Overview

Fresco is an object-oriented application programming interface (API) for graphical user interfaces, covering functionality in Xlib and Xt, and adding structured graphics and application embedding

Fresco is under development by the X Consortium as an open, multi-vendor standard

Fresco fundamentals

Standard object model

Resolution independence

Graphical embedding

Standard object model

- *OMG's CORBA*
- *IDL for defining object interfaces*
- *Choice of implementation languages*
- *Object references instead of pointers*
- *Distribution through ORB*

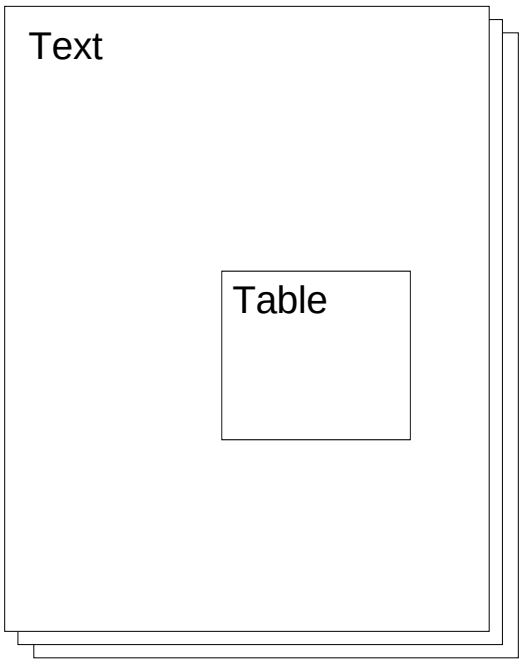
Resolution independence

- *Floating-point coordinates*
- *Stencil/paint operations*
- *Printing (generation of PostScript)*
- *Device-independent graphics state*

Graphical embedding

- *Structured graphics traversals*
- *Focus, menubar, palette sharing*
- *Edit-in-place*
- *3D ”friendly”*
- *Cross address-space calls ”for free”*

Embedding example



Distributed embedding

- *X supports remote applications, but we want remote objects "inside" a window*
- *Applications become objects*
- *User composes objects*
- *Objects can be implemented in different address spaces or on different machines*

Current status

- *Spec draft available*
  - *About 40 interfaces*
- *Sample implementation in progress*
  - *ORB-less IDL-to-C++ translator (ix)*
  - *Simple threads interface*
  - *Layout, Figure, Widget kits*
- *Usable implementation targetted for R6*

Object system

- *Language support – IDL*
- *Transparency*
- *Services*

IDL – Interface Definition Language

More abstract than C++
(no implementation details)

Automatically bound to multiple languages

Supports distribution

IDL example

```
interface Window {
   void map();
   void unmap();
   boolean is_mapped();
   void raise();
   void lower();
};
```

C++ binding

Typedefs, structs, enums, constants

Interfaces –> abstract classes

Operations –> pure virtual functions

Attributes –> set/get function pairs

Exceptions –> exceptions or extra param

C++ example

```
class Window {
protected:
   Window();
   virtual ~Window();
public:
   virtual void map() = 0;
   virtual void unmap() = 0;
   virtual unsigned char is_mapped() = 0;
   virtual void raise() = 0;
   virtual void lower() = 0;
};
```

Parameter passing

IDL defines **in**, **out**, *and* **inout**

Semantics are

in – pass value

out – return value

inout – pass and return value

For C++, binding is

in – copy or const ref

out – ref

inout – ref

Object references

- *Like pointers, but (potentially) across address spaces*
- *Interface type is always reference*
  void set_transient(in Window w)
  –> void set_transient(WindowRef w);
- *ORB tracks object location*
- *All operations are virtual function calls*
  vtbl points to local or stubs

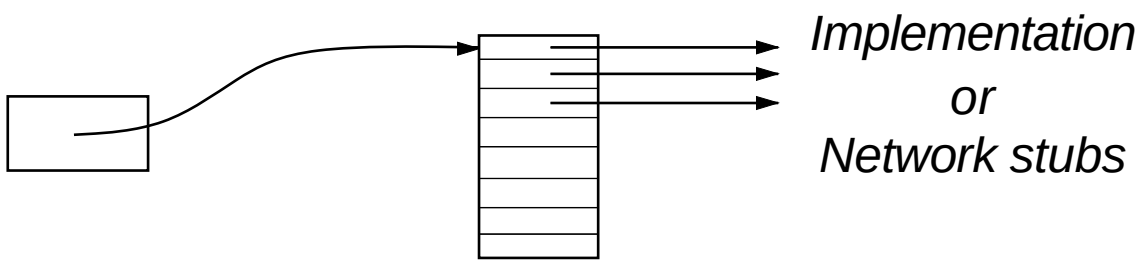
Transparency

Local or remote machine

Direct memory access or inter-process communication

	Local	*Remote*
Memory	*Function call*	*Page over network*
IPC	*Context switch*	*Network message*

Transparent object references



Array of function pointers

Object services

Naming contexts

Associations among objects

Update notification

Persistence

Concurrency control

Security

A bigger example

Sample interface translator (ix)

IDL for Transform (transformation matrix)

C++ TransformImpl class

Interface translator (ix)

- *Not standard – sample implementation*
- *Optional – can always do manually*
- *Produces .h file from .idl file*
- *Edits .cxx file based on .idl files*
- *Encapsulated in Imake macros*
  - *MakeHeader, EditImplHeader, EditImpl*

Generating a header file

MakeHeader(transform)

Runs cpp to process #include's

Use -include option to generate #include

Write out transform.h, mark read-only

Bracket definitions with #ifndef
to handle redundant #include's

Example: transform.idl

```
interface Transform {
   typedef Coord Matrix[4][4];

   Transform clone();
   void load(in Transform::Matrix m);
   void load_identity();
   void store(out Transform::Matrix m);

   boolean equal(in Transform t);
   boolean identity();

   void scale(in Vertex v);
   void rotate(in float angle, in Axis a);
   void translate(in Vertex v);

   void premultiply(in Transform t);
   void postmultiply(in Transform t);
   void invert();

   void transform(inout Vertex v);
   void inverse_transform(inout Vertex v);
};
```

Generated transform.h

```
/* DO NOT EDIT – Automatically generated ... */

#ifndef transform_h
#define transform_h

class Transform;
typedef Transform* TransformRef;

class Transform {
protected:
   Transform();
   virtual ~Transform();
public:
   typedef Coord Matrix[4][4];
   virtual TransformRef clone() = 0;
   virtual void load(Transform::Matrix m) = 0;
   virtual void load_identity() = 0;
   virtual void store(Transform::Matrix m) = 0;
      ...
```

Defining an implementation

Subclass from generated class or
from another implementation class

Edit subclass source with ix -filter
EditHeaderImpl(xform-impl,transform.idl)
EditImpl(transform,transform.idl)

Use annotations to specify filtering

```
//+ Interface::Operation
//+ Interface::Attribute[=?]
//+ Subclass(Interface::Operation)
//+ Subclass(Interface::Attribute[=?])
```

Example: Initial xform-impl.h

```
class TransformImpl : public Transform {
public:
  TransformImpl();
  TransformImpl(Transform::Matrix m);
  virtual ~TransformImpl();

  //+ Transform::clone
  //+ Transform::load
    ...
private:
  Transform::Matrix mat_;
  boolean identity_ : 1;
  boolean modified_ : 1;

  void init();
  void update();
  Coord det();
};
```

Filtered xform-impl.h

```
TransformRef clone(); //+ Transform::clone
void load(Transform::Matrix m); //+ Transform::load
void load_identity(); //+ Transform::load_identity
void store(Transform::Matrix m); //+ Transform::store
boolean equal(TransformRef t); //+ Transform::equal
boolean identity(); //+ Transform::identity

void scale(const Vertex& v); //+ Transform::scale
void rotate(float angle, Axis a); //+ Transform::rotate
void translate(const Vertex& v); //+ Transform::translate
  ...
```

Initial transform.cxx

```
#include “xform-impl.h”
   ...
//+ Transform
//+

TransformImpl::TransformImpl() { init(); }
TransformImpl::~TransformImpl() { }

//+ TransformImpl(Transform::clone)
{
   return new TransformImpl(mat_);
}

//+ TransformImpl(Transform::identity)
{
   if (modified_) {
      update();
   }
   return identity_;
}
```

Filtered transform.cxx

```
//+ Transform
Transform::Transform() { }
Transform::~Transform() { }
//+

//+ TransformImpl(Transform::clone)
TransformRef TransformImpl::clone() {
   return new TransformImpl(mat_);
}

//+ TransformImpl(Transform::identity)
boolean TransformImpl::identity() {
   if (modified_) {
      update();
   }
   return identity_;
}
```

Creating objects

- *Constructors are global functions*
  - *Specify implementation class*
  - *Imply address space*
  - *Inappropriate for interfaces*
- *Can use operation on another object*
- *Kits create certain kinds of objects*
  - *Widgets, figures, layout*
- *FrescoKit accesses other kits*
- *Global function for bootstrapping FrescoKit*

Object system summary

- *IDL for interfaces*
  - *No data members, inlines*
  - *No pointers to functions*
  - *No overloading, templates, const member functions*
- *C++ for implementations*
  - *Parameters are abstract*
- *Create objects with kits*
- *Cleaner, more extensible design*
- *Support for distibution, multiple languages*

Resolution independence

Coordinates and vertices

Regions

Display management

Drawing and printing

Coordinates

```
enum Axis { X_axis, Y_axis, Z_axis };
typedef float Coord;
typedef long PixelCoord;
struct Vertex { Coord x, y, z; };
typedef float Alignment;
```

Transformations

- *Objects assume real-world units (inch, mm, pt, …)*
- *Units may be scaled by parent*
- *Relative coordinates – object-defined*
- *Absolute coordinates – fully-transformed*

Regions

Describe an arbitrary shape with origin

May be only bounding box

May be full solid geometry

Region interface

```
interface Region {
   struct Span {
      Coord begin, end;
      Coord origin, length;
      Alignment alignment;
   };

   Region clone();
   boolean contains(in Vertex v);
   boolean contains_plane(in Vertex v, in Axis a);
   boolean intersects(in Region r);

   void merge_intersect(in Region r);
   void merge_union(in Region r);
   void subtract(in Region r);
   void transform(in Transform t);

   void span(in Axis, out Span s);
   void merge_span(in Axis a, in Span s);
};
```

Display management

Display – keyboard, mouse, and screens

DisplayStyle – customizable information

Event – low-level device event

Screen – output area

Window – opaque portion of screen

WindowStyle – customizable information

DisplayKit – creates objects

DisplayKit interface

```
interface DisplayKit {
   attribute Display display;
   attribute Screen screen;

   Display open_default();
   Display open(in String name);

   Window application(in Frame f);
   Window top_level(in Frame f, in Window group_leader);
   Window transient(in Frame f, in Window w);
   Window popup(in Frame f);
   Window icon(in Frame f);

   long main(in Frame f);
   long main_window(in Window );
};
```

Drawing and Printing

Brush – line width and pattern

Color – maps to device RGB

Font – character shapes

Raster – bitmaps and images

Painter – 2d stencil/paint operations

DrawingKit – create paint objects

Drawing Kit interface

```
interface DrawingKit {
   typedef sequence<long> Data;

   Brush brush(in Coord width);
   Brush dither_brush(in Coord width, in long pattern);
   Brush patterned_brush(
      in Coord width, in DrawingKit::Data pattern
   );

   Color color_rgb(
      in Color::Intensity red,
      in Color::Intensity green,
      in Color::Intensity blue
   );
   Color find_color(in String name);

   Font find_font(in String fullname);
   Font find_font_match(
      in String family, in String style, in Coord ptsize
   );
```

DrawingKit interface (cont'd)

```
    Raster bitmap_file(in String filename);
    Raster bitmap_data(
      in DrawingKit::Data data,
      in Raster::Index rows, in Raster::Index columns,
      in Raster::Index origin_row,
      in Raster::Index origin_column
    );
    Raster bitmap_char(in Font f, in CharCode c);
    Raster raster_tiff(in String filename);

    Transform identity_transform();
    Transform transform(in Transform::Matrix m);

    Painter printer(in String output);
};
```

Painter interface

```
interface Painter {
   Painter clone();
   Coord to_coord(in PixelCoord p);
   PixelCoord to_pixels(in Coord c);
   Coord to_pixels_coord(in Coord c);

   void begin_path();
   void move_to(in Coord x, in Coord y);
   void line_to(in Coord x, in Coord y);
   void curve_to(
      in Coord x, in Coord y, in Coord x1, in Coord y1,
      in Coord x2, in Coord y2
   );
   void close_path();

   void fill(in Color c);
   void stroke(in Color c, in Brush b);
   void clip();
```

Painter interface (cont'd)

```
    void character(
       in Font f, in CharCode ch, in Coord width,
       in Color c, in Coord x, in Coord y
    );
    void image(in Raster r, in Coord x, in Coord y);
    void stencil(
       in Raster r, in Color c, in Coord x, in Coord y
    );

    attribute Transform matrix;
    void push_matrix();
    void pop_matrix();
    void transform(in Transform t);
    void push_clipping();
    void pop_clipping();
    boolean is_visible(in Region r);

    void comment(in String s);
    void page_number(in String s);
};
```

Resolution independence summary

Coordinate units

Regions, vertices

Display objects

Drawing and printing

Interfaces

Brush
Color
Cursor
Display
DisplayKit
DisplayStyle
DrawingKit
Event
Font
LayoutKit
Painter
Raster
Region
Screen
Transform
Window
WindowStyle

Graphical embedding

Structured graphics

Application object embedding

Glyphs – objects that draw

Viewers – objects that interact with user

Frames – top-level viewer

Structured graphics

- *Graphical objects*
  - *Shapes, properties, transforms*
  - *Overlays, transparency*
  - *Double-buffering*
  - *Sharing instances*
- *Draw, pick, bounding box*
- *Grouping into composites*
- *Attribute propagation*
- *Transcription – save to file*

Embedding

Graphical output

Geometry management

Input handling

Attribute propagation

Focus management

Menubar, palette sharing

Transcription

Common framework

Attribute propagation
Transcription

Graphical objects (sharing), output
Grouping into composites
Geometry management

Input handling
Focus management
Menubar, palette sharing

Advantages of common framework

Simplicity – fewer overall concepts

Scalability – span range of granularity

Interoperability – mix and match

Flexibility – 3d friendly

Power – editing rotated text in drawing

Attribute management

Separate for independent use

Loosely-coupled to structured graphics

Tightly-coupled to embedding

Action – callbacks

Observable, Observer – notification

Style – set of named or inherited attributes

Attribute – associated value

StyleVisitor – for iterating over style

Callbacks

```
interface Action {
   void execute();
   boolean reversible();
   void unexecute();
};
```

Notification

```
//
// These are “mixin” types.
//
interface Observable {
   Action attach(in Observer o);
   void notify();
};
interface Observer {
   void disconnect();
   void update();
};
```

Style interface

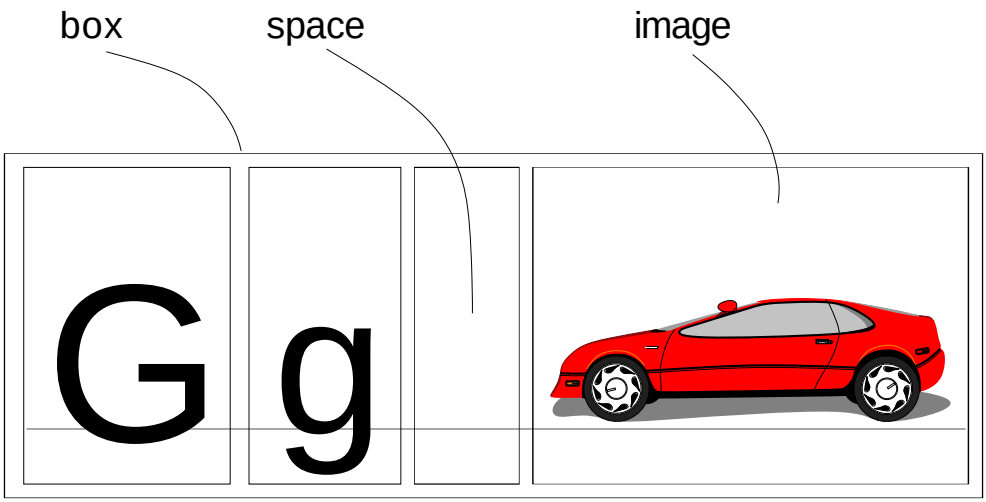
```
interface Style : Observable {
   Style clone();
   Style new_style();
   Style parent();
   void link(in Style parent);
   void unlink();
   void alias(in String s);

   Attribute bind(in String name);
   Attribute resolve(in String name);
   void unbind(in String name);

   void visit_aliases(in StyleVisitor v);
   void visit_attributes(in StyleVisitor v);
   void visit_children(in StyleVisitor v);

   void lock();
   void unlock();
};
```

Attribute interface

```
interface Attribute : Observable {
   String name();
   attribute long priority;

   boolean read_boolean(out boolean b);
   void write_boolean(in boolean b);

   boolean read_string(out String s);
   void write_string(in String s);

   boolean read_coord(out Coord c);
   void write_coord(in Coord c);

   boolean read_value(out any v);
   void write_value(in any v);

   void lock();
   void unlock();
};
```

Glyphs – objects that draw

Think of a glyph as a character



Glyph interface

```
interface Glyph {
   struct Requirement {
      boolean sizes_defined;
      Coord natural, maximum, minimum;
      Alignment alignment;
   };
   struct Requisition {
      Requirement x, y, z;
      boolean preserve_aspect;
   };

   Glyph clone();
   attribute Style style;
   Transform transform();

   void request(inout Glyph::Requisition r);
   void bounds(in GlyphTrail, in Region r);
```

Glyph interface (cont'd.)

```
    void traverse(in GlyphTraversal t);
    void draw(in GlyphTraversal t);
    void pick(in PickTraversal t);
    void undraw(in GlyphTraversal t);

    boolean is_root();
    boolean is_leaf();
    attribute Glyph body;
    void append(in Glyph g);
    void prepend(in Glyph g);
    GlyphOffset first_child();
    GlyphOffset last_child();
    GlyphOffset first_parent();
    GlyphOffset last_parent();

    void need_to_redraw(in Region r);
    void need_to_update();
};
```

Requisition – what a glyph wants

- *Requirement per axis*
- *Returned by Glyph::request*

Allocation – region a glyph gets

- *Span per axis*
- *Passed as part of GlyphTraversal*

Bounds – where a glyph draws

- *Computed by Glyph::bounds*
- *Useful for update, culling*

Making glyphs lightweight

Simple set of operations

No required base class storage

Pass context during traversal

Glyphs may be shared

Glyphs may be heavyweight

Screen update

Some glyphs remember their location

These objects can damage the window

Root object is drawn to repair damage

Output is clipped to damage

Some objects check if they are visible

If not then avoid drawing children

Advantages

- *Objects can share area*
  - *e.g., a label on top of an image*
- *Output may be double-buffered*
  - *Supports smooth interaction, animation*
  - *InterViews uses X pixmaps*
- *Objects control granularity of update*
  - *Cheap objects just draw*

Identifying a glyph instance

GlyphOffset – position within aggregate

GlyphTrail – path traversed

GlyphOffset

```
interface GlyphOffset {
   Glyph parent();
   Glyph child();
   GlyphOffset next();
   GlyphOffset prev();
   void insert(in Glyph g);
   void replace(in Glyph g);
   void remove();
   void notify();
};
```

GlyphTrail

```
interface GlyphTrail {
   GlyphTrail clone();
   Viewer head();
   Glyph end();
   Glyph current();
   void forward();
   void backward();

   Region allocation();
   Region bounds();
   Transform transform();
};
```

Traversal state

```
interface GlyphTraversal {
   void begin_trail(in Viewer v);
   GlyphTrail trail();
   void end_trail();

   void visit(in GlyphOffset o, in Region allocation);

   attribute Painter painter;
   Region allocation();
   Region damage();
};
```

Viewers – user interaction

```
interface Viewer : Glyph, Observer {
   Frame frame();
   Viewer parent();
   ViewerStyle viewer_style();

   void locate(in Region r, in ViewerHit h);
   InputFocus request_focus(in Viewer v);
   InputFocus request_temporary_focus(in Viewer v);
   InputFocus receive_focus(in boolean leaf);
   InputFocus lose_focus();
   InputFocus lose_temporary_focus();
   boolean first_focus();
   boolean last_focus();
   boolean next_focus();
   boolean prev_focus();

   void append_viewer(in Viewer v);
   void prepend_viewer(in Viewer v);
};
```

Viewers often edit data objects

- *Widgets – buttons, menus, scrollbars*
- *Text*
- *Drawing*
- *Image*
- *Builder*
- *Movies*

Interoperability with other toolkits

- *Viewers might be implemented with X subwindows, use Xlib drawing*
- *Partial functionality – embedding without full graphics support*
- *Gateway objects to/from Xt*

Graphical embedding summary

- *Unify structured graphics and embedding*
- *Glyphs*
  - *Graphics, layout, lightweight,*
  - *Shareable, damage-based update*
- *Viewers*
  - *Input handling and focus management*
  - *Menubar, palette sharing*
- *Cross address space embedding “for free”*

Interfaces

Action
Attribute
Frame
Glyph
GlyphOffset
GlyphTrail
GlyphTraversal
InputFocus
Inset
Observable
Observer
PickTraversal
Style
StyleVisitor
TranslationTable
Viewer
ViewerHit

Fresco summary

Developing new standard X API

Cover Xlib and Xt functionality

Use standard object system

Provide resolution independence

Add structured graphics and embedding

Migrate existing widgets

Provide better end-user environment