Chapter 2

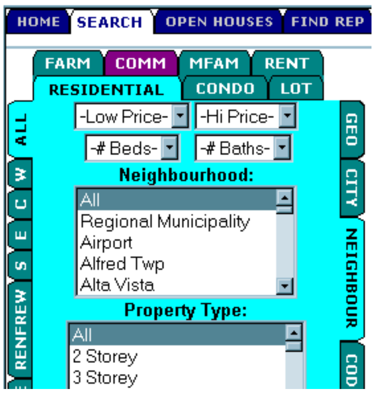
Design

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 <u>know</u> 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.)

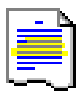
2.2.1 Role of designer

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:

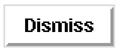
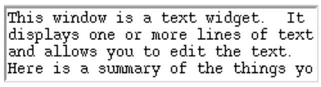
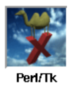
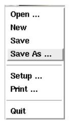
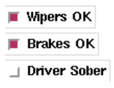
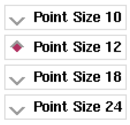
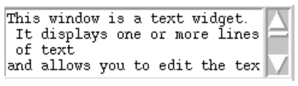
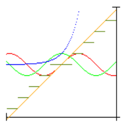
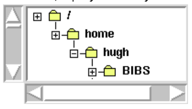


[1]By contrast, many other engineering disciplines *do* have generally accepted techniques to be used.

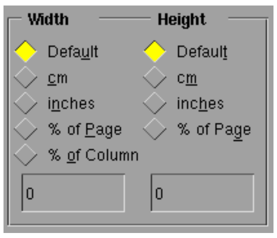
Here there is a clear relationship between the icon and the text file. We also have higher level abstractions - for example, the desktop and wastebasket metaphors. The designer needs to become familiar with successful abstractions like these, so that they can be used and so that new abstractions may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

Button	Testbox	Label
	 This window is a text widget. It displays one or more lines of text and allows you to edit the text. Here is a summary of the things yo	
Menu	**Checkbox**	**Radiobutton**
	 [✓ Wipers OK] [✓ Brakes OK] Driver Sober	 Point Size 10 [✓ Point Size 12] Point Size 18 Point Size 24
Scrollbar	**Graph**	**Directory Tree**
 This window is a text widget. It displays one or more lines of text and allows you to edit the tex		DirTree, diplay directory tree. 

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


Width: [✓ Default] cm inches % of Page % of Column 0
Height: [✓ Default] cm inches % of Page 0

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

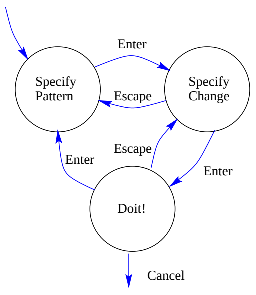
In general, the designer somehow imagines and proposes common scenarios[2] for the use of the software, and

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.

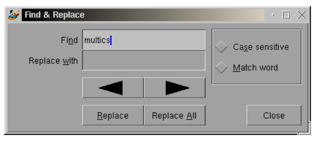
There are a range of tools we can bring to bear on these design problems. For example:

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

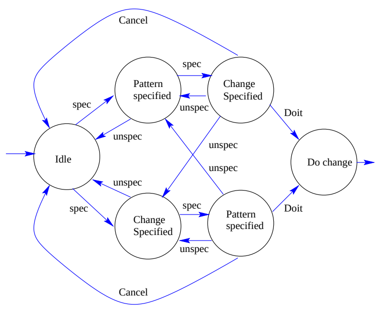
A simple older-style user interface for *find-and-replace* might involve first asking for the pattern of text to find, and then for text to replace it with. A *state-diagram* would look like this:



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:

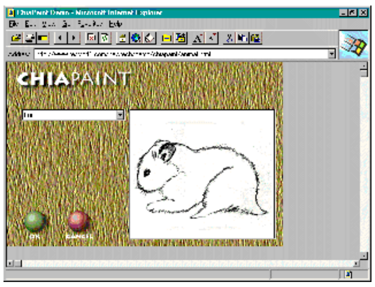


[2]Scenarios=Use_cases. Use_cases=scenarios.

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

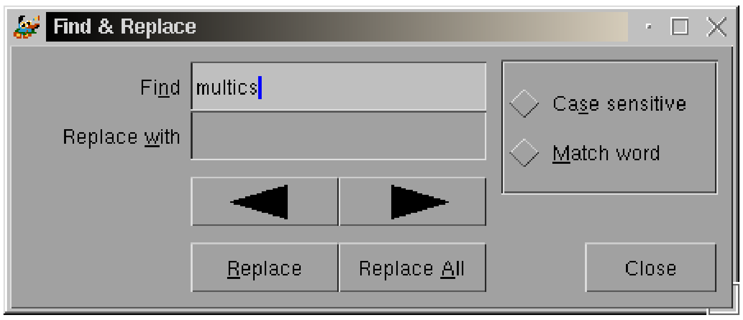
It describes the use of formal specification tools and notations in constructing the interface to a nuclear reactor.

2.3.3 Examples of GUI designs

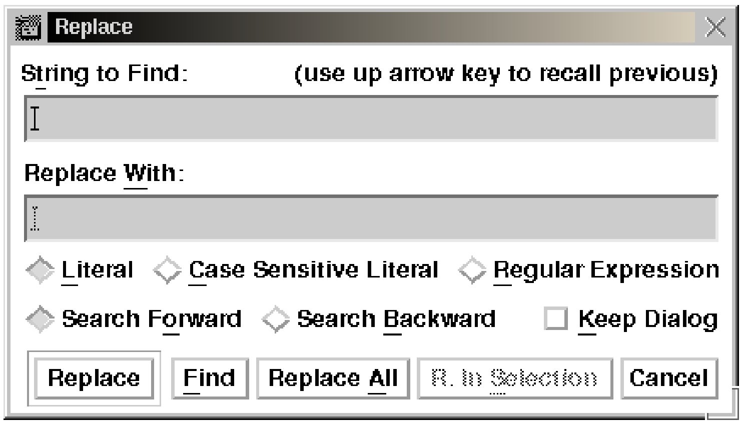
Here are some examples of different designs for similar things, with some brief comparative comments:

Dialog boxes for find-and-replace:

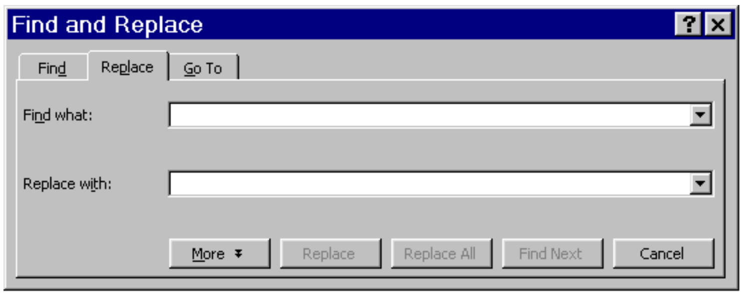
This dialog box in the **LyX** word processor was confusing the first time I tried it:



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

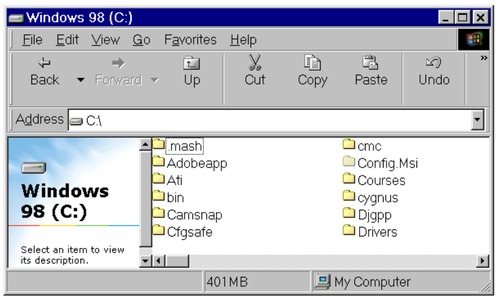


This is the **Word** dialog box.

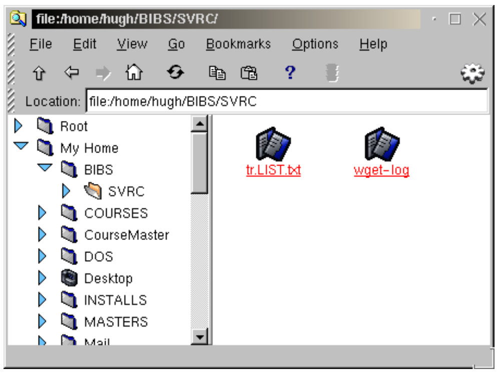


File system navigation:

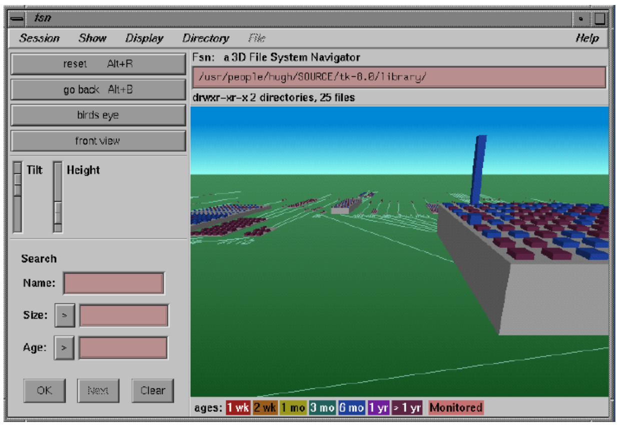
The familiar **win98** file manager borrows the basic concept from MAC file managers, and is quite easy to use.



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


file:/home/hugh/BIBS/SVRC/
File Edit View Go Bookmarks Options Help
Location: file:/home/hugh/BIBS/SVRC
Root
My Home
BIBS
SVRC
COURSES
CourseMaster
DOS
Desktop
INSTALLS
MASTERS
Mail
tr.LIST.txt
wget-log

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.


fsn
Session Show Display Directory File Help
reset Alt+R
go back Alt+B
birds eye
front view
Tilt
Height
Search
Name:
Size:
Age:
OK
Next
Clear
Fsn: a 3D File System Navigator
/usr/people/hugh/SOURCE/tk-8.0/library/
drwxr-xr-x 2 directories, 25 files
ages: 1 wk 2 wk 1 mo 3 mo 6 mo 1 yr > 1 yr Monitored

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:

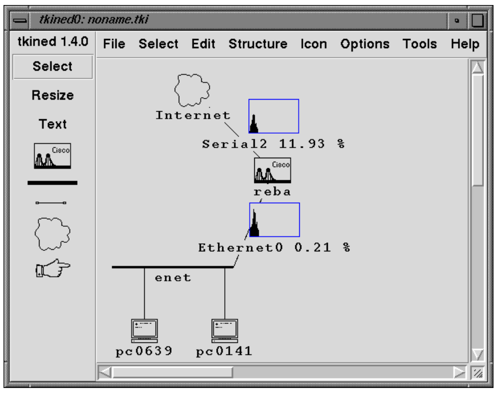
1. User requirement
2. Environment
    (a) Software constraints
    (b) Other constraints
3. Interface design
    (a) Overview
    (b) Interface description
        i. Drill-down and other displays
        ii. Encoding
4. Testing methodologies

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.

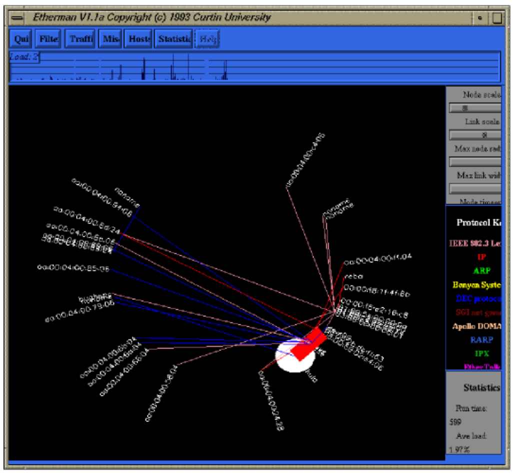
Graphs and diagramming:

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.

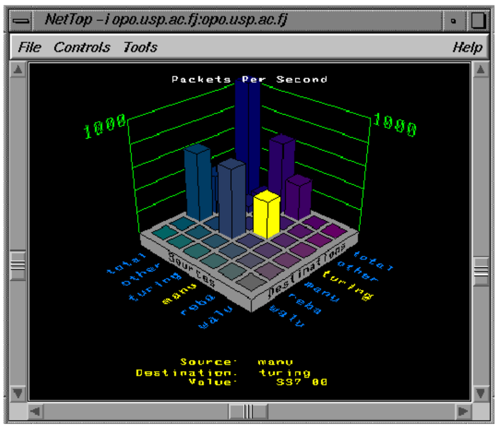


When running, **etherman** collects and displays graphically the ethernet traffic on the directly connected network. In the figure the display shows several hosts communicating with a range of protocols. A host to the bottom right of the display is generating a lot of traffic.

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/

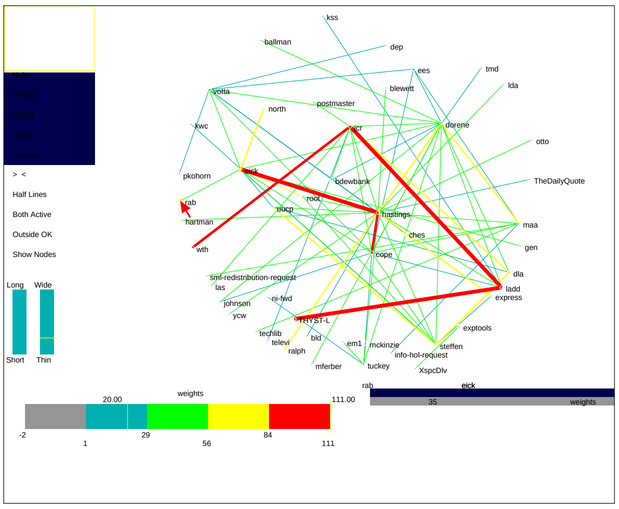
3D graph:



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:

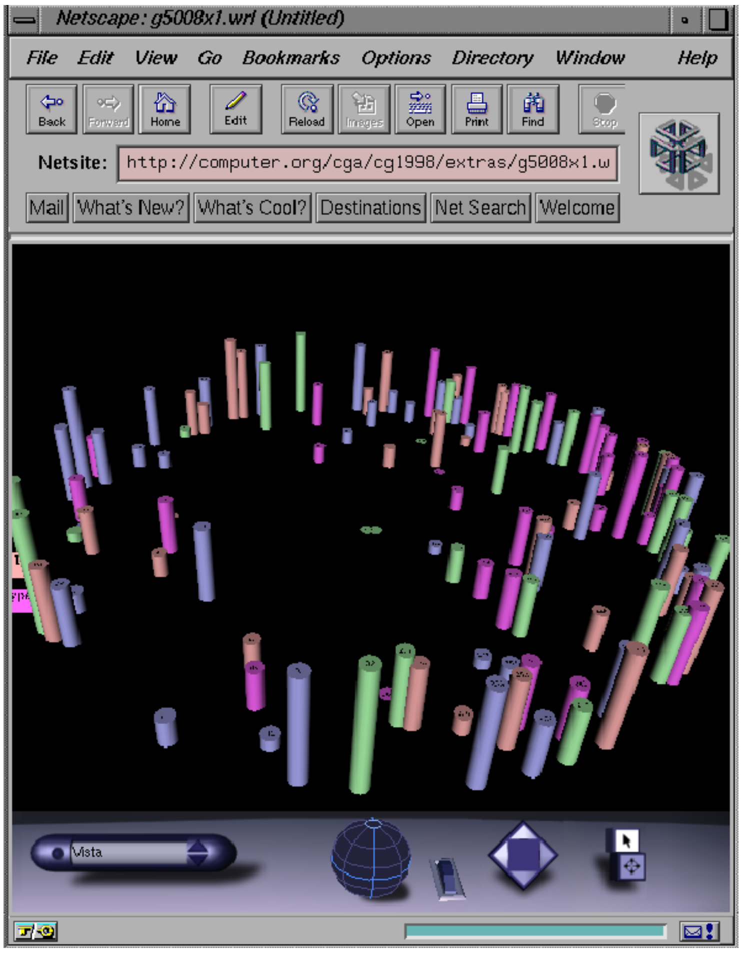
SeeNet was developed as part of a continual research effort in network data analysis at AT&T.



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** (<u>Flo</u>w Ra<u>dar</u>) 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.

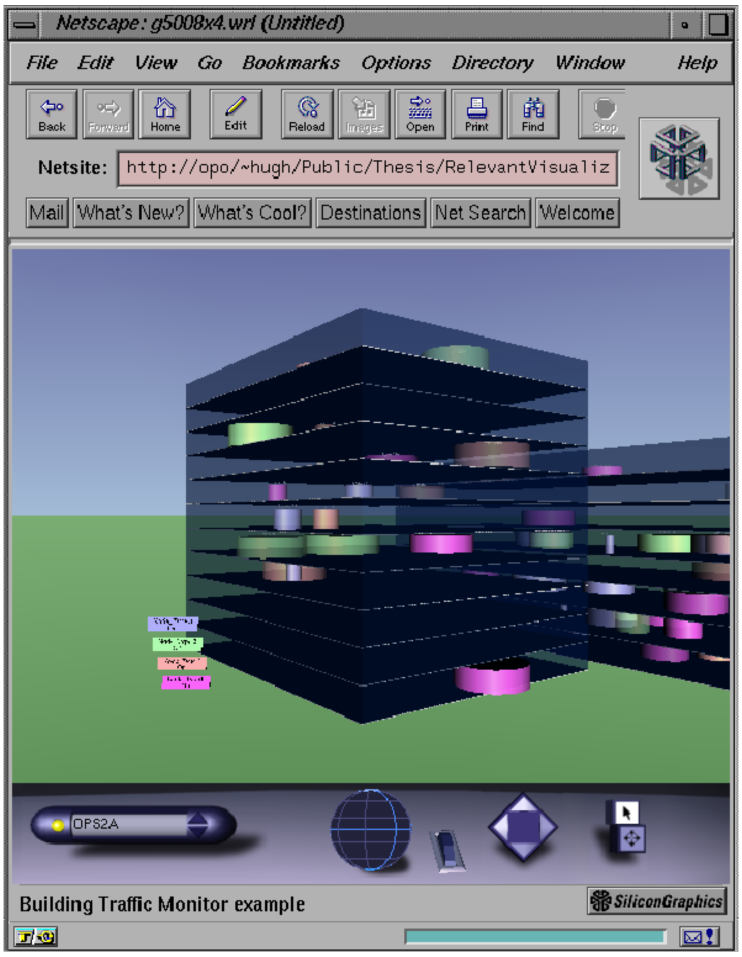

Netscape: g5008x1.wrl (Untitled)
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find Stop
Netsite: http://computer.org/cga/cg1998/extras/g5008x1.w
Mail What's New? What's Cool? Destinations Net Search Welcome
Vista

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.