GUI Builders & No-Code

CS 695 / SWE 699: Programming Tools
Fall 2023
Today

- Part 1 (Lecture) (~60 mins)
- 10 min break!
- Part 2: Tech Talk - Microsoft Power Automate (15 mins)
- Part 3: (In-Class Activity) (60 mins)
Logistics

- HW 2 due today
- No class next week
- HW 3 due in 3 weeks
Overview

• Visual Programming

• GUI Builders
Visual Programming
Definitions

“Visual Programming Language”

“Any system where the user writes a program using two or more dimensions”
[Myers, 1990]

“A visual language manipulates visual information or supports visual interaction, or allows programming with visual expressions”
[Golin, 1990]

“A programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually”.

“A set of spatial arrangements of text-graphic symbols with a semantic interpretation that is used in carrying out communication actions in the world”.
[Lakin, 1989]
Motivation from Psychology

Language determines thought and that linguistic categories limit and determine cognitive categories [1]

In longer sentences meaning of each word may be clear, but the way in which they are strung together makes little sense imposes a tremendous mental workload to understand. [2]

Most design tasks require 3 cognitive skills: search, recognition and inference.
Diverse set of views (and studies) exist today about whether VPLs aid in search or cognition. [3]
Motivation

Some applications are (believed to be) very well suited to graphical development approaches

Scientific visualization
Simulations
User Interfaces
Signal Processing
Data Displays
(Claimed) Advantages of VPLs

• Fewer programming concepts
• Concreteness
• Explicit depiction of relationships
• Immediate visual feedback
• Parallel computation is a natural consequence of many visual programming paradigms
(Claimed) Disadvantages of VPLs

"Deutsch Limit" *

The problem with visual programming is that you can't have more than 50 visual primitives on the screen at the same time.

Some situations in which text has superiority:

- Documentation,
- Naming to distinguish between elements that are of the same kind, and
- Expressing well-known and compact concepts that are inherently textual, e.g. algebraic formulas.
Visual Programming Techniques

• Concreteness: expressing some aspect of a program using instances
  • e.g., display the effects of computation on individual instance

• Directness: small distance between goal and actions required of the user to achieve goal
  • e.g., direct manipulation of object properties

• Explicitness: don’t require inference to understand semantics
  • e.g., depict dataflow edges between variables

• Liveness: offer automatic display of effects of program edits on output
  • e.g., after every edit, IDE reruns code and regenerates output
History of VPLs

1960
- AMBIT/G/L
- Grail
- GAL
- Graphical Program Editor
- Query by Example
- Pygmalion
- I/O Pairs

1980
- Action Graphics
- FORMAL
- ThingLab
- Hi-Visual
- LabView
- PROGRAPH
- PIGS
- Pict
- Rehearsal
- SmallStar

1990
- Cube
- Cantata
- SchemePaint
- PROGRAPH
- AVS
- Montarian
- ChemTrains
- CODE 2.0
- Iconcode
- MViews
- Miro
- HI
- Visual
- LabView
- SPIGS
- Pict
- Rehearsal
- SmallStar

2000
- LOFI/HIP/1
- FOXQ
- VMQL
- GXL
- Euler View
- Yahoo Pipes
- Popfly

Techniques/Goals
- Graphs
- Flowcharts
- Flowchart derivatives
- FORMS
- Demonstrational
- 3D Rendering
- Visual Hierarchy
- Procedures
- Control Structures
- Programmable Graphics
- Animations
- Video Imagery Exploitation
- General purpose, declarative language
- Audio, video and image processing
- Graphical models from behavioral models
- Learning and Cognitive abilities in vision processes
- Handling Scalability, typing, and imperative design
- Collaborative Software Development
- Child Learning
- Xquery by FORMS
- Spreadsheet Analysis
- Visual Model Query
- Layouts
- Specification and Interchange
- Mashups
- Web-based design
- Programming for end-users (2003) / non-Professionals
History of VPLs

- **1960**: Technology Trigger
- **1980**: Period of Early promises
- **1990**: Period of Inflated Expectations
- **2000**: Period of Reality Check

**Resources**

- [Ellis, 1969]: GRAIL
- [Smith, 1975]: Pygmalion
- [Myers, 1990]: Taxonomies for VPL
- [Repenning, 1992]: Agent Sheet
- [Burnett, 1994]: Broad Classifications for VPL Research
- [Kirsten N. Whitley, 1997]: User Studies (for/against VPLs)
- [MacLaurin, 2009]: KODU

**Visual Programming**

- About 18,200 results (0.47 seconds)
- Timeline

**Search Settings**

- 1960-2011 Search other dates
History of VPLs

- Support the cognition aspect of Programming
- Strive for improvements in programming language design
- Let users program in Visual Languages
- (Almost) Make textual languages redundant
- Make programming more accessible
- Support domain-specific designs

# Taxonomy of visual programming languages

<table>
<thead>
<tr>
<th>Specification Technique</th>
<th>Systems:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Textual Languages:</td>
<td>Pascal, Ada, Fortran, Lisp, Ada, etc.</td>
</tr>
<tr>
<td></td>
<td>Tinker, Smallstar</td>
</tr>
<tr>
<td>Flowcharts:</td>
<td>Grail, Pict, FPL, IBGE, OPAL</td>
</tr>
<tr>
<td>Flowchart derivatives:</td>
<td>GAL, PIGS, SchemaCode, PLAY</td>
</tr>
<tr>
<td>Petri nets:</td>
<td>MOPS-2, VERDI</td>
</tr>
<tr>
<td>Data flow graphs:</td>
<td>Graphical Program Editor, PROGRAPH.</td>
</tr>
<tr>
<td></td>
<td>Graphical Thinglab, Music System, HI-VISUAL,</td>
</tr>
<tr>
<td></td>
<td>LabVIEW, Fabrik, InterCONS</td>
</tr>
<tr>
<td>Directed graphs:</td>
<td>AMBIT/G/L, State Transition UIMS, Bauer’s Traces</td>
</tr>
<tr>
<td>Graph derivatives:</td>
<td>HiGraphs, Miro, StateMaster</td>
</tr>
<tr>
<td>Matrices:</td>
<td>ALEX, MPL</td>
</tr>
<tr>
<td>Jigsaw puzzle pieces:</td>
<td>Proc-BLOX</td>
</tr>
<tr>
<td>Forms:</td>
<td>Query by Example, FORMAL</td>
</tr>
<tr>
<td>Iconic Sentences:</td>
<td>SIL-ICON</td>
</tr>
<tr>
<td>Demonstrational*:</td>
<td>Pygmalion, Rehearsal World, Peridot</td>
</tr>
<tr>
<td>None*:</td>
<td>I/O Pairs, Editing by Example</td>
</tr>
</tbody>
</table>

Dataflow Program Representations

• Represent computation as a network
• Nodes correspond to components
• Edges correspond to data flow between components
LaToza

GMU CS 695 / SWE 699 Fall 2023

Figure 2a. Bidirectional diagram using two Slider controls to achieve a Fahrenheit-to-Centigrade converter.

Figure 2b. Internal diagram for the F/C component used in the diagram at left.

Figure 3. A Fabrik diagram computes the image for the slider in figure 2. The Mouse component simulates the slider image to support input as well as output.

Figure 4a. A simple scrollbar diagram. Logic is provided for rotating when the image is wider than high.

Figure 4b. Demonstrating the rotation logic. All display and mouse-tracking operations are properly scaled and rotated.

Figure 5a. The Draw component automatically lays out diagrams as the user creates a drawing.

Figure 5b. By editing the generative diagram, the top-left of the oval is tied to the bottom-right of the rectangle.

Figure 6. Streaming gateways provide iterative capability. Here numbers are converted to rectangles, resulting in a simple bar chart.
Pure Data (1996)
Quartz Composer (2005)
NoFlo (2014)
BLUEPRINTS
GETTING STARTED
Fastgen in 5 minutes

David
Founder
Form Representations

- Program consists of a form, with a network of interconnected cells
- Developers define cell through combination of pointing, typing, gesturing
- Cells may define constraints describing relationships between cells
Forms/3

- Based on constraints between cells
- Supports graphics, animation, recursion
- Concreteness: resulting box is immediately seen
- Directness: demonstrates elements directly
- Level 4 liveness: immediate visual feedback

Margaret M. Burnett, "Visual Programming" In the Encyclopedia of Electrical and Electronics Engineering (John G. Webster, ed.), 1999
Forms/3 Example

http://web.engr.oregonstate.edu/~burnett/Forms3/LED.html
Forms/3 Example

99 bottles of beer on the wall,
bottles of beer...
Take one down, pass it around,
bottles of beer on the wall.

99 fby {{earlier bottles} - 1}
until {{earlier bottles} = 2}

99 bottles of beer on the wall.
99 bottles of beer...
Take one down, pass it around,
98 bottles of beer on the wall.

compose bottles at (4 2)
with fixedWords at (5 2)
with bottles at (4 14)
with {bottles - 1} at (4 38)

by Dr. Margaret M. Burnett and Jonathan Jay Cadiz
Discussion

• Given potential advantages, why isn't all programming now done in a visual programming language?
Cognitive Dimensions of Notations

- Analytical technique for assessing usability of notation through a set of heuristics
- Also terminology for describing usability problems

<table>
<thead>
<tr>
<th>Abstraction gradient</th>
<th>What are the minimum and maximum levels of abstraction? Can fragments be encapsulated?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closeness of mapping</td>
<td>What ‘programming games’ need to be learned?</td>
</tr>
<tr>
<td>Consistency</td>
<td>When some of the language has been learnt, how much of the rest can be inferred?</td>
</tr>
<tr>
<td>Diffuseness</td>
<td>How many symbols or graphic entities are required to express a meaning?</td>
</tr>
<tr>
<td>Error-proneness</td>
<td>Does the design of the notation induce ‘careless mistakes’?</td>
</tr>
<tr>
<td>Hard mental operations</td>
<td>Are there places where the user needs to resort to fingers or penciled annotation to keep track of what's happening?</td>
</tr>
<tr>
<td>Hidden dependencies</td>
<td>Is every dependency overtly indicated in both directions? Is the indication perceptual or only symbolic?</td>
</tr>
<tr>
<td>Premature commitment</td>
<td>Do programmers have to make decisions before they have the information they need?</td>
</tr>
<tr>
<td>Progressive evaluation</td>
<td>Can a partially-complete program be executed to obtain feedback on “How am I doing”?</td>
</tr>
<tr>
<td>Role-expressiveness</td>
<td>Can the reader see how each component of a program relates to the whole?</td>
</tr>
<tr>
<td>Secondary notation</td>
<td>Can programmers use layout, color, or other cues to convey extra meaning, above and beyond the ‘official’ semantics of the language?</td>
</tr>
<tr>
<td>Viscosity</td>
<td>How much effort is required to perform a single change?</td>
</tr>
<tr>
<td>Visibility</td>
<td>Is every part of the code simultaneously visible (assuming a large enough display), or is it at least possible to compare any two parts side-by-side at will? If the code is dispersed, is it at least possible to know in what order to read it?</td>
</tr>
</tbody>
</table>

Diffuseness / Terseness

• How many symbols or graphic elements is required to express a meaning?

• Simple rocket simulation program
  • Basic: 22 LOC, 140 words (fits on screen)
  • LabView: 45 icons, 59 wires (fits on screen)
  • Prograph: 52 icons, 79 connectors, 11 screens

Error-proneness

• Does the design of the notation induce slips?

• Compared to textual language, VPLs
  • Do not need delimiters & separators
  • Fewer identifiers are needed, easier to reference
  • Constructs inserted automatically (e.g., loops)
Viscosity

- How much effort is required to make a simple change?
- Edit Rocket program to take account of air resistance
  - Basic: 63.3 s
  - LabView: 508.3 s
  - Prograph: 193.6 s
- VPLs required many wires to be rebuilt, layout to be tweaked
Visibility

- Is every (relevant) part of the code simultaneously visible?

- LabView does not show both branches of conditional at same time (!)

- Particular problem for nested conditionals

- Prograph has poor support for deep nesting of routines
VPLs Discussion

• Often offers a representation that makes specific tasks easy
  • e.g., tracking data flow
  • Often involves structured editor targeted to specific domain, which may not support full range of programs
• But may make other tasks harder
• Often limited focus on scalability

• May be possible to get benefits of task-specific representations without drawbacks through task specific editor rather than language
GUI Builders
GUI Builders

• User interfaces are visual

• Why not edit interfaces visually?

• Enables direct manipulation - drag and drop to create element, change properties to style, immediately see new feedback
HyperCard (1987)

https://www.loom.com/share/6c57bdb7e4d0488eb2f9f8949b028ef8?t=491
HyperCard (1987)

Fig. 23. An editor for the Positive Gravity Button. When the mouse goes up, Emile will execute 4 actions: Accelerated Motion 1, Stop Increasing 1, and Display a Value 1 (2 times). At the bottom of the screen, we can see the code that Emile will execute. Underlined text corresponds to parameters (or slots) that the user can fill in using menu options and dialog boxes.
Visual Basic (1991)
Elements of a GUI Builder
Challenges

• Web apps dynamically generate HTML based on data
  • Lists of elements, conditional behavior
• Keeping code and GUI builder in sync
  • What happens if code includes something that can't be represented in GUI builder properties (e.g., an animation)
  --> Need to abstract an immutable concrete value set in a property editor into a relationship that is dynamically computed from other variables
• Potential for ambiguity, where there's multiple ways to arrive at a specific concrete value
equiTriPt [x3, y3] [x2, y2] =  
[ (x3 + x2 + sqrt(3) * (y3 - y2))/2!, (y3 + y2 - sqrt(3) * (x3 - x2))/2!]

oneThirdPt [x, y] =  
[ x / 1.5! + x2 / 3!, y / 1.5! + y2 / 3!]

point = [69, 308]
point2 = [642, 301]

makeKochPts depth p1 p2 =  
let oneThirdP2 = oneThirdPt p1 point2 in  
let oneThirdP3 = oneThirdPt p2 point2 in  
let equiTriP2 = equiTriPt oneThirdP2 oneThirdP3 in  
if depth < 2 then  
  [pt1, oneThirdP2, equiTriP2]  
else  
  let makeKochPts2 = makeKochPts (depth - 1) pt1 oneThirdP2 in  
  let makeKochPts3 = makeKochPts (depth - 1) pt2 oneThirdP3 in  
  let makeKochPts4 = makeKochPts (depth - 1) equiTriP2 oneThirdP2 in  
  let makeKochPts5 = makeKochPts (depth - 1) oneThirdP2 equiTriP2 in  

depth = 3

makeKochPts depth point point2 =  
let oneThirdPt2 = oneThirdPt point point2 in  
let oneThirdPt3 = oneThirdPt point2 point in  
let equiTriPt2 = equiTriPt oneThirdPt2 oneThirdPt3 in  
if depth < 2 then  
  [point, oneThirdPt2, equiTriPt2]  
else  
  let makeKochPts2 = makeKochPts (depth - 1) point oneThirdPt2 in  
  let makeKochPts3 = makeKochPts (depth - 1) oneThirdPt3 equiTriPt2 in  
  let makeKochPts4 = makeKochPts (depth - 1) oneThirdPt2 equiTriPt2 in  
  let makeKochPts5 = makeKochPts (depth - 1) oneThirdPt3 equiTriPt2 in  

topPts = makeKochPts depth point point2
botPt = equiTriPt point point2
rightPts = makeKochPts depth point2 botPt
leftPts = makeKochPts depth botPt point

snowflakePts = concat [topPts, rightPts, leftPts]
	polygon1 =  
  let pts = snowflakePts in  
  let [color, strokeColor, strokeWidth] = [114, 360, 2] in  
  polygon color strokeColor strokeWidth pts

svg (concat [polygon1])
10 min break
Tech Talk: Microsoft Power Automate
In-Class Activity

• In groups of 2, try out Microsoft Power Automate
  • https://powerautomate.microsoft.com/en-us/#home-signup
  • Setup the free trial for the Web version of Power Automate
  • Read docs to understand how to build a Power Automate App.
  • Pick a data processing problem based on examples in docs
  • Build a Power Automate App and try it out.