, in what order would the event be processed by the **Frame** and **Button** event handlers?
4. What is a (Microsoft) DLL?
5. Find the code for a minimal Swing “HelloWorld” application (and check that it works).

Further study

- http://www.public.asu.edu/~tobiazz/papers/thesis/local/gui_toolkit_list.html

---

Chapter 5

Scripting language - Tcl/Tk

Scripting is difficult to define. It has existed for a long time - the first scripting languages were job control languages such as the shell program found in Unix systems. Modern scripting languages such as Perl, Tcl, Python, awk, Ruby and so on are general purpose, but often they have more powerful basic operations than those found in conventional general purpose computer languages. For example it is common to have operators that perform regular-expression pattern matching in a scripting language.

Scripting languages are normally interpreted, and the interpreter contains the routines to do the pattern matching. One line of script code may be equivalent to 100 lines of C. However, the overhead in having a (say) 3MB script interpreter is sometimes a problem, although less so these days.

Perl is widely used, as it is found in active web page developments. Tcl/Tk is useful for GUI development, allowing us to prototype new GUI applications quickly.

5.1 How not to use scripting languages

Don’t use to the exclusion of other languages!

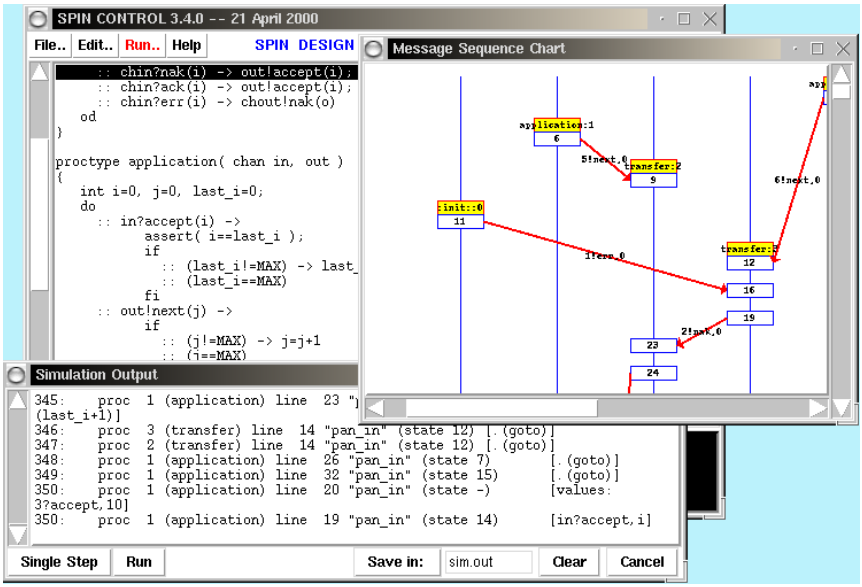
Scripting languages are very good at some things, but sometimes frustratingly bad at other things. For example, many scripting languages use associative, text-based array indexes, and so a simple array lookup may take 1000 times longer than an equivalent lookup in a compiled language.

For this reason, it is common to mix scripting and other languages.

5.2 Tcl/Tk

Wish - the windowing shell, is a simple scripting interface to the Tcl/Tk language. The language Tcl (Tool Command Language) is an interpreted scripting language, with useful inter-application communication methods, and is pronounced 'tickle'. Tk originally was an X-window toolkit implemented as extensions to 'tcl'. However, now it is available *native* on all platforms.

The program *xspin* is an example of a portable program in which the entire user interface is written in wish. The program also runs on PCs using NT or Win95, and as well on Macintoshes.



A first use of wish could be the following:

```
manu> wish
wish> button .quit -text "Hello World!" -command {exit}
.quit
wish> pack .quit
wish>
```

You can encapsulate this in a script:

CODE LISTING **HelloWorld.tcl**

```
#!/usr/local/bin/wish8.1 -f

button .quit -text "Hello World!" -command {exit}
pack .quit
```

If you create this as a file, and make it executable, you should be able to run this simple graphical program.

5.2.1 The structure of Tcl/Tk

The Tcl language has a tiny syntax - there is only a single *command* structure, and a set of rules to determine how to interpret the commands. Other languages have special syntaxes for control structures (if, while, repeat...) - not so in Tcl. All such structures are implemented as *commands*.

There is a runtime library of compiled 'C' routines, and the 'level' of the GUI interface is quite high.

Comments: If the first character of a command is #, it is a comment.

Tcl commands: Tcl commands are just words separated by spaces. Commands return strings, and arguments are just further words.

```
command argument argument
command argument
```

Spaces are important:

```
expr 5*3       has a single argument
expr 5 * 3     has three arguments
```

Tcl commands are separated by a new line, or a semicolon, and arrays are indexed by text:

```
set a(a\ text\ index) 4
```

Tcl/Tk quoting rules :

The "quoting" rules come in to play when the " or { character are first in the word. ".." disables a few of the special characters - for example space, tab, newline and semicolon, and {..} disables everything except \{, \} and \nl. This facility is particularly useful for the control structures - they end up looking very like 'C':

```
while {a==10} {
  set b [tst a]
}
```

Tcl/Tk substitution rules:

Variable substitution: The dollar sign performs the variable value substitution. Tcl variables are strings.

```
set a 12b     a will be "12b"
set b 12$a    b will be "1212b"
```

Command substitution: The []'s are replaced by the value returned by executing the Tcl command 'doit'.

```
set a [doit param1 param2]
```

Backslash substitution:

```
set a a\ string\ with\ spaces\ \
and\ a\ new\ line
```

Tcl/Tk command examples:

Procedures	File Access	Miscellaneous
`proc name {parameters} {body}`	`open <name>`	`source <NameOfFile>`
	`read <fileID>`	`global <varname>`
	`close <fileID>`	`catch <command>`
	`cd <directoryname>`	`format <formatstring> <value>`
		`exec <process>`
		`return <value>`

List operators:

```
split   <string> ?splitcharacters?
concat  <list> <list>
lindex  <list> <index>
... + lots more
```

Control structures:

```
if {test} {thenpart} {elsepart}[1]while {test} {body}
for {init} {test} {incr} {body}
continue
case $x in a {a-part} b {b-part}
```

[1]The Tcl/Tk words *then* and *else* are noise words, which may be used to increase readability.

Widget creation commands:

The first parameter to each is a 'dotted' name. The dot heirarchy indicates the relationships between the widgets.

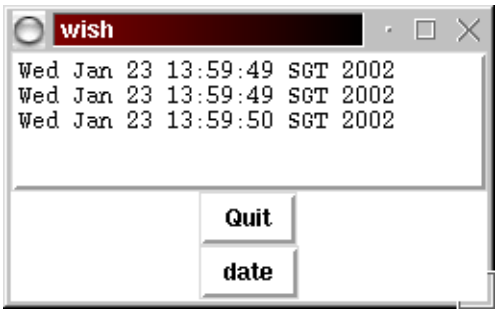
```
% label  <name>   -  optional parameter pairs ...
% canvas <name>   -  optional parameter pairs ...
% button <name>   -  optional parameter pairs ...
% frame  <name>   -  optional parameter pairs ...
% ... and so on
```

When you create a widget ".b", a new command ".b" is created, which you can use to further communicate with it. The geometry managers in Tk assemble the widgets:

```
% pack <name> .... where ....
```

5.2.2 Tcl/Tk example software

Here is a very small Tcl/Tk application, which displays the date in a scrollable window:

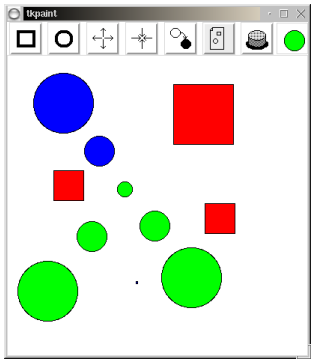


The code for this is:

CODE LISTING **SimpleProg.tcl**

```
#!/usr/local/bin/wish8.1 -f

text .log -width 60 -height 5 -bd 2 -relief raised
pack .log
button .buttonquit -text "Quit" -command exit
pack .buttonquit
button .buttondate -text "date" -command getdate
pack .buttondate
proc getdate {} {
   set result [exec date]
   .log insert end $result
   .log insert end \n
}
```

Here is tkpaint - a drawing/painting program written in Tcl/Tk:



The mainline of the source just creates the buttons, and packs the frame:

CODE LISTING **tkpaint1.tcl**

```
#! /usr/local/bin/wish -f

set thistool rectangle
set thisop grow
set thiscolour black
button .exitbtn   -bitmap @exit.xbm      -command exit
button .squarebtn -bitmap @square.xbm    -command setsquaretool
button .circlebtn -bitmap @circle.xbm    -command setcircletool
button .shrnkbtn  -bitmap @shrink.xbm    -command "set thisop shrnk"
button .growbtn   -bitmap @grow.xbm      -command "set thisop grow"
button .printbtn  -bitmap @print.xbm     -command printit
button .colorbtn -bitmap @newcolour.xbm -command setanewcolour
canvas .net -width 400 -height 400 -background white -relief sunken
canvas .status -width 40 -height 40 -background white -relief sunken

pack .net -side bottom
pack .status -side right
pack .squarebtn .circlebtn -side left -ipadx 1m -ipady 1m -expand 1
pack .exitbtn .printbtn -side right -ipadx 1m -ipady 1m -expand 1
pack .colorbtn .shrnkbtn .growbtn -side right -ipadx 1m -ipady 1m -expand 1

bind .net <ButtonPress-1> {makenode %x %y}
.status create rectangle 10 10 37 37 -tag statusthingy -fill $thiscolour
set nodes 0; set oldx 0; set oldy 0;
```

Routines for dragging, scaling and printing:

CODE LISTING **tkpaint4.tcl**

```
proc beginmove {x y} {
    global oldx oldy
    set oldx $x; set oldy $y
}

proc domove {item x y} {
    global oldx oldy
    .net move $item [expr "$x - $oldx"] [expr "$y - $oldy"]
    set oldx $x; set oldy $y
}

proc altersize {item x y z} {
    .net scale $item $x $y $z $z
}

proc printit {} {
    .net postscript -file "pic.ps"
}
```

Node operations for tkpaint:

CODE LISTING **tkpaint2.tcl**

```
proc makenode {x y} {
   global nodes oldx oldy thistool thiscolor
   set nodes [expr "$nodes+1"]

   set x1 [expr "$x-20"]; set y1 [expr "$y-20"]
   set x2 [expr "$x+20"]; set y2 [expr "$y+20"]
   if {[string compare $thistool "oval"] == 0} {
      .net create oval $x1 $y1 $x2 $y2 -tag node$nodes -fill $thiscolor
   }
   if {[string compare $thistool "rectangle"] == 0} {
      .net create rectangle $x1 $y1 $x2 $y2 -tag node$nodes -fill $thiscolor
   }
   .net bind node$nodes <Enter> ".net itemconfigure node$nodes -width 5"
   .net bind node$nodes <Leave> ".net itemconfigure node$nodes -width 1"
   .net bind node$nodes <ButtonPress-3> "beginmove %x %y"
   .net bind node$nodes <B3-Motion> "domove node$nodes %x %y"
   .net bind node$nodes <ButtonPress-2> "dothisop node$nodes %x %y"
}

proc dothisop {item x y} {
   global thisop
   if {[string compare $thisop "shrink"] == 0} {
       altersize $item $x $y 0.5
   }
   if {[string compare $thisop "grow"] == 0} {
       altersize $item $x $y 2.0
   }
}
```

More routines:

CODE LISTING **tkpaint3.tcl**

```
proc setcircletool {} {
    global thistool thiscolor
    set thistool oval
    .status delete statusthingy
    .status create oval 10 10 37 37 -tag statusthingy -fill $thiscolor
}

proc setsquaretool {} {
    global thistool thiscolor
    set thistool rectangle
    .status delete statusthingy
    .status create rectangle 10 10 37 37 -tag statusthingy -fill $thiscolor
}

proc setanewcolor {} {
    global thiscolor

    if {[string compare $thiscolor "black"] == 0} {
            set thiscolor green
    } { if {[string compare $thiscolor "green"] == 0} {
            set thiscolor blue
        } { if {[string compare $thiscolor "blue"] == 0} {
                set thiscolor red
            } { if {[string compare $thiscolor "red"] == 0} {
                    set thiscolor orange
                } { set thiscolor black }
              }
          }
      }
    .status itemconfigure statusthingy -fill $thiscolor
}
```

5.2.3 C/Tk

In the following example, a Tcl/Tk program is integrated with a C program, giving a very small codesize GUI application, that can be compiled on any platform - Windows, UNIX or even the Macintosh platform without changes.

CODE LISTING **CplusTclTk.c**

```
#include <stdio.h>
#include <tcl.h>
#include <tk.h>

char tclprog[] = "\
      proc fileDialog {w} {\
        set types {\
          { \"Image files\" {.gif} }\
          { \"All files\" *}\
        };\
        set file [tk_getOpenFile −filetypes $types −parent $w];\
        image create photo picture −file $file;\
        set glb_tx [image width picture];\
        set glb_ty [image height picture];\
        .c configure −width $glb_tx −height $glb_ty;\
        .c create image 1 1 −anchor nw −image picture −tags \"myimage\";\
      };\
      frame .mbar −relief raised −bd 2;\
      frame .dummy −width 10c −height 0;\
      pack .mbar .dummy −side top −fill x;\
      menubutton .mbar.file −text File −underline 0 −menu .mbar.file.menu;\
      menu .mbar.file.menu −tearoff 1;\
      .mbar.file.menu add command −label \"Open...\" −command \"fileDialog .\";\
      .mbar.file.menu add separator;\
      .mbar.file.menu add command −label \"Quit\" −command \"destroy .\";\
      pack .mbar.file  −side left;\
      canvas .c −bd 2 −relief raised;\
      pack .c −side top −expand yes −fill x;\
      bind . <Control−c> {destroy .};\
      bind . <Control−q> {destroy .};\
      focus .mbar";

int
main (argc, argv)
     int argc;
     char **argv;
{
    Tk_Window mainWindow;
    Tcl_Interp *tcl_interp;

    setenv ("TCL_LIBRARY", "/cygnus/cygwin−b20/share/tcl8.0");
    tcl_interp = Tcl_CreateInterp ();
    if (Tcl_Init (tcl_interp) != TCL_OK || Tk_Init (tcl_interp) != TCL_OK) {
        if (*tcl_interp−>result)
            (void) fprintf (stderr, "%s: %s\n", argv[0], tcl_interp−>result);
        exit (1);
    }
    mainWindow = Tk_MainWindow (tcl_interp);
    if (mainWindow == NULL) {
        fprintf (stderr, "%s\n", tcl_interp−>result);
        exit (1);
    }

    Tcl_Eval (tcl_interp, tclprog);
    Tk_MainLoop ();

    exit (1);
}
```

The first half of the listing is a C string containing a Tcl/Tk program. The second part of the listing is C code which uses this Tcl/Tk.

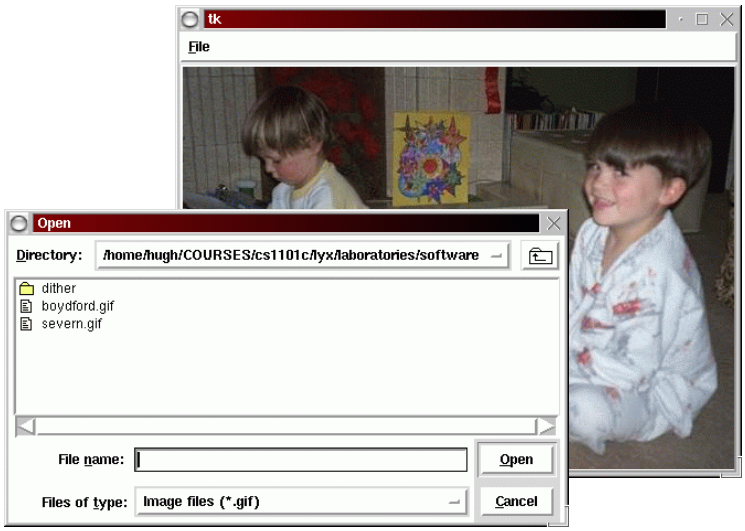
On a Win32 system, we compile this as:

```
gcc -o CplusTclTk CplusTclTk.c -mwindows -ltcl80 -ltk80
```

On a UNIX system we use:

```
gcc -o CplusTclTk CplusTclTk.c -ltk -ltcl -lX11 -lm -ldl
```

And the result is a simple viewer for GIF images. The total code size is 57 lines. The application looks like this when running:



5.3 Summary of topics

In this module, we introduced the following topics:

- Practical programming in Tcl/Tk
- Other Tk language bindings
- Some sample programs

Questions for Module 4

1. Given the frame **.frm** containing a canvas and a quit button, give sensible names for the canvas and the button.
2. Modify **SimpleProg.tcl** to have an extra button **clear** above the **quit** button which clears the date display.
3. Modify **SimpleProg.tcl** to have an extra button **clear** to the left of the **quit** button which clears the date display.
4. What is the effect of the following tcl command? **set a [exec ls]**
5. What is the effect of the following tcl command? **set a expr 3 + 4**
6. Write a minimal Tk application which puts up a single **File** menu with a **Quit** item in it.

Further study

- http://www.pconline.com/~erc/tclwin.htm
- http://tcl.activestate.com/scripting/
- http://www.msen.com/~clif/TclTutor.html

Chapter 6

Introduction to Java/Swing

Java is commonly used for deploying applications across a network. Compiled Java code may be distributed to different machine architectures, and a native-code interpreter on each architecture interprets the Java code. The core functions found in the Java interpreter are called the JFC (Java Foundation Classes). JFC provides generally useful classes, including classes for GUIs, accessability and 2D drawing. The original GUI classes in Java are known as AWT - the Abstract Windowing Toolkit. AWT provides basic GUI functions such as buttons, frames and dialogs, and is implemented in native code in the Java interpreter.

By contrast, Swing is not implemented in native code - instead it is implemented in AWT. Swing and AWT can (and normally do) coexist - we may use the buttons from Swing, alongside AWT event handlers.

The advantages of Swing are:

1. Consistent look-and-feel - The look and feel is consistent across platforms.
2. Pluggable look-and-feel - The look and feel can be switched on-the-fly.
3. High-level widgets - the Swing components are useful and flexible.

In general, the Swing components are easier to use than similar AWT components.

6.1 How not to use Swing

The same concerns that applied to Tcl/Tk deployment apply to the use of Swing. If the target computers are slow, then the interpreter overhead may make the application frustratingly slow. With the rate of increase in speed in processors, this concern is minimized.

Processor intensive applications written in Java often seem to make the GUI appear slow and unresponsive. This is probably due to internal thread scheduling techniques in the interpreter.

6.2 Getting started

There are quite a few development environments for building Java applications and applets, and two of them are suggested for use in this course. However - if you have something better, or that you feel more comfortable with, please just use that. The systems are:

- **j2sdk1.3.1** - the Java development kit from Sun. It includes Java compilers, interpreters, debuggers and demo software, and local copies of it for WinXX and LINUX are found here:
  - http://www.comp.nus.edu.sg/~cs3283/ftp/Java/j2sdk-1_3_1_02-win.exe
  - http://www.comp.nus.edu.sg/~cs3283/ftp/Java/j2sdk-1_3_1_02-linux-i386.bin
- **Netbeans** - A GUI builder for Java applications and applets, again for WinXX and LINUX:
  - http://www.comp.nus.edu.sg/~cs3283/ftp/Java/NetBeansIDE-release331.exe
  - http://www.comp.nus.edu.sg/~cs3283/ftp/Java/NetBeansIDE-release331.tar.gz

Each of these systems is documented and described at public web sites - look at Sun's Java web site, and http://www.netbeans.org. In addition - there are local copies of some of the documentation here:

- The **JFC API** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/jfcapi/
- The Netbeans **API** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/OpenAPIs/
- The **Java tutorial** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/JavaTutorial/
- **Swing Connect** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/swingConnect/

Once you have installed the j2sdk, find the file called SwingSet2.jar, inside the demo heirarchy somewhere, and change to the directory. Then try:

```
java -jar SwingSet2.jar
```

6.3 Swing programming

In this course I hope to clarify the general style of Swing applications, and show sufficient examples to build *menu'd* GUI applications with interesting graphical interactions. The same strategy was used in the introduction to Tcl/Tk. A good book that covers this material in detail is

The JFC Swing Tutorial, by Kathy Walrath and Mary Campione.

The toplevel components provided by Swing are:

1. **JApplet** - for applets within web pages
2. **JDialog** - for dialog boxes
3. **JFrame** - for building applications

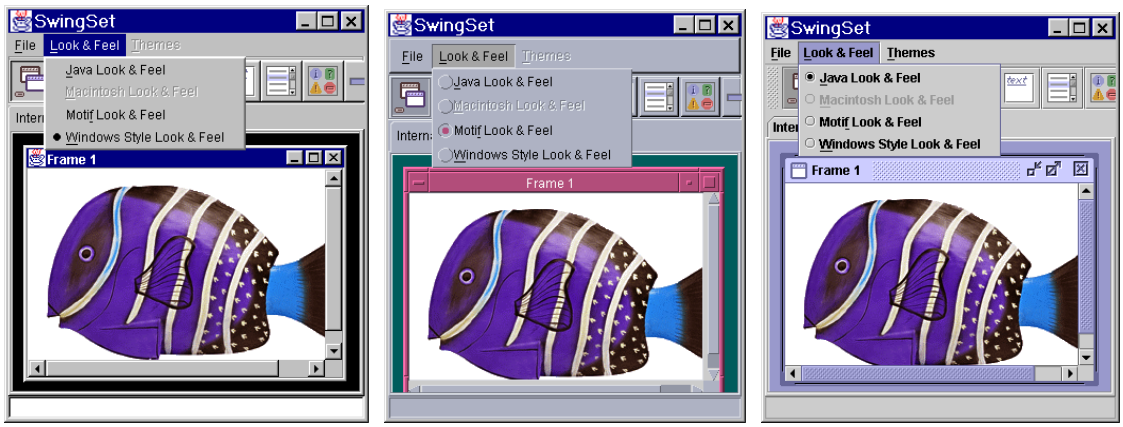
All other Swing components derive from the **JComponent** class. **JComponent** provides

- **Tool tips** - little windows with explanations
- **Pluggable look and feel** - as described
- **Layour management** - items within the component
- **Keyboard action management** - Hot keys and so on.
- And other facilities

Swing implements an MVC architecture.

6.3.1 Pluggable look and feel

It is relatively easy to change the look and feel of an application - here are three:



If you wished to use the WinXX look-and-feel, in the main of your application, you can make the following call:

```
UIManager.setLookAndFeel( "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
```

6.4 Example application

It is traditional to begin with a “Hello World” example, but I will start with “Hello Singapore”, and you will have to move up to “Hello World” as you progress.

CODE LISTING **t2.java**

```
public class t2 extends javax.swing.JFrame {
    public t2() {
        initComponents();
    }
    private void initComponents() {
        jLabel2 = new javax.swing.JLabel();
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });
        jLabel2.setText("Hello Singapore!");
        jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        getContentPane().add(jLabel2, java.awt.BorderLayout.CENTER);
        pack();
    }
    private void exitForm(java.awt.event.WindowEvent evt) {
        System.exit(0);
    }
    public static void main(String args[]) {
        new t2().show();
    }
    private javax.swing.JLabel jLabel2;
}
```

This code should not require much explanation - it just instantiates a **JLabel**, and sets the text field. Perhaps the only explanation needed is why it is so large! The code is generated from a GUI builder, and follows a particular software architecture. In this presentation of Swing, I will use the same, despite the possibility of smaller code-size applications.

When we compile and run this application we get:

The call to **getContentPane** returns the **contentPane** object for the frame - this is a generic AWT container for components associated with each **JFrame**. The **addWindowListener** call is from **java.awt.Window**, and adds the specified window listener to receive window events from this window.

6.5 Example applet

An equivalent Hello-world applet:

CODE LISTING **HelloWorldApp.java**

```
public class HelloWorldApp extends javax.swing.JApplet {
    public HelloWorldApp() {
        initComponents();
    }
    private void initComponents() {
        jLabel1 = new javax.swing.JLabel();
        jLabel1.setText("Hello Singapore!");
        jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        getContentPane().add(jLabel1, java.awt.BorderLayout.CENTER);
    }
    private javax.swing.JLabel jLabel1;
}
```

This code follows the same structure - it just instantiates a **JLabel**, and sets the text field, although in this code, the class extends a **JApplet** instead of a **JFrame**. When we compile and run this application we get a **HelloWorldApp.class** file, which has to be referenced in a web page:

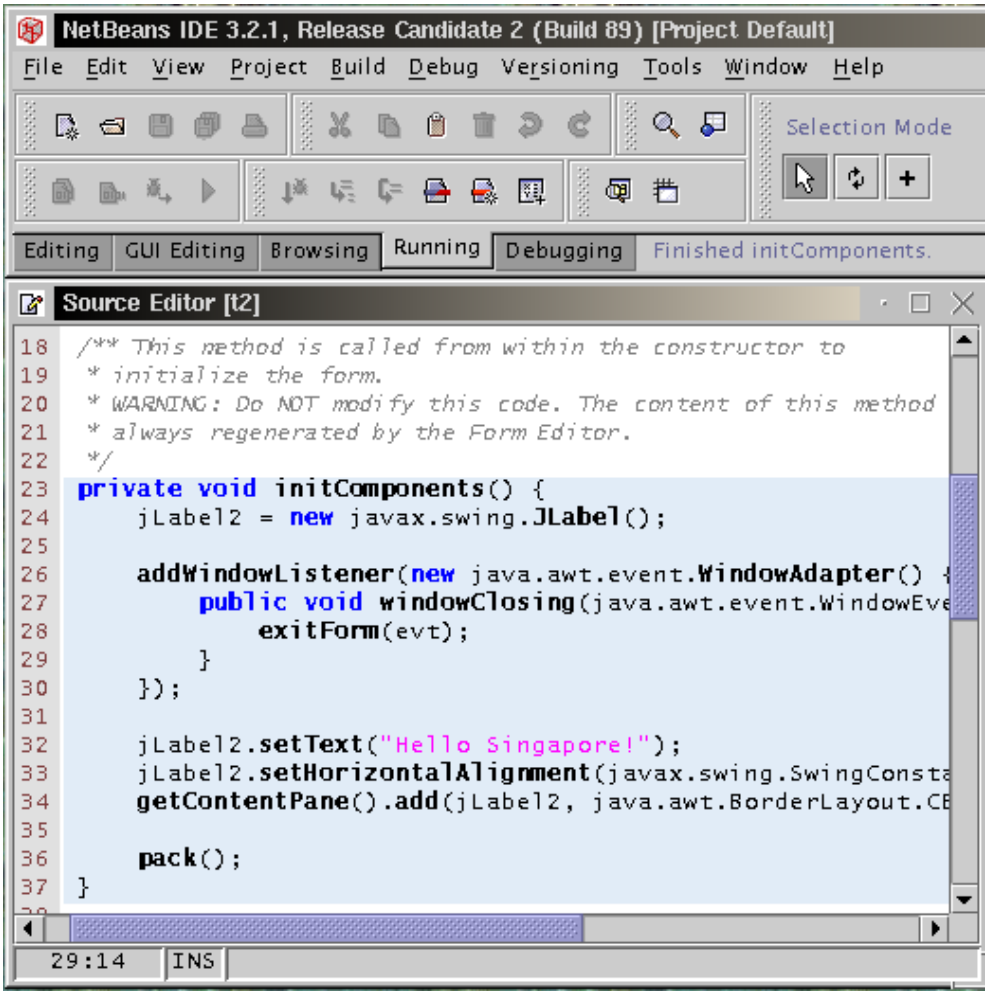
CODE LISTING **HelloWorldApp.txt**

```
<BASE HREF="http://www.comp.nus.edu.sg/~hugh/swing/">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
The HelloWorld Applet <p>
<EMBED type         = "application/x-java-applet;version=1.1.2"
       java_CODE    = "HelloWorldApp.class"
       java_ARCHIVE = "applets.jar"
       WIDTH        = 400
       HEIGHT       = 50   ></EMBED>
</HTML>
```

The end result is:

6.6 Using the netbeans IDE

Simple programs like the ones just presented may be created using the GUI builder found in **net-beans** (http://www.netbeans.org), using a very small number of button presses and keystrokes. Here is a screen shot:



6.7 Summary of topics

In this module, we introduced the following topics:

- Tool sets for Java/Swing
- The relationship between JFC, Java and Swing.
- Simple first programs

Questions for module 5

1. What is meant by "*the MVC architecture*" mentioned in section 5.3?
2. Investigate how you would create a "*ToolTip*" in Tcl/Tk - give a small code segment which demonstrates the *ToolTip*.
3. Investigate how you would create a "*ToolTip*" in Java/Swing - give a small code segment which demonstrates the *ToolTip*.
4. Write a minimal Java/Swing application which puts up a single **File** menu with a **Quit** item in it.
5. The **javax.swing.UIManager** class is used to manipulate the look-and-feel of an application - as seen in section 5.3.1. How can you discover which look-and-feel strategies are implemented in the Java development environment?

Further study

- The **JFC API** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/jfcapi/
- The Netbeans **API** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/OpenAPIs/
- The **Java tutorial** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/JavaTutorial/
- **Swing Connect** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/swingConnect/

Chapter 7

Java continued ...

We continue with our study of Java/Swing by looking at various semi-related topics, but be reminded that the notes given here are very brief, and should be supplemented by studying the examples and explanations found in the Java Tutorial on the web site.

Visible and invisible Java/Swing elements are found in a containment heirarchy. At the top level we have containers for the different types of application (i.e. an applet, or an application, or a dialog). In a middle level we have the panes, and at a lower level the individual components.

Level	**Container**
Top-level	JFrame JApplet JDialog
Mid-level	JPanel JScrollBar JTabbedPane
Component-level	JButton JLabel ...

Every GUI component must be part of a containment hierarchy[1]. Each top-level container has a content pane, and an optional menu bar, and Java/Swing components are added to either the content pane or the menu bar. Every component must be placed somewhere in this containment heirarchy, or it will not be visible.

[1]To view the containment hierarchy for any frame or dialog, click its border to select it, and then press Control-Shift-F1. A list of the containment hierarchy will be written to the standard output stream.

7.1 Layout management

Every container has a default layout manager, which may be over-ridden with your own if for some reason the existing one is unsatisfactory. The Java platform supplies a range of layout managers, but here we will just look briefly at three. Note that these are AWT components, not Swing .

7.1.1 BorderLayout

BorderLayout is the default layout manager for every content pane, and assists in placing components in the north, south, east, west, and center of the content pane.

```
contentPane.add(new JButton("B1"), BorderLayout.NORTH);
```

7.1.2 BoxLayout

BoxLayout puts components in a single row or column. Here is code to create a centered column of components:

```
pane.setLayout(new BoxLayout(pane, BoxLayout.Y_AXIS));
pane.add(label);
pane.add(Box.createRigidArea(new Dimension(0,5)));
pane.add(...);
```

7.1.3 CardLayout

CardLayout is for when a pane has different components at different times. You may think of it as a stack of same-sized cards.

```
cards = new JPanel();
cards.setLayout(new CardLayout());
cards.add(p1, BUTTONPANEL);
cards.add(p2, TEXTPANEL);
```

You can choose the top card to show:

```
CardLayout cl = (CardLayout)(cards.getLayout());
cl.show(cards, (String)evt.getItem());
```

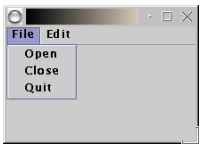
7.2 Creating menus

The menu classes are descendants of **JComponent**, and may be used in any higher-level container class (**JApplet** and so on). Here is a small example of a simple menu application, given in the *netbeans* program style:

CODE LISTING **menutest.java**

```
public class menutest extends javax.swing.JFrame {
    public menutest() {
        initComponents();
    }
    private void initComponents() {
        jMenuBar1  = new javax.swing.JMenuBar();
        jMenu1     = new javax.swing.JMenu();
        jMenuItem1 = new javax.swing.JMenuItem();
        jMenuItem2 = new javax.swing.JMenuItem();
        jMenuItem3 = new javax.swing.JMenuItem();
        jMenu2     = new javax.swing.JMenu();
        jMenuItem4 = new javax.swing.JMenuItem();
        jMenu1.setText("File");
        jMenuItem1.setText("Open");
        jMenu1.add(jMenuItem1);
        jMenuItem2.setText("Close");
        jMenu1.add(jMenuItem2);
        jMenuItem3.setText("Quit");
        jMenuItem3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem3ActionPerformed(evt);
            }
        });
        jMenu1.add(jMenuItem3);
        jMenuBar1.add(jMenu1);
        jMenu2.setText("Edit");
        jMenuItem4.setText("Cut");
        jMenu2.add(jMenuItem4);
        jMenuBar1.add(jMenu2);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });
        setJMenuBar(jMenuBar1);
        pack();
    }

    private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
        System.exit(0);
    }
    private void exitForm(java.awt.event.WindowEvent evt) {
        System.exit(0);
    }
    public static void main(String args[]) {
        new menutest().show();
    }
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JMenuItem jMenuItem2;
    private javax.swing.JMenuItem jMenuItem3;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenuItem jMenuItem4;
}
```

The end result is:



7.3 Threads in Swing

Java supports a multi-threading mechanism that is sometimes useful. User programs and applets may create several threads to manage different parts of the application. As in any multi-threaded system, we may have critical sections if two threads attempt to access the same variables at the same time. To create threads there are some helpful classes such as **SwingWorker** or **Timer**.

Most Swing components are not thread safe - this means that if two threads call methods on the same Swing component, the results are not guaranteed. The single-thread rule:

Swing components can be accessed by only one thread at a time.

A particular thread, the event-dispatching thread, is the one that normally accesses Swing components. To get access to this thread from another thread we can use **invokeLater()** or **invokeAndWait()**.

7.3.1 Creating threads

Many applications do not require threading, but if you do have threads, then you may have problems debugging your programs. However, you might consider using threads if:

- Your application has to do some long task, or wait for an external event, without freezing the display.
- Your application has to do someting at fixed time intervals.

The following two classes are used to implement threads:

1. **SwingWorker**[2]: To create a thread
2. **Timer**: Creates a timed thread

To use **SwingWorker**, create a subclass of it, and in the subclass, implement your own **construct()** method. When you instantiate the **SwingWorker** subclass, the runtime environment creates a thread but does not start it. The thread starts when you invoke **start()** on the object.

Here's an example of using **SwingWorker** from the tutorial - an image is to be loaded over a network (given a URL). This may of course take quite a while, so we don't block our main thread - (if we did this, the GUI may freeze).

[2] If you find that your distribution does not include SwingWorker.class, download and compile it.

The following code shows the better way of loading the remote image:

CODE LISTING **ImageLoader.java**

```
private void loadImage(final String imagePath,
                       final int index) {
     final SwingWorker worker = new SwingWorker() {
         ImageIcon icon = null;
         public Object construct() {
             icon = new ImageIcon(getURL(imagePath));
             return icon;
         }
         public void finished() {
             Photo pic = (Photo)pictures.elementAt(index);
             pic.setIcon(icon);
             if (index == current)
                 updatePhotograph(index, pic);
             }
         };
     worker.start();
}
```

The **Timer** class is used to repeatedly perform an operation. When you create a **Timer**, you specify its frequency, and you specify which object is the listener for its events. Once you start the timer, the action listener's **actionPerformed()** method will be called for each event.

7.3.2 Event dispatching thread

The event-dispatching thread is the main event-handling thread. It is normal for all GUI code to be called from this main thread, even if some of the code may take a long time to run. However - we have already mentioned that we should not delay the event-dispatching thread.

Swing provides a solution to this - the **InvokeLater()** method may be used to safely run code in the event-dispatching thread. The method requests that some code be executed in the event-dispatching thread, but returns immediately, without waiting for the code to execute.

```
Runnable doWorkRunnable = new Runnable() {
    public void run() { doWork(); }
};
SwingUtilities.invokeLater(doWorkRunnable);
```

7.4 Handling events

Actions associated with Java/Swing components raise events - moving the mouse or clicking a JButton all cause events to be raised. The application program writes a listener method to process an event, and registers it as an event listener on the event source. There are different kinds of events, and we use different kinds of listener to act on them. For example:

Action	Listener type
Button click	ActionListener
A window closes	WindowListener
Mouse click	MouseListener
Mouse moves	MouseMotionListener
Component becomes visible	ComponentListener
Keyboard focus	FocusListener
List selection changes	ListSelectionListener

The listener methods are passed an event object which gives information about the event and identifies the event source.

7.4.1 Event handlers

When you write an event handler, you must do the following:

- Specify a class that either implements a listener interface or extends a class that implements a listener interface.

```
public class MyClass implements ActionListener { ...
```

- Register an instance of the class as a listener upon the components.

```
Component.addActionListener(instanceOfMyClass);
```

- Implements the methods in the listener interface.

```
public void actionPerformed(ActionEvent e) {
    ...//code that reacts to the action...
}
```

Make sure that your event handler code executes quickly, or your program may seem to be slow. In the sample code given so far, we have used window listeners to react if someone closes a window, but not to capture other sorts of events.

7.4.2 Handling events

CODE LISTING **CheckBoxDemo.java**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CheckBoxDemo extends JPanel {
    JCheckBox chinButton;
    JCheckBox glassesButton;
    StringBuffer choices;
    JLabel pic;
    public CheckBoxDemo() {
        chinButton = new JCheckBox("Chin");
        glassesButton = new JCheckBox("Glasses");
        CheckBoxListener myListener = new CheckBoxListener();
        chinButton.addItemListener(myListener);
        glassesButton.addItemListener(myListener);
        choices = new StringBuffer("--ht");
        pic = new JLabel(new ImageIcon( "geek-" + choices.toString() + ".gif"));
        pic.setToolTipText(choices.toString());
        JPanel checkPanel = new JPanel();
        checkPanel.setLayout(new GridLayout(0, 1));
        checkPanel.add(chinButton);
        checkPanel.add(glassesButton);
        setLayout(new BorderLayout());
        add(checkPanel, BorderLayout.WEST);
        add(pic, BorderLayout.CENTER);
        setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    }
    class CheckBoxListener implements ItemListener {
        public void itemStateChanged(ItemEvent e) {
            int index = 0;
            char c = '-';
            Object source = e.getItemSelectable();
            if (source == chinButton) {
                index = 0;
                c = 'c';
            } else if (source == glassesButton) {
                index = 1;
                c = 'g';
            }
            if (e.getStateChange() == ItemEvent.DESELECTED)
                c = '-';
            choices.setCharAt(index, c);
            pic.setIcon(new ImageIcon( "geek-" + choices.toString() + ".gif"));
            pic.setToolTipText(choices.toString());
        }
    }
    public static void main(String s[]) {
         JFrame frame = new JFrame("CheckBoxDemo");
         frame.addWindowListener(new WindowAdapter() {
             public void windowClosing(WindowEvent e) {
                 System.exit(0);
             }
         });
         frame.setContentPane(new CheckBoxDemo());
         frame.pack();
         frame.setVisible(true);
    }
}
```

Here is an example of event handling code, simplified from the tutorial. It displays a small graphic, and has two checkboxes. When you change either checkbox, an **itemListener** responds to the event and changes the graphic.

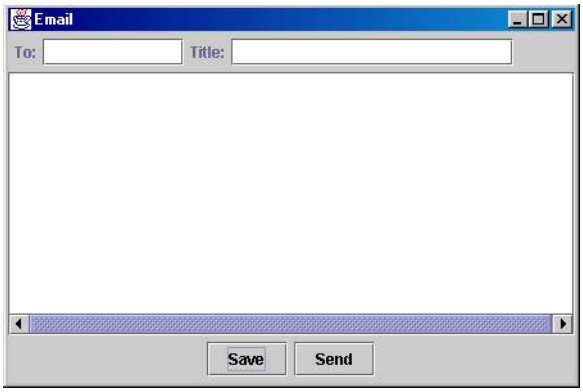
7.5 Summary of topics

In this module, we introduced the following topics:

- The containment heirarchy
- Layout managers
- Menus
- Threading
- Event handling

Questions for module 5

1. Research the root pane that comes with every highest level container in Java Swing. Briefly describe each of its componenets and state what each could be used for.
2. Give layout management code for the following:



3. Give code for a small menu-style application which makes the console beep whenever a menu item is selected.

Further study

- The **Java tutorial** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/JavaTutorial/
- **Swing Connect** at http://www.comp.nus.edu.sg/~cs3283/ftp/Java/swingConnect/

Chapter 8

Web interfaces

A common feature of many modern GUI applications is that there is a strong desire to deliver the applications via web browsers. MSIE and Navigator have differences in their implementation of (what should be) standard extensions such as Java, and so any web based developments need to be tested on any target platforms.

However - its not really that hard to write CGI/ PHP and/or Java applications that work on all platforms. In this section we will briefly look at CGI, PHP and Java.

8.1 CGI - Common Gateway Interface

CGI is a standard for helping web servers run external programs and return *dynamic* web pages.

For example, a simple dynamic web page might return the current date and time, calculated by running the '**`date`**' program, and formatting the results as a web page. The following script shows the idea:

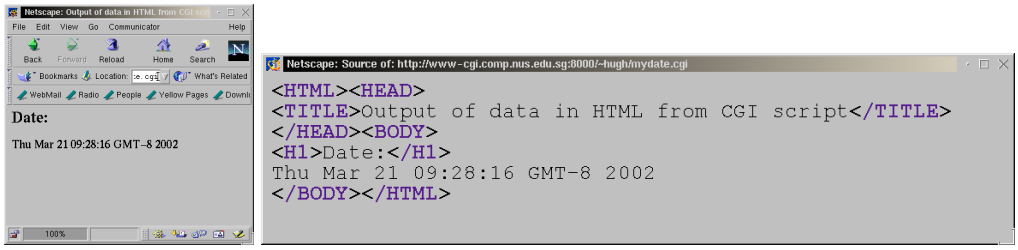
CODE LISTING **mydate.cgi**

```
#!/bin/sh

cat <<EOM1
Content-type: text/html

<HTML><HEAD>
<TITLE>Output of data in HTML from CGI script</TITLE>
</HEAD><BODY>
<H1>Date:</H1>
EOM1
date
cat <<EOM2
</BODY></HTML>
EOM2
```

When this script is placed in the directory **`public_cgi`** in your home directory on one of the UNIX systems, then you may refer to

```
http://www-cgi.comp.nus.edu.sg:8000/~yourid/mydate.cgi
```

and you will get the following display:



This is of course a trivial example, but shows the fundamental idea of running a script, to get dynamic content in a web page. In this case, the script is not passed any data from the client browser - it just runs the shell **`/bin/sh`** and the **`date`** program on the server.

Note that there is no requirement for your CGI program to be a shell script. You may use any suitable scripting language, or compiled programs, and **`perl`** is very commonly used in this role. The most important thing to remember is that whatever your CGI program does, it should not take too long to process.

8.1.1 CGI environment variables

To pass parameters to CGI programs, environment variables are used. The following **`perl`** script can display all the enviroment variables passed to a CGI program:

CODE LISTING **env.cgi**

```
#!/usr/local/bin/perl

print "Content-type: text/html\n\n";
print <<EndOfHTML;
<html><head><title>Print Environment</title></head>
<body>
EndOfHTML

foreach $key (sort(keys %ENV)) {
   print "$key = $ENV{$key}<br>\n";
}

print "</body></html>";
```

When this script is run, we get something like the following. This may go some way towards explaining to you how some systems know which web browser you are using!

```
DOCUMENT_ROOT = /usr/local/apache/htdocs
GATEWAY_INTERFACE = CGI/1.1
HTTP_ACCEPT = image/gif, image/x-
xbitmap, image/jpeg, image/pjpeg, image/png, */*
HTTP_ACCEPT_CHARSET = iso-8859-1,*,utf-8
HTTP_ACCEPT_ENCODING = gzip
HTTP_ACCEPT_LANGUAGE = en
HTTP_CONNECTION = Keep-Alive
HTTP_HOST = www-cgi.comp.nus.edu.sg:8000
HTTP_REFERER = http://www-cgi.comp.nus.edu.sg:8000/~hugh/
HTTP_USER_AGENT = Mozilla/4.79 [en] (X11; U; Linux 2.2.16 i686)
PATH = /usr/local/bin:/usr/bin:/bin:/usr/local/php/bin
QUERY_STRING =
REMOTE_ADDR = 137.132.90.155
REMOTE_HOST = dhcp-hugh.ddns.comp.nus.edu.sg
REMOTE_PORT = 3343
REQUEST_METHOD = GET
REQUEST_URI = /~hugh/env.cgi
SCRIPT_FILENAME = /home/staff/hugh/public_cgi/env.cgi
SCRIPT_NAME = /~hugh/env.cgi
SERVER_ADDR = 137.132.90.7
SERVER_ADMIN = websp@comp.nus.edu.sg
SERVER_NAME = www-cgi.comp.nus.edu.sg
SERVER_PORT = 8000
SERVER_PROTOCOL = HTTP/1.0
SERVER_SOFTWARE = Apache/1.3.19 (Unix) PHP/4.0.5 mod_perl/1.25
TZ = Singapore
```

The bold strings are the names of the environment variables passed to the CGI program. To the right of the name is the value of that environment variable.

8.1.2 CGI forms

CGI is commonly used for processing simple form-based applications. That is, the client display has a form, and the user keys-in, or selects items in the form, which is then submitted to the CGI program for processing. The principal mechanism for passing small form contents to the CGI program uses environment variables, which are passed to the called CGI program.

The form contents are found inside an environment variable called **`QUERY_STRING`**, as a series of **`name/value`** pairs. This mechanism is known as the **`GET`** mechanism, and a typical URL would look like this:

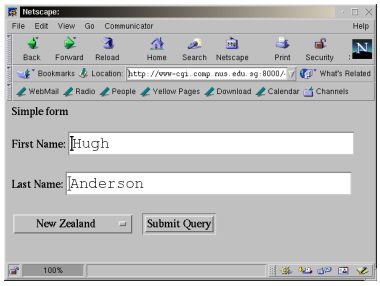
```
http://www-cgi.comp.nus.edu.sg:8000/~yourid/myform.cgi?name1=value1&name2=value2
```

An alternative mechanism is the **`POST`** mechanism, in which the **`STDIN`** of the CGI program is used to process the form data. A simple example of a form based web page:

CODE LISTING **form.html**

```
<html><head>Simple form</head>
<body>
<form action="env.cgi" method="GET">
      First Name:  <input type="text" name="First" size=30><p>
      Last Name:  <input type="text" name="Last" size=30><p>
      <select name="Home">
      <option>Singapore    <option>Malaysia
      <option>Indonesia    <option>New Zealand
      <option>The rest of the world!
      </select>
      <input type="submit">
</form>
</body></html>
```

This produces a page that looks like this:



When the form is submitted, the **`QUERY_STRING`** looks like this:

```
QUERY_STRING = First=Hugh&Last=Anderson&Home=New+Zealand
```

Within a CGI program, this series of name-value pairs may be used to return a dynamic web page based on this form data. **`Perl`** is a particularly useful language to use in this context, as it has powerful operators form managing strings, and the **`QUERY_STRING`** can be **`split`** quickly into its component parts. There are security issues with unrestricted CGI programs - since they run powerful programs (like perl and csh) with arbitrary parameters, they may be a source of (hacker) intrusion. It is for this reason that CGI usage is restricted here at NUS.

8.2 PHP

PHP is a server-side scripting language that is embedded in your web pages. It looks very like standard HTML scripts, but when a client browser queries a PHP enhanced web page, the server automatically interprets the PHP, and then sends back an ordinary HTML page. There are no enhancements needed for browsers to access PHP web pages.

The two tags **<?php** and **?>** start and end a PHP script, and identify a PHP code segment. The PHP code itself is a reasonably powerful programming language similar to Java, C and Perl, with functions, variables and so on.

PHP stands for PHP - Hypertext Preprocessor, a recursive acronym (like GNU - Gnu's Not UNIX), and is a generally useful HTML/server preprocessor. However it is particularly useful if you wish your web pages to access databases. It is common to pair up PHP with MySQL, but PHP is not limited to one database type. It may be used with any of the commercial databases. For example if you wish to use PHP to access a Microsoft SQL server, you can install the ODBC support in the server machine, and access the server directly.

Here is sample PHP code embedded in a PHP-enhanced web page. It shows PHP connection to a MySQL server, selection of a database and an SQL query:

```
<?php
    ...
    mysql_pconnect("host","user","password")
        or die("Unable to connect to SQL server");
    mysql_select_db("dbasename")
        or die("Unable to select database");
    $numguests = mysql_query("SELECT COUNT(*) FROM guests")
        or die("Select Failed!");
    ...
?>
```

This may all be integrated with standard HTML and form-based web pages to construct a GUI. PHP suffers less from the security issue than perl or csh CGI scripts do. After writing this sentence I went to see what the latest security issues with PHP were, and discovered that recently there has been a major loophole discovered in PHP POST upload code:

> *Each of the flaws could allow an attacker to execute arbitrary code on the victim's system.*

The exploit appers to have been disovered before any use of it, so assuming you have a relatively recent installation of PHP, you should in general have less security worries than with CGI scripts.

8.3 Java enhanced

In Section 6.5 we saw a very simple hello-world Java applet inserted in a web page. Here is a little Java applet for a Lissajous figure:

CODE LISTING **Lissajous1.java**

```
/* @(#)Lissajous.java
 *  Original version was written in 0.4 95/04/09
 *  by Hugh Anderson for HotJava browser.
 *
 *  Updated by L. Gladney to Java 1.0 JDK on 4/13/97.
 *
 * Patrick Chan (chan@scndprsn.Eng.Sun.COM ) has suggested that it
 * would be nice if every point  had a different display, so mouse
 * X motion now controls the ratio of frequencies, and mouse Y motion
 * controls the amplitude. */

import java.applet.Applet;
import java.awt.*;

public class Lissajous extends Applet implements Runnable {
    Thread animate=null;
    double pi=3.14159265359;
    int fx=50;
    int fy=100;
    int diffx=0;
    int amp=50,phase=0;                          // amplitude, phase
    int delay = 50;                              // speed set by length
                                                 // of sleep between refreshes
    public void init() {
        resize(200, 200);                        // resize to fixed width,height
    }

    public void paint(Graphics g) {
        int X,Y,YY=0,lastx=0,lasty=0,temp=0,rev=0;
        g.drawRect(0, 0, size().width - 1, size().height - 1);  // outline
        if ( fy < fx ) {                                        // frequency
            temp = fx;
            fx = fy;
            fy = temp;
            rev = 1;
        }
        for (int x = 0 ; x <= 360 ; x += 4) {                   // loop
            X = (int) ( amp*Math.sin( x*2.0*pi/360.0 ));        // x pos
            YY = (x*fy/fx)+phase;
            Y = (int) ( amp*Math.sin( YY*2.0*pi/360.0 ));
            if (x==0) { lastx=X; lasty=Y; }
            if (rev==1) { g.drawLine(lastx+100,lasty+100,X+100,Y+100); }
             else { g.drawLine(lasty+100,lastx+100,Y+100,X+100); }
            lastx=X;
            lasty=Y;
        }
        if ( rev==1 ) {
           temp=fx;
           fx = fy;
           fy = temp;
         }
        phase = YY;
        /* Fix an error ... phase shouldn't increase forever..... */
        if ( phase < 0 ) { phase += 360; };
        if ( phase >=  360 ) { phase -= 360; };
        g.drawString( fx + ":" + fy,10,20);
    }
```

Here is the rest of the code...

CODE LISTING **Lissajous2.java**

```
    public void run() {
        while (true) {
            repaint();
            try { Thread.currentThread().sleep(delay);  // delay
            }
              catch (Exception e) { };
        }
    }

    public void start() {
        if ( animate == null ) {
            animate = new Thread(this);
            animate.start();
        }
    }
    public void stop() {
        if (animate != null ) {
            animate.stop();
            animate = null;
        }
    }

    public boolean mouseDown(Event e, int x, int y ) {
    Graphics gc;
    gc = getGraphics();

        diffx = fx-x;
    System.out.println("Got a mouse event at " + x + "," + y);
        return true;
    }

    public boolean mouseDrag(Event e, int x, int y) {
        fx = x+diffx;
        if ( fx <= 0 ) { fx = 1; };
        amp = y;
        return true;
    }

    public String getAppletInfo() {
                return "Lissajous by Hugh Anderson/Larry Gladney ";
        }

        public String[][] getParameterInfo() {
                String [][] info = {
                        {"delay    ","int     ", "delay, default=50"}
                };
                return info;
        }

}
```

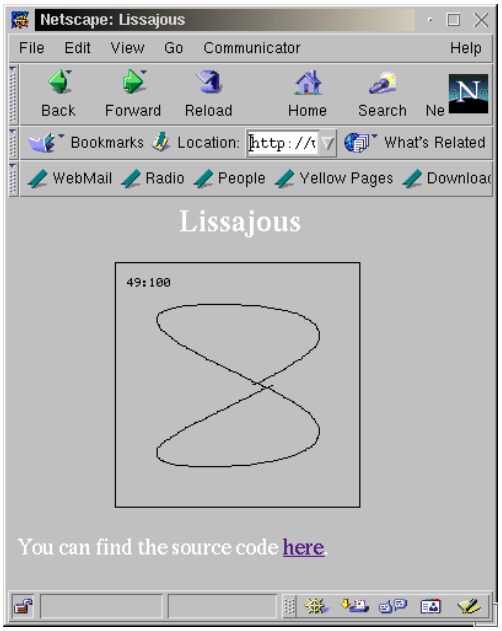
I wrote this code soon after the first Java language was made public, and had forgotten about it until last week, when I went looking for an applet, and found my name on it! This code may be found at

```
http://olddept.physics.upenn.edu/courses/gladney/minicourse/lectures/lecture2.html
```

or locally at

```
http://www.comp.nus.edu.sg/~hugh/Lissajous/Lissajous.html
```

This produces a page that looks like this:



8.4 Summary of topics

In this module, we introduced the following topics:

- Web-based application architectures
- CGI, PHP and Java

Tutorial 6 - questions for week 12 (Mar 27, 2002)

1. (Research) Make up a simple CGI form, similar to the one given on page 71, which uses the **`POST`** method for reading the data, and prints out the three fields.
2. (Research) Make up a simple PHP processed form, similar to the one given on page 71, and which prints out the three fields.
3. Examine the **`Lissajous.java`** code. What is the function of the **`diffx`** variable?

Further study

- http://php.net

8.5 Assignment 4 - Implementation

In this assignment, you may work in a group of up to 4 people - or you may do it by yourself. The assignment is worth 40% of your assignment grade, and is due at 5:00 p.m. on Friday, 19th April, 2002 - please deliver to Hugh's room. The assignment is to be done using Java/Swing.

Task:

Your task is to implement and document a multi-user simple text editor. By this I mean that you and other people may safely simultaneously edit a single file (perhaps using a shared file system). Note that there is no one correct way of doing this, and feel free to experiment with different approaches.

You may re-use existing simple text editor code found in many introductory Java/Swing tutorials if you wish, but the extensions to make it multi-user must be your own.

Here are brief descriptions of possible approaches to this problem:

1. (Simplest) Each user edits their own copy of the file. When the user saves the file, other users are notified that the original file has changed, and given a choice to load the new changed file, or to just continue.
2. (Medium) Each user edits a part of the file, chosen when they open the file from the remaining editable parts (For example, lines 1-100 for user 1, lines 101-200 for user 2). When the user saves the (part) of the file, other users are notified that the original file has changed, and their programs automatically load the new changed parts.
3. (Very tricky) Each user edits their own copy of the file. As changes are made, all users screens are updated.

Feel free to dream up more advanced shared editing schemes. Note that since communication between the programs may be tricky, I suggest your programs communicate through another shared file.

Deliverables:

- A title page containing your names and matriculation numbers.
- A ten to twenty page document containing
  - A brief summary and justification of the overall strategy used for shared editting - perhaps with a state diagram describing the states of each edit program, with an argument as to why your program is "safe".

  - An overview of the interface design
  - A user manual for the interface

- A disk containing the code, with a (small) README file to explain how I am supposed to run your software.

Note that this assignment requires you to implement the application, not just to design one.

Assessment:

The assessment is as follows:

Documentation	25%
Code style/quality	25%
Justification of the strategy	25%
Operation of the interface	25%

Chapter 9

Visualization

The visualization of data using GUI applications should be distinguished from other computer-graphics concerns. In visualization, we are concerned with *exploration* of data, with its attendant concerns of encoding strategies and so on. In computer-graphics, we may be more concerned with rendering techniques.

Before exploring the implementation of a 3D visualization, we look at some aspects of the context within which the visualization is to be used.

9.1 The use of 3D

A successful visual metaphor has some analog with *real-world* physics. Some studies suggest that a 10-fold improvement in item density can be achieved in using three dimensional displays.

The **etherman** display has proven effective in observation of networks with 50 or less nodes, but becomes cluttered and unusable with more nodes on-screen. By extending the display into the third dimension, it immediately becomes clearer. Our familiarity with spatial location allows us to understand that objects further away will be smaller, and this reduces the visual clutter. However if a *far away* object increases in size, we immediately notice, and can rotate the display to observe more closely. We notice even if the *far away* object is still smaller (in screen *real-estate* terms) than closer objects. This human cognitive behaviour becomes apparent as soon as sufficient visual cues have been given to persuade the observer that the display is in three dimensions.

For example: Figure 9.1 shows the output of an original program[1] to display tasks active on a UNIX machine. The size of the spheres indicate the amount of memory used by each process, the colour represents the owner. This display can be rotated and used to examine activity in a way unattainable using standard system process viewing tools. The display has over 100 visible nodes, but it is still easy to identify and investigate individual nodes.

[1] http://opo.usp.ac.fj/~hugh/Public/Viz/ThesisWork/processes1

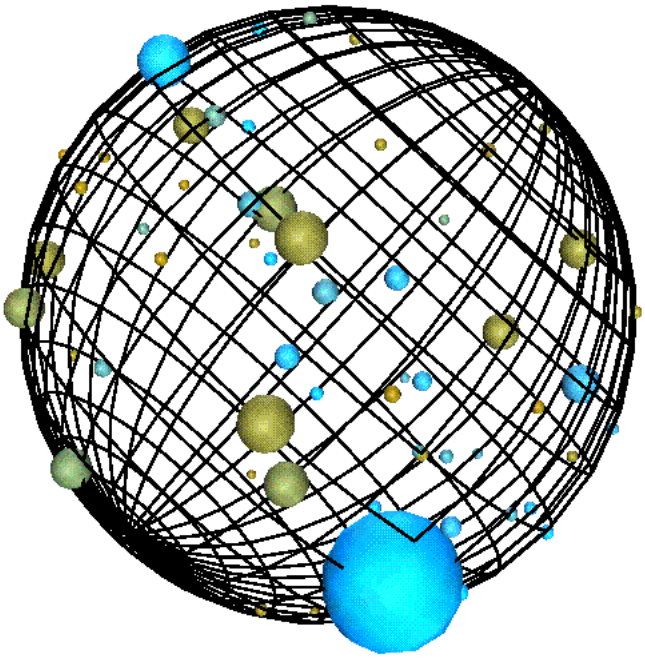


Figure 9.1: Display of tasks in a multi-tasking environment.

9.2 OpenGL

OpenGL was originally the SGI in-house graphics system, but now is the most widely accepted graphics standard, with chip, API and OS support for all platforms. It is possible to code directly using the OpenGL API, but more normal to use a toolkit which encapsulates some abstraction, built on top of OpenGL calls.

Open GL is standard on all UNIXes and all versions of Windows since Win95. The API supports functions for rendering, buffering, anti-aliasing, shading, colouring, texture-mapping, a display list, Z-buffering and so on. To give the flavour of raw OpenGL programming, here is a small application:

CODE LISTING **teapot.c**

```
#include <GL/glut.h>

void
Teapot (long grid)
{
    /* ... code to construct drawlist of teapot here. */
}

static void
Init (void)
{
    glEnable (GL_DEPTH_TEST);
    glLightModelfv (GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);
    /* Lighting model, materials... */
}

static void
SpecialKey (int key, int x, int y)
{
    switch (key) {
     case GLUT_KEY_UP:
        rotX -= 20.0;
        glutPostRedisplay ();
        break;
        /* Move in other directions */
    }
}

static void
Draw (void)
{
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix ();
    /* ... translations ... */
    glCallList (teaList);
    glPopMatrix ();
    glutSwapBuffers ();
}

int
main (int argc, char **argv)
{
    glutInit (&argc, argv);
    type = GLUT_RGB | GLUT_DEPTH;
    type |= (doubleBuffer) ? GLUT_DOUBLE : GLUT_SINGLE;
    glutInitDisplayMode (type);
    glutInitWindowSize (300, 300);
    glutCreateWindow ("TeaPot");
    Init ();
    glutReshapeFunc (Reshape);
    glutKeyboardFunc (Key);
    glutSpecialFunc (SpecialKey);
    glutDisplayFunc (Draw);
    glutMainLoop ();
}
```

9.3 Java 3D, VTK - toolkits for 3D

These systems are 3D OO toolkits embedded in Java and C++ respectively. The Java 3D application programming interface (API) provides a set of object-oriented interfaces that support a simple, high-level programming model.

The Visualization ToolKit (VTK) is an open source OO software system for 3D consisting of a C++ class library, and several interface layers for Tcl/Tk, Java, and Python. VTK has a wide variety of visualization and graphical functions, and has been installed and tested on both UNIX and Windows.

9.4 Case study - network traffic application

A user-requirement specification for a network traffic application may begin with something like:

---

*This visualization is to assist network managers in planning and monitoring their networks. It allows interactive exploration of network datalink traffic, and is intended for use both for visualization of immediate-mode (*real-time*) data, and for visualization of historical data. (The visualization is the same in each case, except that* time only travels forward *in the immediate mode.)*

The visualization will help answer questions such as the following:

- *Which segments carry the most traffic?*
- *Which sections of the network are down?*
- *At what times, and where do traffic bottlenecks occur?*
- *What is the line utilization for different lines at different times?*
- *What types of traffic are used most?*
- *Would routing or switching be effective here?*

For this *network* traffic application, the following elements are represented:

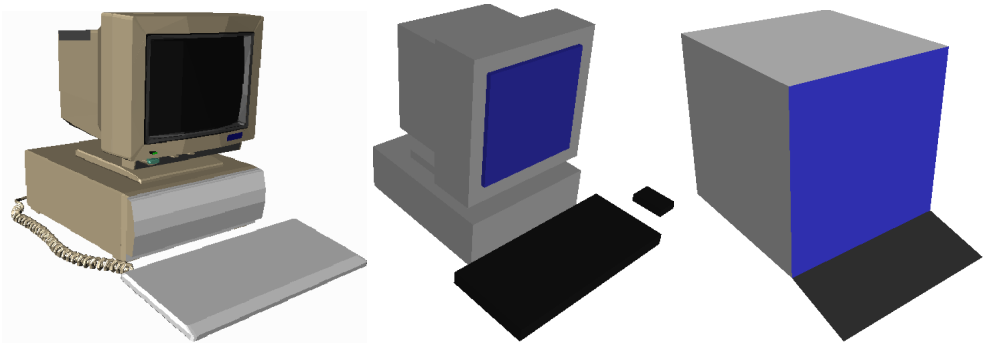
- Background: - to convince the viewer that the display is *three dimensional*...
- Nodes: - a computer, a network device...
- Traffic: - the amount of traffic flow...
- Protocol: - the *type* of traffic...
- Errors: - errors in traffic could be further traffic *protocols*...
- Trends: - for changes over time...
- Association: - for network insights...

9.4.1 Node representation

In our chosen context, the nodes represent computers or network components such as hubs, routers, bridges or switches. In locational or representational displays we may want to differentiate between the type of node, but in the more *abstract* displays, there may be no need to do this.

In Figure 9.2 we see a range of possible options for more concrete representations of nodes.

The computer represented in Figure 9.2(a) has about 2000 flat triangular surfaces (some of them hidden). If we were visualizing a campus with (say) 500 computers using this representation,



(a) 2000 polygons (b) 84 polygons (c) 14 polygons

Figure 9.2: Concrete node representations.

Machine	Rendering speed	Computer (a)	Computer (b)	Computer (c)
Graphics Workstation	485,000 Δ/sec	0.485 frames/sec	11.5 frames/sec	69 frames/sec
PC1	30,000 Δ/sec	0.03 frames/sec	0.71 frames/sec	4.3 frames/sec
PC2	11,000 Δ/sec	0.011 frames/sec	0.26 frames/sec	1.6 frames/sec

Table 9.1: Workstation redraw speed.

then our rastering engine has to recalculate the positions and shading of 1,000,000 polygons each time it redraws the screen. This will happen even if the item is so *far away* that it only takes up a single pixel on the screen.

A typical modern hardware rastering engine can calculate 485,000 shaded Δ/sec (triangles per second), and hence our screen refresh rate would be about half a frame per second, giving a *jerky* look. By contrast, the computer shown in Figure 9.2(c) has only 14 flat triangular surfaces, giving a frame rate in excess of 70 frames per second.

Standard PCs often come with graphics cards that support pixel movement on screen, but their overall performance in shaded Δ/sec is normally considerably below 485,000 polygons per second. Table 9.1 gives the resultant frame rates for displaying onscreen 500 of the node representations in Figure 9.2.

It is clear from this table that if we wish our visualizations to be viewed on a range of platforms, we must choose our node representations carefully to minimize rendering time.

Some representation methods for three dimensional objects allow different *levels of detail*. In the VRML specification, a single object may be represented in different ways depending on how much screen *real estate* it uses up. If the object is near you, it could be represented in detail, but if it is a long way away, the representation could be as simple as a coloured square.

The following VRML code represents a cone in two ways using an LOD (Level Of Detail) node. If the distance from the user to the object is smaller than the first range value specified, then the first version is drawn. If the distance is greater than the last range specified, the last version is drawn.

```
#VRML V2.0 utf8
LOD {
    range [20]
    level [
       #full detail 16 sided cone
       Shape{
          appearance Appearance { material Material { diffuseColor 1.0 1.0 1.0}}
          geometry Extrusion{
             crossSection [ -1 0, 0 0, -1 -2 -1 0]
             spine [1 0 0 , 0.866 0 0.5, 0.5 0 0.866, 0 0 1 , -0.5 0 0.866, -
0.866 0 0.5,
                    -1 0 0, -0.866 0 -0.5, -0.5 0 -0.866, 0 0 -1 ,0.5 0 -0.866,
                     0.866 0 -0.5, 1 0 0]
          }
       }
       #low detail 4 sided cone, actually a pyramid
       Shape{
          appearance Appearance { material Material { diffuseColor 1.0 1.0 1.0}}
          geometry Extrusion{
             crossSection [ -1 0, 0 0, -1 -2 -1 0]
             spine [1 0 0 , 0 0 1, -1 0 0, 0 0 -1 , 1 0 0]
          }
       }
    ]
}
```

9.4.2 Traffic and protocol representation

A simple immediate way to represent traffic between two nodes is to just draw a line between them. The nature of network communication on a typical Local Area Network (LAN) is such that the resultant lattice is likely to be relatively sparse.

For example: at the datalink layer, on average, a workstation at any one time may only be communicating with six or seven other datalink addresses - two broadcast addresses, (say) two file servers, a WINS or DNS server. and a proxy. So - rather than having a lattice with $\frac{n^2-n}{2}$ interconnections, we have $k(n-1)$ interconnections, where k is some small integer. Even so, a lattice with 500 nodes may have 3,000 interconnections and may look jumbled.

A line indicates source and destination, but not the *amount* of traffic. Three systems for this purpose have been examined:

1. Colour coding (black through red to white for maximum traffic),
2. Line width, and
3. The length of partial lines, as discussed in Eick's papers.

Using a linear increase in the line width appears most effective, although it does increase clutter. It also leaves the colour information free for use in some other encoding. The linear scale needs a sensible maximum, and experimentation has shown that a maximum width should be equivalent to the size of the node.

A simple line or cylinder also does not tell which way the traffic is flowing. We evaluated the following cues by modeling them in *geomview*, a geometrical modelling package.

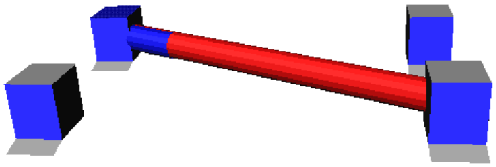


Figure 9.3: Partial length representation of bi-directional traffic.

1. Separate arrows
2. Partial lengths

In Figure 9.3 we see the traffic between two computers, the size of the cylinder between the machines indicating the total amount of network traffic, and the two colours indicating the relative amounts of traffic going each way. The nodes and cylinders themselves are coloured according to the dominant protocol type.

9.4.3 Trend representation

Trends are sometimes difficult to find in large sets of data such as found in our application. Once an examination of a visualization has indicated that a trend may be possible, it is normally easy to frame the questions needed to verify the trend.

- *"It looks like HTTP usage on these segments is increasing..."*
  (⇒Plot HTTP usage for the segment machines versus time).
- *"It looks like HTTP usage is increasing when FTP usage is decreasing..."*
  (⇒Plot HTTP and (1-FTP) versus time).

Graphing continues to be the pre-eminent way of representing trends and the role of visualization is to assist in finding the trends.

The four-dimensional visualization methods outlined and demonstrated by Olaf Holt and Nils McCarthy in NDdemo (the fourth dimension being explicit time) could perhaps be used in trend analysis, but the visualization is a little hard to use.

A final method is to attempt to encode previous visualizations *on-top-of* the current one (but perhaps semi-transparent) - the idea here is one of *visual* echoes. In only some circumstances can this be successful. There are two options:

1. Echoes are fixed on the screen, and we can move the visualization away from them, leaving a trail like this:
   In the worst case though, we have just ended up using one of our three display dimensions for *"time"*.

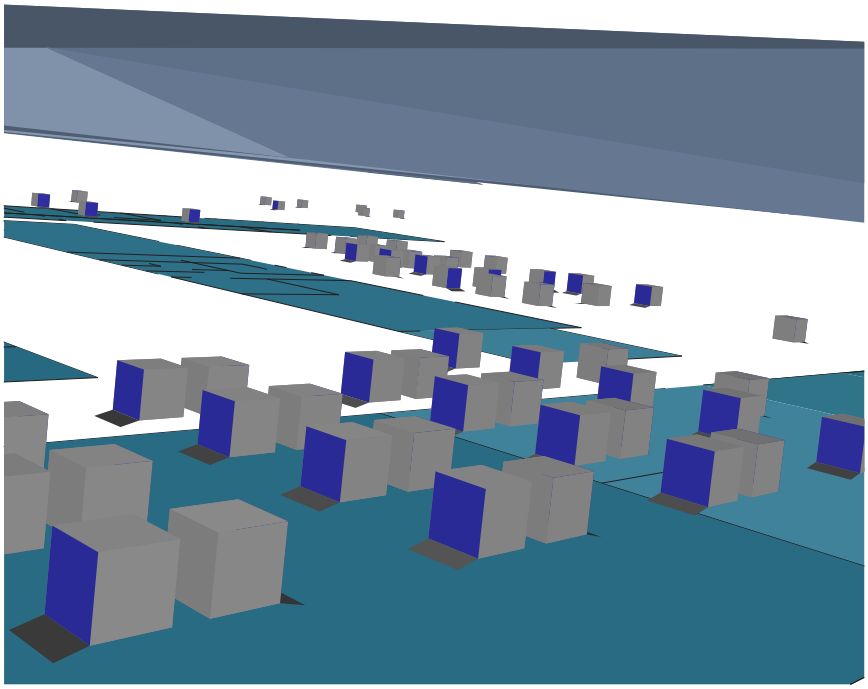


Figure 9.4: Locational view

2. Semi-transparent echoes are co-located with the visualization. In this case, we can only show some of the history. We can show a reducing item, but not an increasing one.

9.4.4 Display

In Figure 9.4, we see an early locational view showing the fixed components of the visualization, and modeled using **geomview**. It shows nodes for the computers, floor plans for the buildings, and a transparent roof. The display uses the normal 3D navigation tools for adjustment. From the display it is easy to identify the location of machines, and the display should be efficient enough to support display and manipulation of the entire network (with 500 machines as a suitable goal), and - yes - the computers are floating in mid air. (Since we are concerned with efficiency we choose the simplest understandable visualization, and tables just become extra polygons to draw).

The visualizing tool supports rotation and translation of the display, so that the observer can easily focus on regions of interest. Suitable systems are found in the CosmoPlayer VRML viewer, and in **geomview**. Note that this visualization is not dependant on the navigation or implementation method.

Cables and network infrastructure are not marked on the display, but the display does support an aggregation-by-rule construct. This aggregation can be used to associate machines all on the same segment, or all used by the same department.

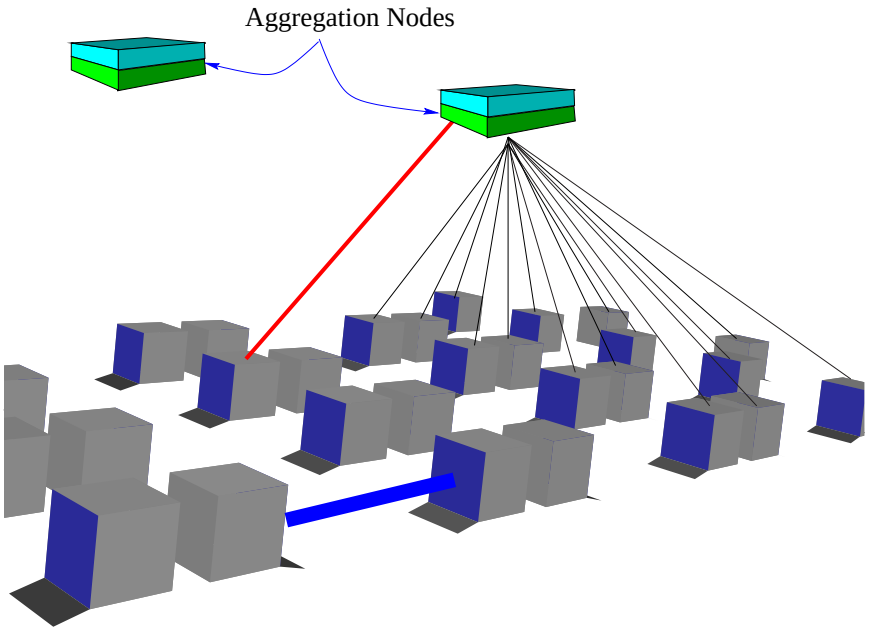


Figure 9.5: Aggregation nodes

The components are chosen to be the best minimum complexity representation consistent with fast updates. The frame update speed for the most complex display is better than 2 frames per second.

Each node or aggregation in the overview display is clickable to turn it off or on. If a node is turned off, its traffic no longer is displayed (either directly or as part of some aggregated traffic). If an aggregation is turned off, any existing traffic displays to its nodes are removed. This facility is used to allow fast reduction of visual clutter.

Aggregation nodes also have an aggregation *switch*, which allows them to combine all traffic for subsidiary nodes. When this switch is on, lines connect the aggregation to its subsidiary nodes.

The aggregation node floats above its associated nodes. In Figure 9.5, we see two aggregation nodes, with the one on the right aggregating traffic to and from all its subsidiary nodes. All traffic to or from these nodes is displayed going to the aggregation node. The other nodes are not aggregated, and display traffic directly.

Each node or aggregation in the overview display is clickable to identify specific information about that node. This information does not replace the display, but appears in a separate window. Initially this information may just be textual information such as the name of the node along with traffic totals, but eventually, it is expected that the drill-down display will be the metaphor display specified elsewhere, showing only the selected node in 3D, along with any associated nodes.

Operating System	Web Browser	VRML 2.0 Plugin
IRIX	Navigator 3.01S	CosmoPlayer 1.0.2b3 or later
	Communicator 4.04	CosmoPlayer 1.0.2b3 or later
	Communicator 4.07	CosmoPlayer 2.1 beta
Macintosh	Communicator 4.04	CosmoPlayer 2.1 or later
WIN32	Navigator 3.01	CosmoPlayer 1.0 beta 3 or Intervista WorldView 2.0 or later
	Communicator 4.04	CosmoPlayer 1.0 beta 3 or Intervista WorldView 2.0 or later
	MSIE 3.0	CosmoPlayer 1.0 beta 3 or Intervista WorldView 2.0 or later
	MSIE 4.0	CosmoPlayer 1.0 beta 3 or Intervista WorldView 2.0 or later

Table 9.2: Systems which support the EAI.

9.5 3D VRML visualization implementation

The VRML visualizer is a relatively small Java program which must be loaded as an applet along with a VRML view of the network. A small web page is created, and may be used to view the visualization using a web browser such as Netscape along with the CosmoPlayer VRML plugin.

Unfortunately, not all combinations of web browser and VRML plugin work correctly with the EAI, but the systems in Table 9.2 are known to work. These systems were current in 1999. This year (2002) all the systems I tried at NUS appeared to work fine.

Load the default web page in the directory, and the visualization should be visible. To finish using the visualizer, you must exit the browser entirely. If not, the Java applet keeps communicating with the **collector**.

In Figure 9.6, we see an active VRML display within a browser. The computer nearby is generating a lot of traffic. In the distance we can see other nodes, and the roof and floors.

9.5.1 3DVNT VRML software

3DVNT includes software to create a default HTML web page for the VRML visualization. The current default web page is like this:

```
<html><head> <title>Sample 3DVNT Page</title> </head>
<center><H1>Sample 3DVNT Page </H1></center>
<center> <embed src="root.wrl" height="600" width="700"> </center>
<center> <applet code="View1.class" width="100" height="10" mayscript>
<PARAM name="segment" value="MACS"> <PARAM name="port" value="9876">
<PARAM name="host" value="opo.usp.ac.fj"> </applet> </center>
OK?
</html>
```

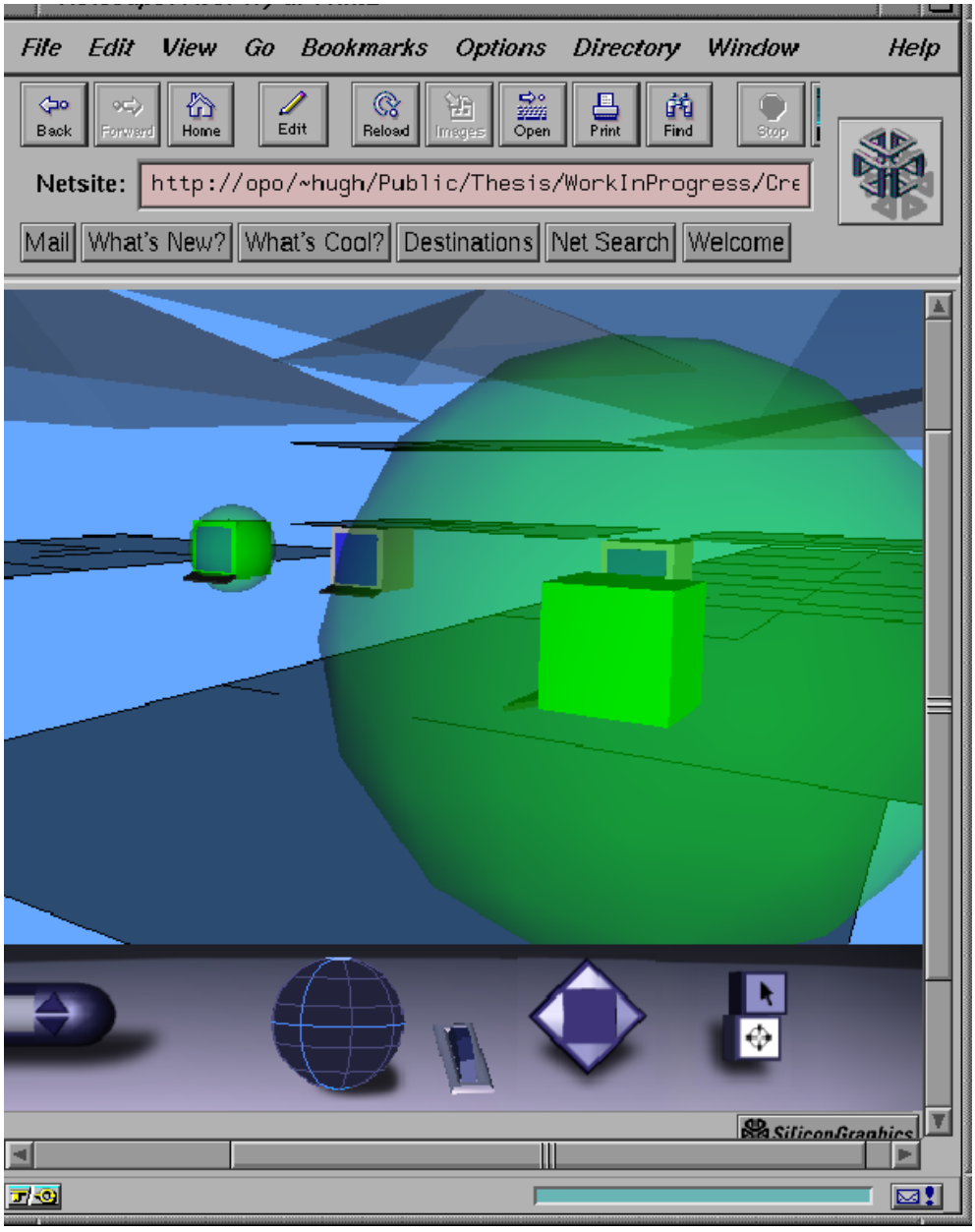


Figure 9.6: 3DVNT view within Netscape browser.

The *root.wrl* file which forms the basis of the VRML visualization is of the following format:

```
PROTO CLUSTER  [] { ... }     # Cluster definition
PROTO KEYBOARD [] { ... }     # Keyboard definition
PROTO SCREEN   [] { ... }     # Screen definition
PROTO GLOBE    [] { ... }     # Traffic sphere definition
# Some setting up declarations
Background { skyColor .4 .66 1 }
NavigationInfo { type [ "EXAMINE", "ANY" ] speed 400 }
Viewpoint { position 0 400 0 orientation 0 1 0 4 description "Camera 1" }
# Lines, floors and roofs
DEF LINES  Transform { ... }
DEF FLOORS Transform { ... }
DEF ROOFS  Transform { ... }
# and then the nodes
DEF node1  Transform { ... }
DEF node2  Transform { ... }
# ... and so on ...
```

Each node is of the following form:

```
DEF node1 Transform {
    translation 4350 150 4365
    rotation 0 1 0 4.71238
    children [
       KEYBOARD {}
       SCREEN {}
       DEF node1box Transform {
          children [
             Shape {
                appearance Appearance { material DEF node1boxcolor Material { diffuseColor 0.8 0.8 0.8 } }
                geometry Box { size 50 50 50 }
       }  ]  }
       DEF node1sphere Transform {
          scale 1 1 1
          children [
             Shape {
                appearance GLOBE {}
                geometry Sphere { radius 1 }
}  ]  }  ]  }
```

The Java visualyzer software maintains a link to a remote data collector, and uses the EAI to modify the images in the VRML view.

```
// using the VRML External Interface.


import java.applet.*;
import java.awt.*;
import java.util.*;
import vrml.external.field.*;
import vrml.external.exception.*;
import vrml.external.Node;
import vrml.external.Browser;
import java.io.*;
import java.net.*;

public class View1 extends Applet {
//  public static final int DEFAULT_PORT = 9877;
  Browser browser;
  Socket s = null;
  DataInputStream in = null;
  String line;

  public void init() {
    System.out.println("Test.init()...");
  }
  void SocketStart  () throws java.io.IOException {
    String port = this.getParameter("port");
    int p = Integer.parseInt(port);
    try {
      String host = getCodeBase().getHost();
      System.out.println("Request came from: " + host);
       s = new Socket(host, p);
    }
    catch (UnknownHostException e) {
      System.out.println("No socket: " + e);
    }
  }
  public void start() {
    int count=0;
    Node node2sphere=null;
    Node appear=null;
    EventInSFVec3f[] scalein=new EventInSFVec3f[100] ;
    EventInSFColor[] appears=new EventInSFColor[100] ;
    float[] val = new float[3];
    int[] lastval  = new int[100];
    int n;
    String id,vl;

    while (count != 100) {
       scalein[count] = null;
       appears[count] = null;
       lastval[count] = 0;
       count=count+1;
    }
    try {
       SocketStart();
     }
    catch (java.io.IOException e) {
      System.out.println("No socket: " + e);
    }

    System.out.println("Test.start()...");
    browser = (Browser) vrml.external.Browser.getBrowser(this);
    System.out.println("Got the browser: " + browser);

    count = 0;
        try {
            in = new DataInputStream(s.getInputStream());
```

```
            while(true) {
                line = in.readLine();
                if (line == null) {
                    System.out.println("Server closed connection.");
                    break;
                }
                if (line.regionMatches(0,"n",0,1)) {
                   n = line.indexOf(32,2);
                   id = line.substring(2,n);
//                   System.out.println(">>>"+id+"<<<");
                   vl= line.substring(n+1);
//                   System.out.println("+++"+vl+"---");
                   Integer a = Integer.valueOf(id);
                   Integer b = Integer.valueOf(vl);
                   if (scalein[a.intValue()]==null) {
                      try {
                         node2sphere = browser.getNode("node"+id+"sphere");
                         System.out.println("Got the sphere node: " + node2sphere);
                      }
                         catch (InvalidNodeException e) {
                         System.out.println("PROBLEMS! node2sphere: " + e);
                      }
                      try {
                         scalein[a.intValue()] = (EventInSFVec3f) node2sphere.ge
tEventIn("scale");
                         System.out.println("Got the sphere scale node: " + appears[a.in
tValue()]);
                      }
                      catch (InvalidNodeException e) {
                         System.out.println("PROBLEMS! (scalein): " + e);
                      }

                      try {
                         appear = browser.getNode("node"+id+"boxcolor");
                         System.out.println("Got the Boxcolor node: " + appear);
                      }
                         catch (InvalidNodeException e) {
                          System.out.println("PROBLEMS! appearance: " + e);
                      }
                      try {
                         appears[a.intValue()] = (EventInSFColor) appear.getEven
tIn("set_diffuseColor");
                         System.out.println("Got the Boxcolor color node: " + appears[a.i
ntValue()]);
                      }
                         catch (InvalidNodeException e) {
                          System.out.println("PROBLEMS! appearance color: " + e);
                      }
                   }
                   if (b.intValue()==-1) {
                      val[0] = (float)1.0;
                      val[1] = (float)1.0;
                      val[2] = (float)1.0;
                   } else {
                      val[0] = (float)(b.intValue()*20)+1;
                      val[1] = (float)(b.intValue()*20)+1;
                      val[2] = (float)(b.intValue()*20)+1;
                   }
                   scalein[a.intValue()].setValue(val);

                   if ((b.intValue()==0) != (lastval[a.intValue()]==0)) {
                      if (b.intValue()==0) {
                         val[0] = (float)0.8;
                         val[1] = (float)0.8;
                         val[2] = (float)0.8;
                         appears[a.intValue()].setValue(val);
```

```
                        } else {
                           if (b.intValue()==-1) {
                              val[0] = (float)0.1;
                              val[1] = (float)0.1;
                              val[2] = (float)0.1;
                              appears[a.intValue()].setValue(val);
                           } else {
                               val[0] = (float)0.0;
                               val[1] = (float)1.0;
                               val[2] = (float)0.0;
                               appears[a.intValue()].setValue(val);
                           }
                        }
                     }
                     lastval[ a.intValue()]=b.intValue();

                  }
//                  System.out.println(line);
               }
            }
            catch (IOException e) { System.out.println("Reader: " + e); }

      }


    public Browser getBrowser() {
      return browser;
    }

}
```

9.6 Summary of topics

In this module, we introduced the following topics:

- Visualization versus computer-graphics
- OpenGL
- (Briefly) Java3D, VTK
- VRML/Java/EAI

Tutorial 7 - questions for week 13 (April 3, 2002)

1. Find a minimal VRML file which constructs a solid cube.
2. Find minimal OpenGL display-list code to draw a cube.
3. Examine the Java code given for using the EAI to modify the VRML. How exactly does the code get a reference to a VRML node?
4. Examine the Java code given for using the EAI to modify the VRML. How exactly does the code modify a VRML node?

Further study

- sunsite for Java3D
- The EAI specification

Chapter 10

MFC

The Microsoft Foundation Classes provide all the classes needed to produce GUI (Microsoft) Windows programs. A typical development cycle with MFC involves using a rapid application development tool such as the wizard found in Visual C++, and then modifying the resultant source code. The RAD development cycle is relatively easy, but unfortunately, the second phase is not.

10.1 MFC menus

There are many ways to create menus in MFC, but it is common to use a special menu resource file. A resource file for a simple File/Quit menu might look like this:

```
#define MYAPP_EXIT 3210
MyApp MENU
    POPUP “File”
    {
       MENUITEM “Exit”,MYAPP_EXIT
    }
}
```

In the **`Create`** call, you can do something like this:

```
Create( NULL, “Example”, ..., CRect(...), NULL, “MyApp” );
```

The **`MYAPP_EXIT`** message may be bound using the **`DECLARE_MESSAGE_MAP()`** macro, and with the following declaration:

```
ON_COMMAND( MYAPP_EXIT,OnExit )
```

Finally, we need a message handler:

```
afx_msg void CMenusWin::OnExit()
{
    SendMessage( WM_CLOSE );
}
```

10.2 MFC Programming

Here is a simple initial example of an MFC application built using Microsoft Visual C++, modified from the Deitel&Deitel MFC book:

The code to do this is here:

CODE LISTING **FirstApp.cpp**

```
#include <afxwin.h>

class CFirstWindow : public CFrameWnd {
public:
   CFirstWindow();
   ~CFirstWindow();
private:
   CStatic *m_pGreeting;
};

CFirstWindow::CFirstWindow()
{
   Create( NULL,
           "First Application",
           WS_OVERLAPPEDWINDOW,
           CRect( 100, 100, 400, 220 ) );
   m_pGreeting = new CStatic;
   m_pGreeting->Create(
      "Hello World!", // text
      WS_CHILD | WS_VISIBLE | WS_BORDER
      | SS_CENTER,
      CRect( 80, 30, 200,50 ),
      this );
}

CFirstWindow::~CFirstWindow()
{
   delete m_pGreeting;
}

class CFirstApp : public CWinApp {
public:
   BOOL InitInstance()
   {
      m_pMainWnd = new CFirstWindow();
      m_pMainWnd->ShowWindow( m_nCmdShow );
      m_pMainWnd->UpdateWindow();
      return TRUE;
   }

} FirstApp;
```

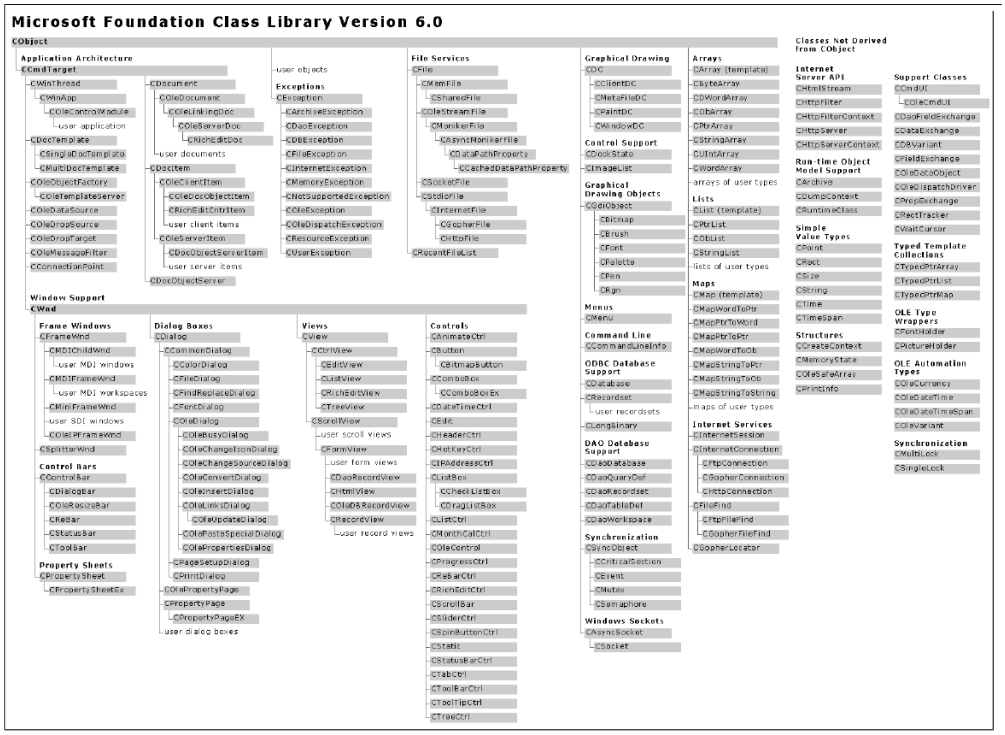
Note the use of Hungarian notation:

Prefix	Meaning
`C`	Class declaration
`m_`	Class member variable
`p`	Pointer
`n` or `i`	Integer
`On`	Event or message handler

This appears relatively easy, just instantiating a **`Cstatic`** object (a simple window). We can use much the same techniques to create dialogs (**`CDialog`**) or drawing windows (**`CFrameWnd`**)

10.3 MFC class hierarchy

The following heirarchy diagram shows the MFC components.



10.4 Summary of topics

In this module, we introduced the following topics:

- MFC
- MFC class heirarchy
- Simple programming

Tutorial 8 - questions for week 14 (April 10, 2002)

1. Give a minimal menu driven application for MFC.
2. Compare the MFC message model with the Java Event model.
3. Outline a strategy for porting an MFC program to UNIX.
4. Outline a strategy for porting a Tcl/Tk program to MFC.

Bibliography

[1] J. P. Bowen, M. G. Hinchey, and D. Till, editors. *ZUM'97: The Z Formal Specification Notation, 10th International Conference of Z Users, Reading, UK, 3–4 April 1997*, volume 1212. Springer-Verlag, 1997.

[2] Stephen G. Eick. Engineering perceptually effective visualizations for abstract data.

[3] J. Jacky. *The Way of Z: Practical Programming with Formal Methods*. Cambridge University Press, 1997.

[4] Nancy Leveson and Clark S. Turner. An investigation of the therac-25 accidents. *IEEE Computer*, 26(7):18–41, July 1993.

[5] Bertrand Meyer. *Object Oriented Software Construction*. Prentice Hall, 1991.

[6] Roger S. Pressman. *SOFTWARE ENGINEERING A Practitioner's Approach*. McGraw-Hill, 4 edition, 1997.

[7] E. Swing. Flodar: Flow visualization of network traffic. *IEEE Computer Graphics and Applications*, 18(5):6–8, September 1998.

Appendix A

Extra notes on Tcl/Tk

This section includes some extra material related to the use of Tcl/Tk for developing GUI applications. In particular - constructing menu items, using the Tk Canvas and structured data items. There are pointers to some supplied reference material. Note the following points related to trying out Tcl/Tk:

- If you are using **cygwin-b20**, the wish interpreter is called **cygwish80.exe**. This file is found in the directory **/cygnus/cygwin-b20/H-i586-cygwin32/cygwish80.exe**. Make a copy of this file in the same directory, and call it **wish8.0.exe** for compatibility with UNIX Tcl/Tk scripts.
- In the first line of your tcl files, you should put **#!wish8.0**
- If you download the file **~cs3283/ftp/demos.tar** and extract it into **/cygnus**, you will have a series of Tcl/Tk widget examples in **/cygnus/Demos**. Change into the directory **/cygnus/Demos**, and type **./widget**.
- There is a Tcl/Tk tutor, and many learn-to-program-Tcl/Tk documents available at many sites on the Internet - if you continue to have trouble, you may wish to try them.

There is no substitute for just trying to program - set yourself a small goal, and discover how to do it in Tcl/Tk.

A.1 Tcl/Tk menus

The menu strategy is fairly simple -

1. Make up a frame for the menu
2. Add in the top level menu items
3. For each top level item, add in the drop-menu items
4. For each nested item, add in any cascaded menus.
5. Remember to pack it...

As an example, the following code creates a fairly conventional application with menus, a help dialog, and cascaded menu items.

CODE LISTING **Menus.tcl**

```
#!/usr/bin/wish

frame .mbar -relief raised -bd 2
pack .mbar -side top -fill x

frame .dummy -width 10c -height 100
pack .dummy

menubutton .mbar.file -text File -underline 0 -menu .mbar.file.menu
menu .mbar.file.menu -tearoff 0
.mbar.file.menu add command -label "New..." -command "newcommand"
.mbar.file.menu add command -label "Open..." -command "opencommand"
.mbar.file.menu add separator
.mbar.file.menu add command -label Quit -command exit
pack .mbar.file  -side left

menubutton .mbar.edit -text Edit -underline 0 -menu .mbar.edit.menu
menu .mbar.edit.menu -tearoff 1
.mbar.edit.menu add command -label "Undo..." -command "undocommand"
.mbar.edit.menu add separator
.mbar.edit.menu add cascade -label Preferences -menu .mbar.edit.menu.prefs
menu .mbar.edit.menu.prefs -tearoff 0
.mbar.edit.menu.prefs add command -label "Load default" -command "defaultprefs"
.mbar.edit.menu.prefs add command -label "Revert" -command "revertprefs"
pack .mbar.edit  -side left

menubutton .mbar.help -text Help -underline 0 -menu .mbar.help.menu
menu .mbar.help.menu -tearoff 0
.mbar.help.menu add command -label "About ThisApp..." -command "aboutcommand"
pack .mbar.help  -side right

proc aboutcommand {} {
    tk_dialog .win {About this program} "Hugh wrote it!"  {} 0 OK
}
```

A.2 The Tk canvas

The Tk canvas widget allows you to draw items on a pane of the application. Items may be tagged when created, and then these tagged items may be bound to events, which may be used to manipulate the items at a later stage.

This process is described in detail in Robert Biddle's "Using the Tk Canvas Facility", a copy of which is found at **~cs3283/ftp/CS-TR-94-5.pdf**.

Note also the use of dynamically created variable names (**node$nodes**).

Tutorial 5 - questions for week 5 (Feb 6, 2002)

All these questions relate to practical programming concerns in Tcl/Tk.

1. How can you create a cascade menu item where the cacaded items are radiobuttons (i.e. - only can select one at a time).
2. Tcl only supports a simple idea of variable scope. How can you create global variables? How can you create local variables? How can you refer to global variables within a proc?
3. How would you create structured type variables in Tcl? (i.e. a variable with various sub-components).
4. How do you change the name on the surrounding window decoration for an application?
5. How can you create a file-selection dialog box when you click on an 'open' menu item? How does your appliaction know which file was selected? How does your application know when no file is selected?
6. How can you change the cursor (to -say- a watch), when it moves over a particular item embedded in a canvas?

Further study

- TclTk widgets:
  http://www.comp.nus.edu.sg/~cs3283/ftp/CS-TR-94-5.pdf,

  http://www.comp.nus.edu.sg/~cs3283/ftp/demos.tar.

A.3 Assignment 3 - Implementation

I have moved assignment 4 here, as we have not yet got to Java/Swing.

In this assignment, you may work in a group of up to 4 people - or you may do it by yourself. The assignment is worth 30% of your assignment grade (10.5% of your final mark), and is due at 5:00 p.m. on Friday, 8th March, 2002 - please deliver to Hugh's room. The assignment is to be done using Tcl/Tk or Perl/Tk or C/Tk.

Task:

Your task is to implement and document a user interface for a new software analysis tool, which I will attempt to describe below. Note that this is a real task, and parts of this tool have already been prototyped. However I am looking for better ideas :)

The interface is intended to manage the overall process of an analysis of program source code. Beginning with some initial document, the developer uses (external) program analysis tools[1] to transform an existing document into a new document. This document in turn may be further transformed using the same, or different analysis tools.

The user of your application will use a range of different analysis tools, in different orders, and some analysis sequences will be useful, and others may not. However all transformations are to be recorded.

The first prototype of the user interface is shown in Figure A.1, however, your version need not look at all like this - my expectation is that you should be able to come up with better interfaces - I am a little worried about putting this one up, because It may *limit* you in your development (which I do not want to do). The interface maintains a library of documents. The activities performed here are:

- Entering new documents into the library from external source code.
- Transforming existing documents using a range of analysis programs.

The process of applying analysis programs to transform documents in the library leads to a library structure which is a directed graph without loops. This may be represented by a hierarchy diagram, and this sort of diagram is used as a basis for the interface.

The idea here is that each dot represents a document, with the directed arrows representing the application of an analysis program to a document. Operations on documents are done by selecting the document in the GUI interface, and then selecting actions (in this case from a menu). Once the external analysis program has created a new document, it is displayed as a dot in the interface, with its directed arrows.

[1] These tools have already been developed in a separate project, but may be run as commands using the **exec** Tcl facility. For example **prove1 <filein.txt >fileout.txt**.

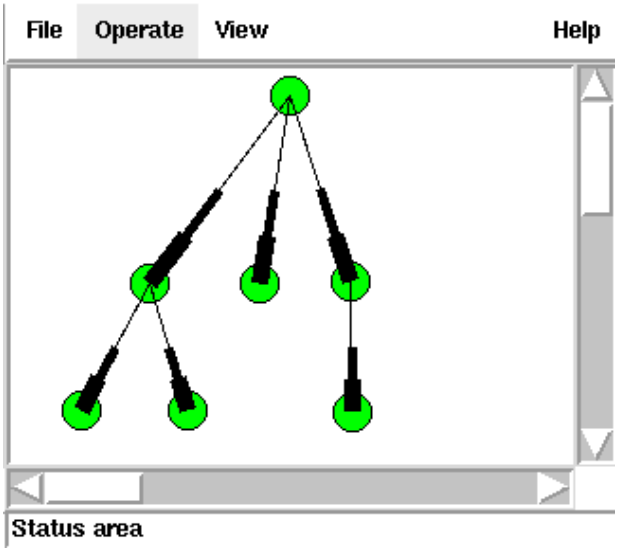


Figure A.1: Document management tool

The graphical view displayedshould be easy to navigate, and display a complete history of the program analysis.

Notes:

The following points form part of the functional specification of the user interface.

1. At any time you should be able to save and restore the state of the development (i.e. all the things visible, and the state of the library).
2. The program should be safe on failure - that is - if an external transformation tool dies for some reason, it will not affect the stability of your library and display.
3. The general flow of operation of the interface is you select (one or more) documents, and then select a single analysis tool, which is passed the selected documents, and returns a new document. The new display should reflect the new document's position in the library.
4. The document names are managed by your tool, which might name the documents according to some internal strategy (doc1, doc2, doc3 :). However - the tool should also maintain descriptions associated with each document, that may be entered by the user, and displayed later. (Perhaps something like - if you right-click on a document, you get the description).
5. The transforms between documents also may have attached descriptions, editable and displayable in a similar manner.
6. The descriptions should be time stamped.
7. The transforms should be time stamped.

Tips:

I'm just guessing here, but

1. You probably have to maintain a configuration file (or files) for the library - containing the state of the display, and the names and descriptions of the elements in the display.
2. Rather than hard coding which transform tools are to be used, you might read a configuration file containing the tools and which menu/button items they are to be associated with.
3. You may dummy up tools, (tool1, tool2 tool3 and so on), by just using a simple program like cat or copy, to make an exact duplicate of the document.

Deliverables:

- A title page containing your names and matriculation numbers.
- A ten to twenty page document containing
  - A brief summary of the overall design constraints
  - An overview of the interface design
  - A user manual for the interface
- A disk containing the code, with a (small) README file to explain how I am supposed to run your software.

Note that this assignment requires you to implement the application, not just to design one.

Assessment:

The assessment is as follows:

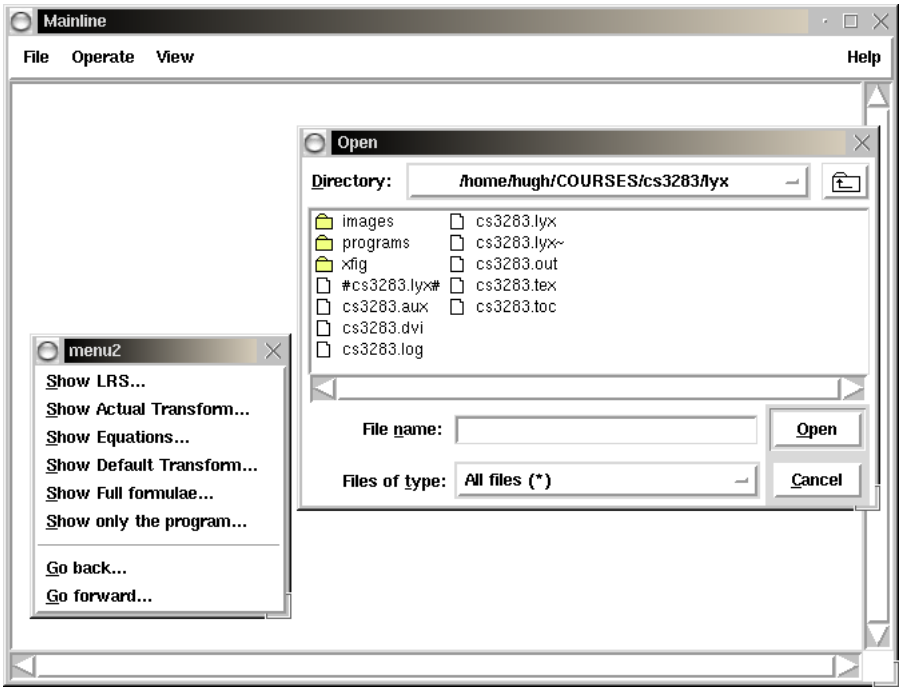
Documentation	25%
Code style/quality	25%
Operation of the interface	50%

Appendix B

Case study: GUI implementation

Here is an example of a *quick-n-dirty* GUI interface for a prototype application used for analysis of software. The prototype application uses libraries of C and fortran (!) routines to do numerical analysis, and a script written in perl to glue these all together, and transform complex data structures from binary to readable forms.

The GUI interface is easy to use and represents about 5% of the effort in developing the application.



B.1 Perl/Tk code

Perl has a Tk module - the perl interface to Tk - and it provides most of the user interface. The mainline is quite simple:

CODE LISTING **mainline.pl**

```
use Tk;
my $currentslice = 0;
my $currentpp    = 0;
my $disptype     = 2;
my $main = new MainWindow;

<<SetupMenu>>
<<SetupFileMenu>>
<<SetupEditMenu>>
<<SetupViewMenu>>

$main->configure(-menu =>$menubar);

<<SetupScrolledMainArea>>

MainLoop;

<<FileOpenDialogBox>>
```

We then set up the menu bar. In this code we use the cascade model.

CODE LISTING **SetupMenu.pl**

```
$menubar = $main->Menu;
$filemenu = $menubar->cascade(-label=>"File" );
$editmenu = $menubar->cascade(-label=>"Operate" );
$viewmenu = $menubar->cascade(-label=>"View" );
$helpmenu = $menubar->cascade(-label=>"Help");
$helpmenu->command(-command => \&about_choice,
                   -label => "About TkMenu...",
                   -underline => 0);
```

And then we set up the menu items in each menu. First the file menu:

CODE LISTING **SetUpFileMenu.pl**

```
$filemenu->command(-command => sub { fileDialog( $main, 'open' );
                                     printf "Opening $thisfile\n";
                                     readfile($thisfile);
                                     writefile($thisfile . ".ppx");},
                   -label => "Open...",
                   -underline => 0);
$filemenu->separator;
$filemenu->command(-label => "Exit",
                   -command => \&exit_choice,
                   -underline => 1);
```

Then the edit menu:

CODE LISTING **SetUpEditMenu.pl**

```
$editmenu->command(-command => sub {Tp($currentslice,1,1);},
                   -label => "Crank with widening...",
                   -underline => 0);
$editmenu->command(-command => sub {Tp($currentslice,1,10);},
                   -label => "Crank with widening (10X)...",
                   -underline => 0);
$editmenu->command(-command => sub {Tp($currentslice,0,1);},
                   -label => "Crank...",
                   -underline => 0);
$editmenu->command(-command => sub {Tp($currentslice,0,10);},
                   -label => "Crank (10X)...",
                   -underline => 0);
$editmenu->command(-command => sub {Cousot($currentslice);},
                   -label => "Cousot...",
                   -underline => 0);
$editmenu->separator;
$editmenu->command(-command => sub {widening($currentslice);},
                   -label => "Widen...",
                   -underline => 0);
```

Finally the view menu:

CODE LISTING **SetUpViewMenu.pl**

```
$viewmenu->command(-command => sub {$disptype=0;display($currentslice,0);},
                   -label => "Show LRS...",
                   -underline => 0);
$viewmenu->command(-command => sub {$disptype=1;display($currentslice,1);},
                   -label => "Show Actual Transform...",
                   -underline => 0);
$viewmenu->command(-command => sub {$disptype=2;display($currentslice,2);},
                   -label => "Show Equations...",
                   -underline => 0);
$viewmenu->command(-command => sub {$disptype=3;display($currentslice,3);},
                   -label => "Show Default Transform...",
                   -underline => 0);
$viewmenu->command(-command => sub {$disptype=4;display($currentslice,4);},
                   -label => "Show Full formulae...",
                   -underline => 0);
$viewmenu->command(-command => sub {$disptype=5;display($currentslice,5);},
                   -label => "Show only the program...",
                   -underline => 0);
$viewmenu->separator;
$viewmenu->command(-command => sub { if ($currentslice>0) {
                                        $currentslice=$currentslice-1;
                                        if ($currentpp==0) {
                                           $currentpp=$codesize;
                                        }
                                        $currentpp=$currentpp-1;
                                        display($currentslice,$disptype);
                                     }},
                   -label => "Go back...",
                   -underline => 0);
$viewmenu->command(-command => sub { if ($currentslice+1<$maxslice){
                                        $currentslice=$currentslice+1;
                                        $currentpp=$currentpp+1;
                                        if ($currentpp==$codesize) {
                                           $currentpp=0;
                                        }
                                        display($currentslice,$disptype);
                                     }},
                   -label => "Go forward...",
                   -underline => 0);
```

Here is the code for an OpenFile dialog box:

CODE LISTING **FileOpenDialogBox.pl**

```
sub exit_choice {
   exit;
}

sub fileDialog {
   my $w = shift;
   my $operation = shift;
   my $types; my $file;
   @types = (["Code files", '.pp'],
             ["Work files", '.ppx'],
             ["All files", '*'] );
   $file = $w->getOpenFile(-filetypes => \@types);
   if (defined $file and $file ne '') {
      $thisfile = $file;
   }
}
```

You can find a copy of this code at

http://www.comp.nus.edu.sg/~cs3283/ftp/original.pl

It may be run by typing "**perl original.pl**".