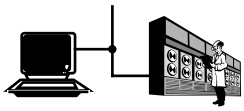
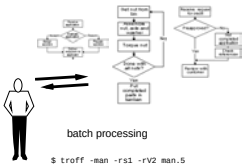
Why Use Application Frameworks 1

character-oriented terminals



conversation metaphor



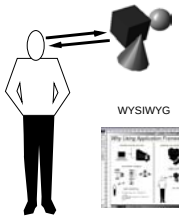
batch processing

```
$ troff -man -rs1 -rV2 man.5

.SH NAME
ls \- list the contents of a directory
.SH SYNOPSIS
.B ls
[
```

graphical user interface workstations

model world metaphor



WYSIWYG

Why Use Application Frameworks 2

graphical user interfaces (GUI) make applications

+ easy to learn
+ intuitive to operate

BUT:

- take up to 88% of programming time
  due to semantic gap between window system and application

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
  …
  XSetStandardProperties (display,top,…,&myhint);
  mygc = XCreateGC (display,top,0,0);
  XSelectInput (display,top,ButtonPressMask |…|ExposureMask);
  XMapRaised (display, top);/* Window mapping */
  XMapWindow (display, win1);
  while (done == 0){ /* Main event-reading loop */
    XNextEvent (display, &myevent); /* Read the next event */
    switch (myevent.type){/* process keyboard mapping changes */
    case : MappingNotify
      XRefreshKeyboardMapping (&myevent);
    …
```

Why Use Application Frameworks 3

Toolkits with APIs

+ widgets
+ standard look (~ feel ?)
+ application and window system are decoupled and thus more portable

It is crucial, whether toolkits are extensible in a structured manner.

Why Use Application Frameworks 4

Strategies to enhance structuring/flexibility of toolkits:

finer granularity of toolkit functions

```
MakeWinArea(…);AddBorder(…);AddVertScrollBar(…);…
```

configurability of toolkit functions

```
MakeWindow(x,y,w,h,border,bothScrolls,closeBox,…);
```

modification of toolkit's source code

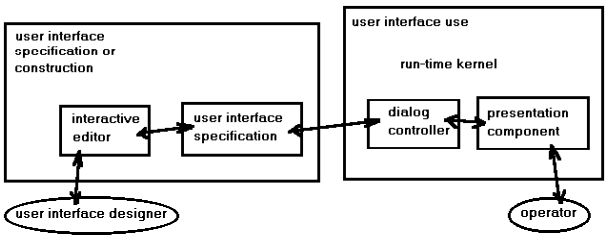
```
$ vi XYZtoolkit.[h,c]
```

program sceletons

```
$ vi XYZsceleton.c
```

Why Use Application Frameworks 5

User Interface Management Systems (UIMS)



- hardly applicable when high semantic feedback required
- hardly extensible
- an additional language has to be learned by the programmer

Why Use Application Frameworks 6

Bertrand Meyer's "Open/Closed" Principle

• A system's components have to be ***closed***, so they can be offered with a fixed and defined ***interface*** in a library.

• A system's components have to be ***open***, so they can be ***adapted*** to new requirements without affecting already existing clients.

Why Use Application Frameworks 7

Bertrand Meyer's "Open/Closed" Principle

Classes are closed

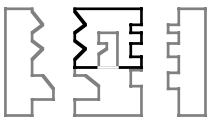
"**black box**" reuse or reuse by instantiation
client uses a class without changing it

```
Button *b= new Button("Print");
Window *w= new Window(400,300);
w->Add(b);
w->Show();
```

Classes are open

"**white box**" reuse or reuse by inheritance
client extends/changes a class when subclassing

```
class myButton : public Button {
void Draw() {
    Button::Draw();
    // draw fancy border
    // …
}
```



Why Use Application Frameworks 8

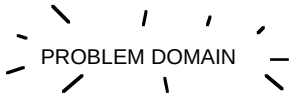
Concepts of Object-Oriented Programming Languages

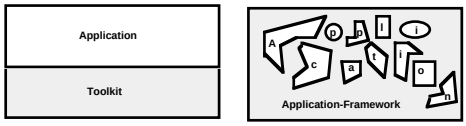
- data abstraction
  toolkit's components = classes
- inheritance
  programming by difference:
  structured approach to extensibility
- polymorphism
  lean programming interface
- dynamic binding
  structured replacement for hooks

Why Use Application Frameworks

(Finally!) Frameworks

- not independent but related classes (design patterns)
- finding abstractions by looking at several applications of same





Visual Programming

Reuse in Application Frameworks

White-box Reuse

Reuse by Subclassing
The internals of parent classes are visible.
Dependency of changes in the implementation / interface.

Black-box Reuse

Object Composition
The interfaces of components are known, their internals aren't.
Dependency of changes in the interfaces only.
Configurability at runtime

Visual Programming

Visual Object Composition
Interactive connecting of pluggable components.
Advantages of black-box reuse combined with comfortable handling, i.e. no coding, compiling, or linking, easy specification.

Visual Programming

Building blocks

Data Units

Data Repository
Read/write (Single Data Item, Data Array, NetCDF, etc.)
Read only (Constants, Timer, Clock)
User interface component (menu, slider, checkbox, radio button, etc.)
Data Filter
Basic arithmetic functions (Scaler, Shifter)
Mathematical functions (trigonometric, logarithmic, etc. and their inversions)
Conditional functions (if-then-else, threshold)
Data Mapper
2D, 3D, Audio, MIDI, TextTable

Data Ports

Dependent
Independent

Connectors

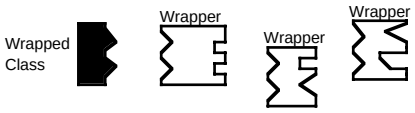
Bi-directional

Visual Programming

Realization

- How to make the Framework's classes available in the visual programming environment

Wrapper design pattern



- How to trigger messages between interconnected components

Special requirement: bidirectionality
Mathemathical Model:

$$(y_1, y_2, \ldots, y_n) = f(x_1, x_2, \ldots, x_m);$$

$$(x_1, x_2, \ldots, x_m) = f^{-1}(y_1, y_2, \ldots, y_n);$$

Visual Programming

Data Types

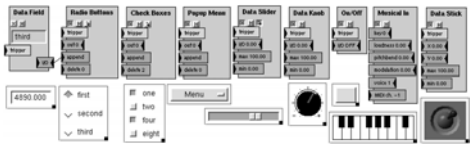
- short
- long
- float
- double
- char
- string

Integration with MET++

Integration on two levels

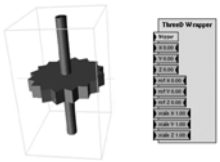
User Interface Builder

Drag & Drop of GUI-components
Separation of GUI components and visual program



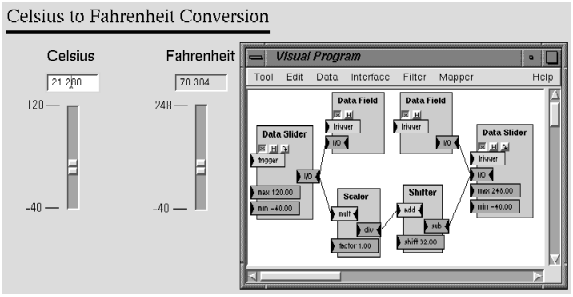
Media Wrappers

Visual Objects
Temporal Events



Integration with MET++

Separation of User Interface and Visual Program



Bidirection

Bidirectional data flow

Repositories

Reading
Writing

User Interface Components

Manipulation by user
Setting by visual program

Filters

Function, e.g. e^x
Inverse, e.g. ln(x)

Mappers

Setting by visual program
Manipulation by user

Applications

17

Visual Programming for Multimedia

Visualization/Sonification of Data

self-describing input file format (netCDF)

```
netcdf xyzArr {
dimensions:
      index= 6;
variables:
      double xpos (index);
      double ypos (index);
      double zpos (index);
data:
      xpos= 1.0, 3.0, -2.0, 4.0, -5.0, 4.0;
      ypos= 1.0, 3.0,  1.0,-3.2,  2.0, 4.0;
      zpos= 0.6, 3.1, -4.0,-3.2,  3.0, 1.0;
}

netcdf plane6x4 {
dimensions:
      x= 6;
      y= 4;
variables:
      long zpos (x,y);
data:
      zpos=     0,1,1,2,1,1,1,1,1,0,4,1,
                1,4,0,1,1,1,1,1,2,1,1,3;
}
```

Applications

18

Mapping

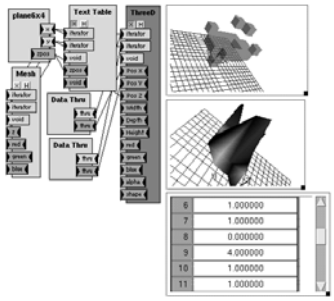
Visualization/Sonification of Data (contd.)

generic mappers

```
netcdf plane6x4 {

dimensions:
        x = 6;
        y = 4;
variables:
        float zpos (x,y);
data:

    zpos=
          0,1,1,2,
          1,1,1,1,
          1,0,4,1,
          1,4,0,1,
          1,1,1,1,
          2,1,1,3;
}
```



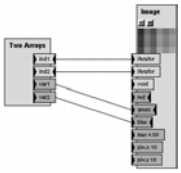
Applications

19

Mapping of data

Iterator Port

Mapping



Applications

20

Visual Programming for Multimedia

CAD

- Parametric Construction
  - Constraining of interactive manipulation
  - Maintaining relationships between 3D objects

Redundancy-Free Animation Specification

- Maintaining relationships between 3D objects
- Animation of Keyplayers only

Executable Documents

- Embeded Functionality in a Documentation
- WWW
- better than scripting
- (a JAVA rival?)

Web-Sites

ET++ / MET++

ET++

ftp://ftp.ubilab.ubs.ch/pub/paper/Wei94.ps.Z
ftp://ftp.ubilab.ubs.ch/pub/paper/german-tutorial.tar.gz
ftp://ftp.ubilab.ubs.ch/pub/ET++/paper/

MET++/ET++

ftp://ftp.ifi.unizh.ch/pub/projects/met++/papers/