

---

# HotDraw

A Framework for 2-D graphics

# HotDraw

---

Original done at Tektronix by Kent Beck and  
Ward Cunningham

Drawing tool

Oscilloscope user interface prototyper

Diagramming inspector

Electronic schematic editor.

PERT chart editor.

# Components

---

Drawing -- a collection of figures

Figure -- a visible component of a drawing

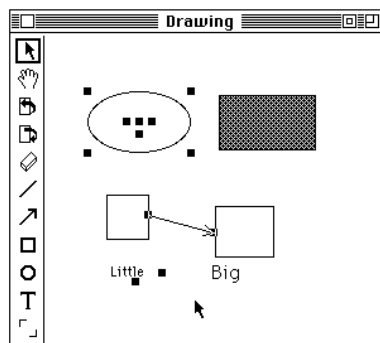
Tool -- a user interface “mode”

Handle -- a small box that controls an aspect of a figure

DrawingEditor -- the whole system

# HotDraw example

---



## Drawing

---

Drawing keeps track of figures.

Methods for managing figures

*add:, remove:, bringToFront:,  
sendToBack:*

*displayOn:*

Methods for animating (default does  
nothing) *step, wait*

## Figure

---

Can display itself

Knows its location on the screen

Can move and change its size

Knows the drawings and other figures that  
depend on it and notifies them when it  
changes

Can create handles for itself

## Figure (cont)

---

Figure is a `VisualComponent`, but it keeps track of its own location.

So, implement *bounds* and *displayOn:*.

Figure provides *containsPoint:*  
*containedBy:*, *center*, *right*, *bottom*, etc.

## EllipseFigure

---

A new Figure class must implement **displayFigureOn:**, which is called by **displayOn:**

Both methods take a `GraphicsContext` as an argument.

## **EllipseFigure>>displayOn: aGC**

self isOpaque

ifTrue: [self displayFilledOn:  
          aGraphicsContext].

self lineWidth > 0

ifTrue: [self displayOutlineOn:  
          aGraphicsContext]

## **displayFilledOn: aGraphicsContext**

aGraphicsContext paint: self fillColor.

aGraphicsContext

displayWedgeBoundedBy: self bounds

startAngle: self startAngle

sweepAngle: self sweepAngle

---

**displayOutlineOn: aGraphicsContext**

aGraphicsContext lineWidth: self  
lineWidth.

aGraphicsContext paint: self lineColor.

aGraphicsContext

displayArcBoundedBy: self bounds

startAngle: self startAngle

sweepAngle: self sweepAngle

## Figure class hierarchy

---

Figure

EllipseFigure

CachedFigure

ImageFigure

CompositeFigure

PolylineFigure

TextFigure

ViewAdapterFigure

## *PolylineFigure*

---

Figure made up of line segments

BezierFigure

LineFigure

SplineFigure

## Figure as an abstract class

---

Concrete methods

- keep track of drawing

Abstract methods

- display
- knows bounding box

Template methods

- tell whether point is inside it

# Tool

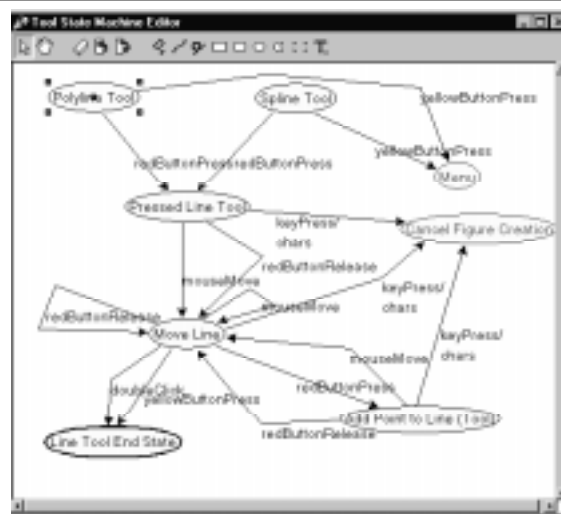
Tool has an icon (for palette) and a cursor (for canvas).

Tool acts like a part of the controller. The drawing controller delegates all operations to the current tool, so changing the current tool changes the behavior of the editor.

Tool is a state for the controller, and has a set of states that it can be in.

Tool is a state machine.

# StateMachine editor



## Tools and the State pattern

---

One Tool class

Action defined with a Smalltalk block

Data stored in a dictionary so subclasses are not needed

State transitions stored in a dictionary so it is easy to change.

Tool definitions stored in class methods of Tool.

## Handle

---

A handle is a figure that looks like a little box. It is attached to a figure and does something to the figure when you press it.

Each Figure has a "handles" method that creates a set of handles for it.

## Handles

---

TrackHandle -- executes block while tracking

IndexedTrackHandle

TentativePositionHandle

## HotDraw and MVC

---

HotDraw hides most of MVC

DrawingEditor, DrawingController

Drawing is a View!

HotDraw programmer needs to know about DrawingEditor and VisualComponent protocol, but not about controllers.

# DrawingEditor

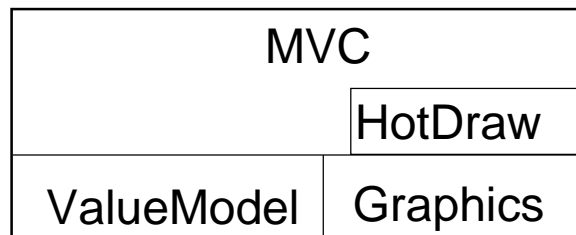
---

The DrawingEditor knows the tools on the palette and the current tool.

DrawingView gets Drawing from DrawingEditor.

# HotDraw and MVC

---



## Domain Knowledge

---

A framework describes how to build a particular kind of program.

- HotDraw embodies a theory of interactive graphics
- HotDraw contains knowledge of graphics tricks

## Domain Knowledge

---

Developers of framework have to understand application domain.

Framework can hide lots of information from its users.

Frameworks make programmers more specialized.

## HotDraw Summary

---

Building an application consists primarily of reusing existing classes as well as creating a few new classes .

Framework reuses not only code, but also design and analysis.

## Black-box Frameworks

---

Replacing reuse by inheritance with reuse by object composition makes it easier to make programs without defining new classes.

Ultimate framework lets you plug objects together with visual language.

## History of HotDraw

---

V1 Tools used the State pattern, there was a subclass of Tool for each kind of tool.

V2 Tools became a table of Commands. Tools became event driven. Figures used a constraint system.

V3 Figures became as much like VisualComponents as possible. Tools became state machines, constraints and commands were eliminated.

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

537

## Black-box Frameworks have

---

- more kinds of objects
- more artificial kinds of objects
- more complex relationships between objects
- more objects

Not-quite-ultimate framework forces you to debug more complex system.

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

538

## Life-cycle

---

Reusable code requires many iterations.

Johnson's law of software engineering  
If it hasn't been tested, it doesn't work.

Corollary: software that hasn't been reused is not reusable.

## Hard Facts of Reuse

---

Using reusable code is easy - writing reusable code is hard.

Software must be tested for its reusableness.

Fixing reusability bugs may require major changes.

Good design requires domain knowledge.

## Origin of Frameworks

---

Frameworks are a generalization of a subsystem or application.

Abstract classes are generalizations of concrete subclasses.

Frameworks evolve.

Interfaces should minimize changes.

Framework is often "refactored" as it evolves.

## Framework is:

---

- reuse of design of subsystem or entire application
- tells how to break problem into objects
- consists of classes (usually abstract) and the way they interact
- not just static interface, but invariants and abstract algorithms
- reuse of collaborative model

## Frameworks

---

- made up of a set of interlocking design patterns
- evolve over time, usually becoming more black-box
- are hard to design, but give the big payoff of OOP