Visual Programming Languages

SWE 795, Spring 2017

Software Engineering Environments

Adapted from slides by Vishal Dwivedi,
Human Aspects of Software Development, Spring 2011

Today

- Part 1 (Lecture)(~50 mins)
- Break!
- Part 2 (Discussion)(~30 mins)
    - HW3 presentations
- Part 3 (Discussion)(~60 mins)
    - Project work

Definitions

"Programming"

''The process of transforming a mental plan of desired actions for a computer into a representation that can be understood by the computer''

– Jean-Michel Hoc and Anh Nguyen-Xuan

"Single-dimensional characteristics"

The compilers or interpreters programs as long, one-dimensional streams.

Definitions

"Visual Programming"

"Programming in which more than one dimension is used to convey semantics." *- Myers, 1990*

"Token"

"A collection of one or more multi-dimensional objects".

Examples:

Multi-dimensional graphical objects
Spatial relationships
Use of the time dimension to specify "before-after" semantic relationships.

"Visual Expression"

"A collection of one or more tokens"

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".

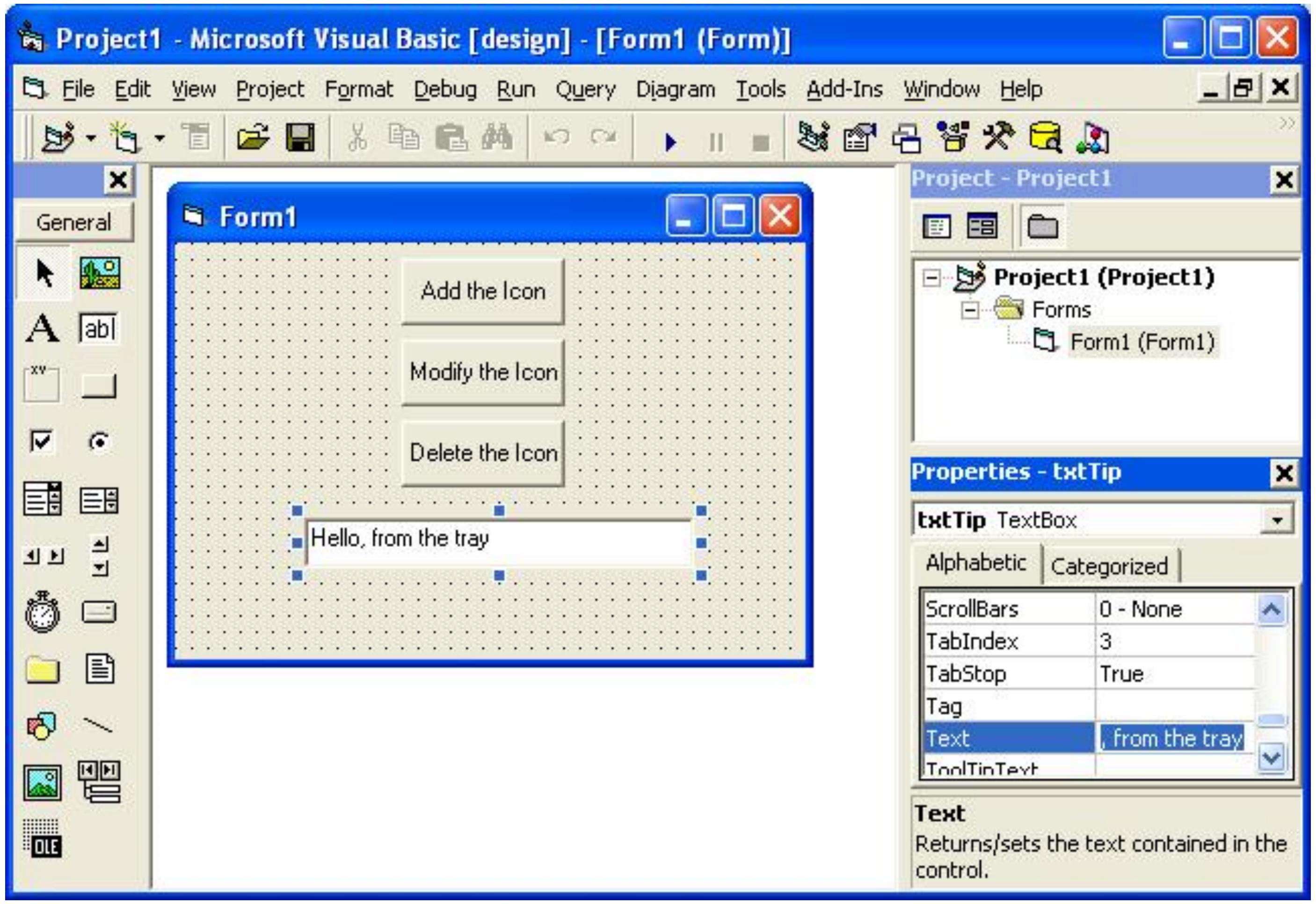
"A set of spatial arrangements of text-graphic symbols with a semantic interpretation that is used in carrying out communication actions in the world". [Lakin, 1989]

What is not a Visual Programming Language?

Programming Languages like Visual Basic, Visual C++, Visual C sharp, Delphi, etc do not satisfy the *multi-dimensional characterization.*

They are primarily Textual languages with:

- A graphical GUI builder
- A visual user interface



Goal of VPL Research

- To strive for improvements in programming language design.
- To make programming more accessible to some particular audience.
- To improve correctness with which people perform programming tasks.
- To improve the speed with which people perform programming tasks.

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]

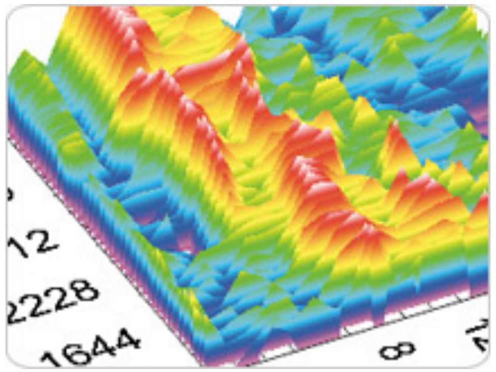
Most design tasks require 3 cognitive skills: search, recognition and inference.
Diverse set of views (and studies) exist today about whether VPLs aid in search or cognition. [3]

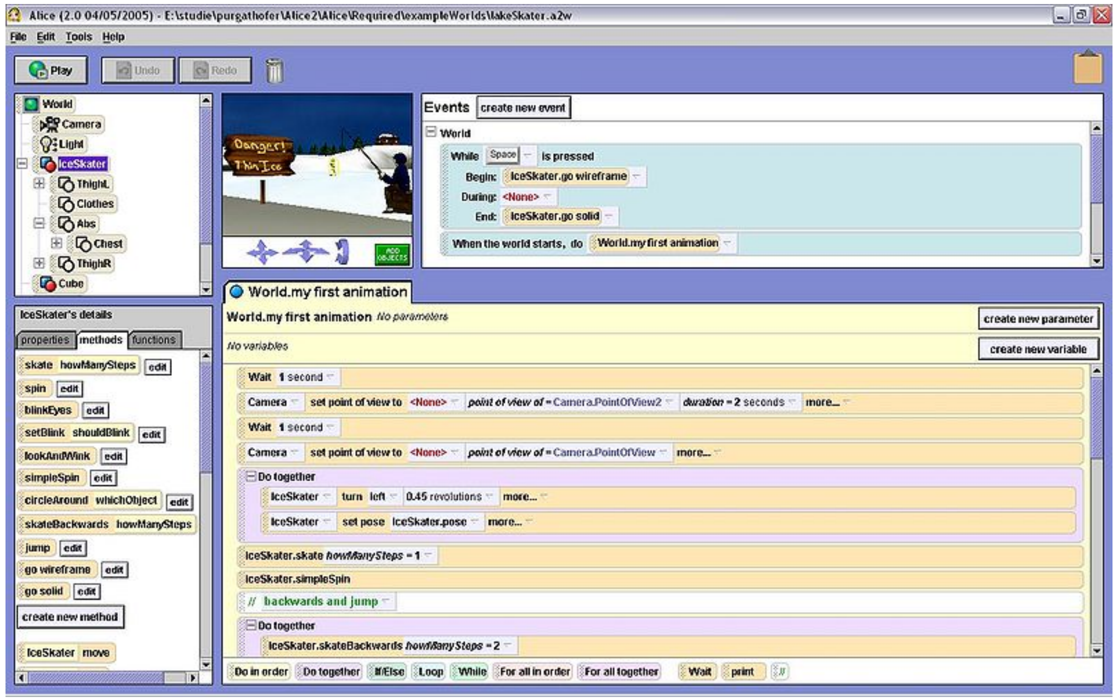
[1] Sapir, E. (1929): 'The Status of Linguistics as a Science'. In E. Sapir (1958): *Culture, Language and Personality* (ed. D. G. Mandelbaum). Berkeley, CA: University of California Press
[2] Christopher D. Wickens, "Engineering Psychology and Human Performance", 3rd Edition
[3] J. H. Larkin and H. A. Simon. Why a diagram is (sometimes) worth ten thousand words. Cognitive Science, 11:65-99, 1987.

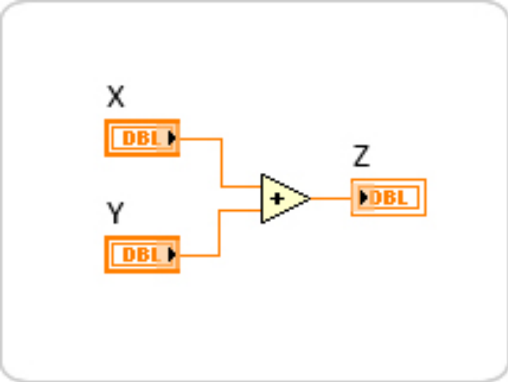
Motivation

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 Languages 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
- Livenesss: offer automatic display of effects of program edits on output
  - e.g., after every edit, IDE reruns code and regenerates output

Levels of liveness

- Level 1: No semantic feedback offered
  - e.g., using ER diagram for documentation
- Level 2: Semantic feedback, but not offered automatically
  - e.g., interpreters
- Level 3: Incremental semantic feedback automatically provided after edit, regenerating onscreen output
  - e.g., spreadsheets
- Level 4: Incremental semantic feedback offered after edits & systems events (e.g., clock ticks, mouse clicks)
  - e.g., some Smalltalk environments (?)

Tanimoto, S., VIVA: a visual language for image processing. Journal of Visual Languages Computing 2(2): 127-139, June 1990.

History of VPLs

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

- Action Graphics
- FORMAL
- ThingLab
- Hi-Visual
- LabView
- PROGRAPH
- PIGS
- Pict
- Rehearsal
- SmallStar
- Forms
- Editing by Example
- PICT
- Lotus 1-2-3
- SIL-ICON
- VisiCalc
- HiGraphs
- Miro
- StateMaster

- Cube
- Cantata
- SchemePaint
- CODE 2.0
- Iconicode
- MViews
- AVS
- Mondrian
- ChemTrains
- Vampire
- VIPR
- SPE

- LOFI/HIPI
- FOXQ
- VMQL
- GXL
- Euler View
- Yahoo Pipes
- Popfly

1960

Techniques
- Graphs
- Flowcharts
- Flowchart derivatives
- FORMS
- Demonstrational

1980

Techniques
- Graphs
- Flowcharts
- Flowchart derivatives
- FORMS
- Demonstrational
- Data Flows
- Spreadsheets
- Matrices
- Jigsaw Puzzles
- Petri nets
- Flowchart derivatives

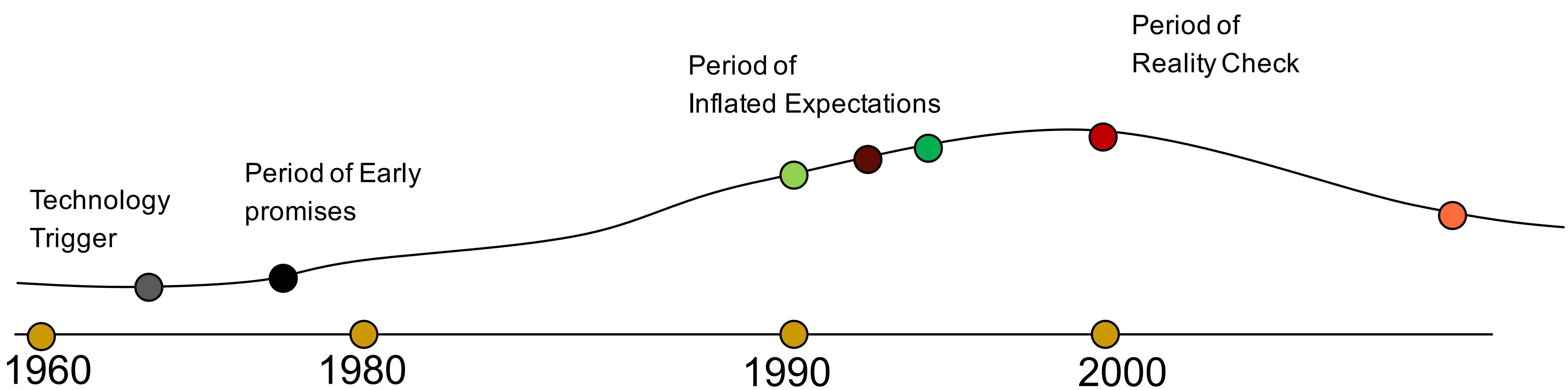
1990

Techniques/Goals
- 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

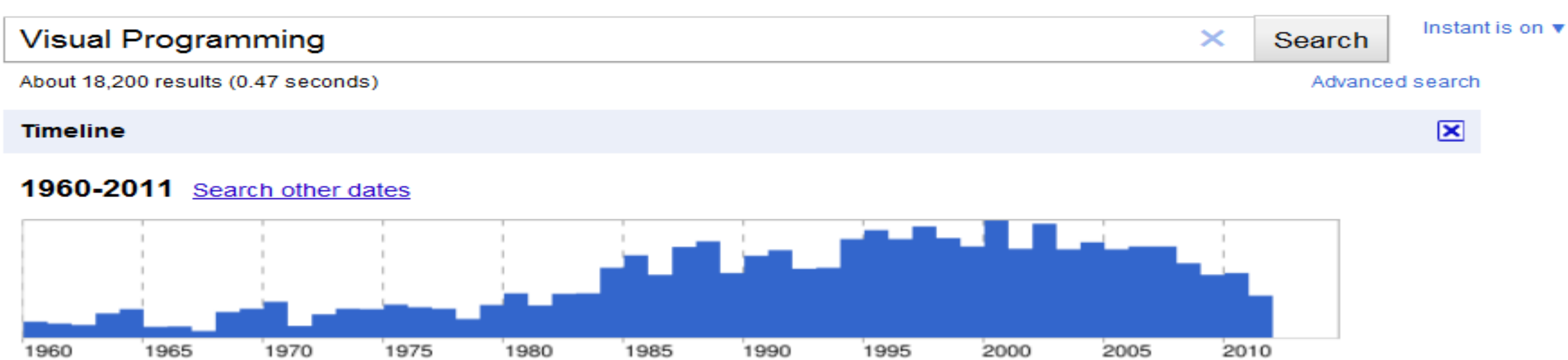
2000

Techniques/Goals
- 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

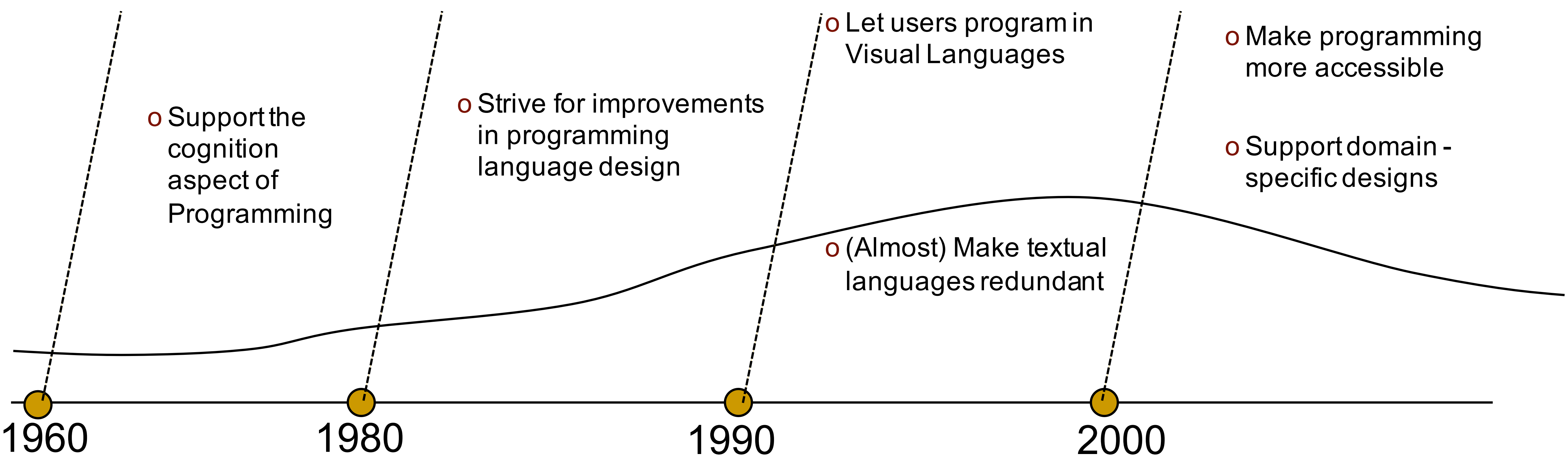
History of VPLs



- **[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



History of VPLs



Taxonomy of visual programming languages

Specification Technique:	Systems:
Textual Languages:	Pascal, Ada, Fortran, Lisp, Ada, etc. Tinker, Smallstar
Flowcharts:	Grail, Pict, FPL, IBGE, OPAL
Flowchart derivatives:	GAL, PIGS, SchemaCode, PLAY
Petri nets:	MOPS-2, VERDI
Data flow graphs:	Graphical Program Editor, PROGRAPH, Graphical Thinglab, Music System, HI-VISUAL, LabVIEW, Fabrik, InterCONS
Directed graphs:	AMBIT/G/L, State Transition UIMS, Bauer's Traces
Graph derivatives:	HiGraphs, Miro, StateMaster
Matrices:	ALEX, MPL
Jigsaw puzzle pieces:	Proc-BLOX
Forms:	Query by Example, FORMAL
Iconic Sentences:	SIL-ICON
Spreadsheets*:	VisiCalc, Lotus 1-2-3, Action Graphics, ''Forms''
Demonstrational*:	Pygmalion, Rehearsal World, Peridot
None*:	I/O Pairs, Editing by Example

Brad A. Myers. "Taxonomies of Visual Programming and Program Visualization," Journal of Visual Languages and Computing. vol. 1, no. 1. March, 1990. pp. 97-123.

Dataflow Program Representations

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

Prograph

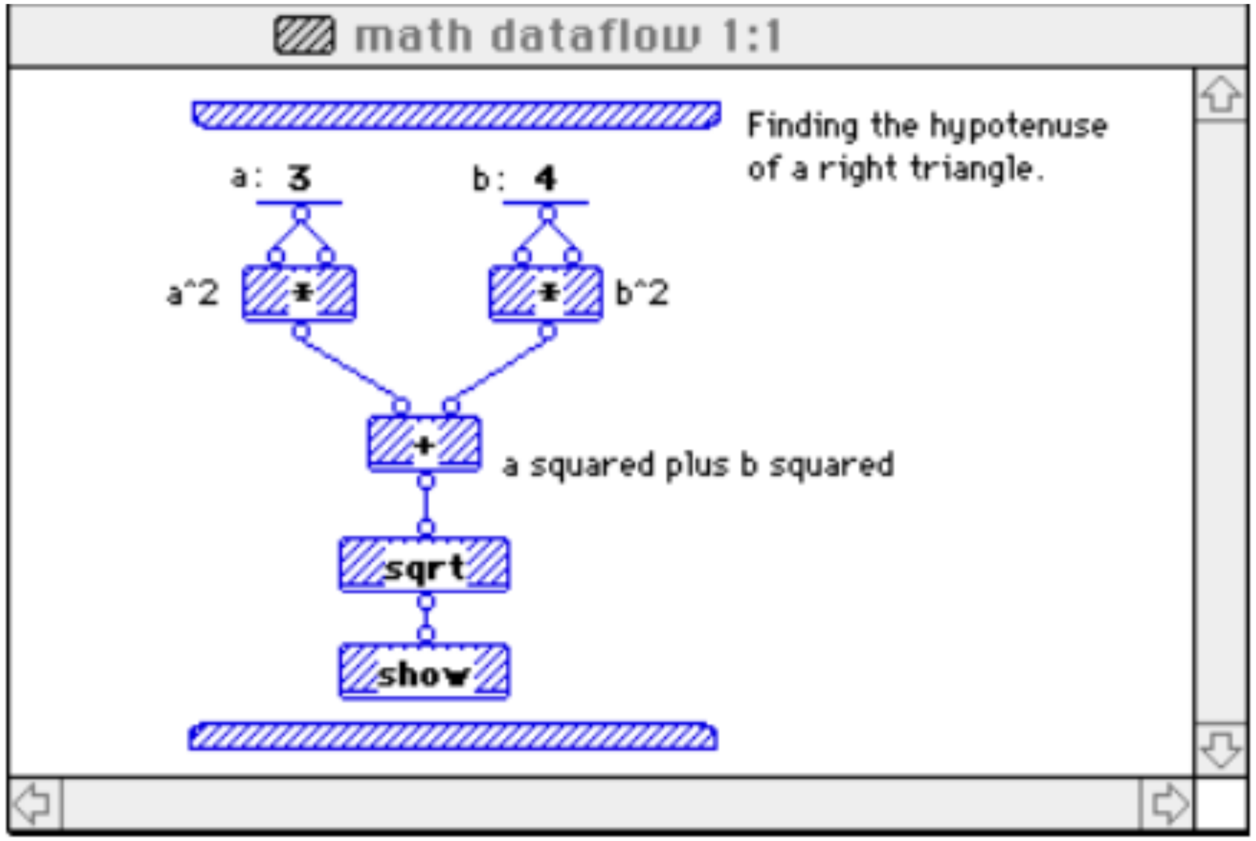
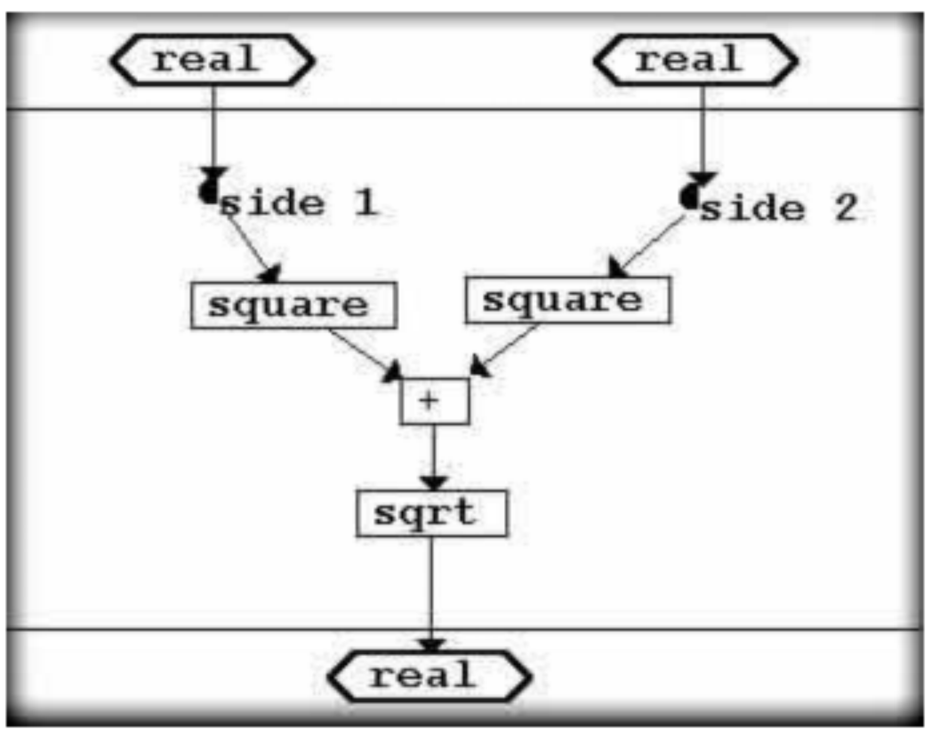


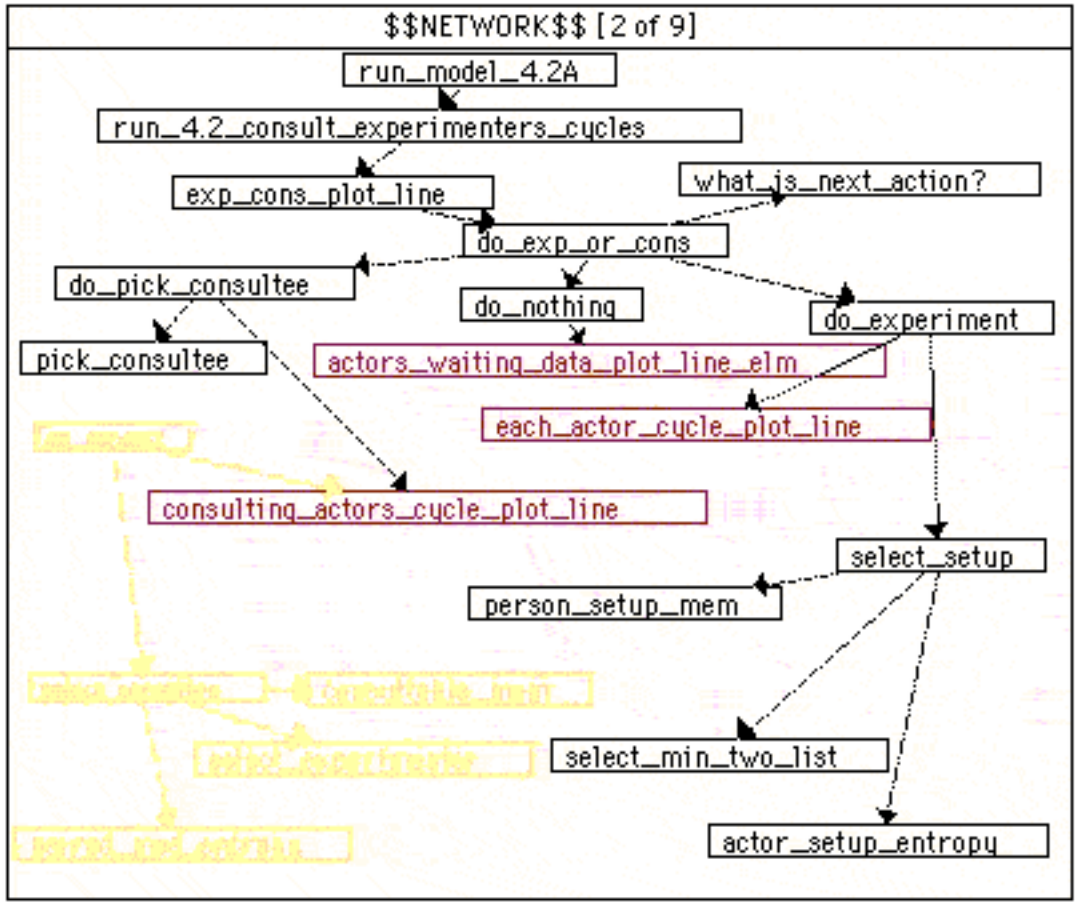
Figure 3: Dataflow programming in Prograph. Here the programmer is using the low-level (primitive) operations to find the hypotenuse of a right triangle. Prograph allows the programmer to name and compose such low-level graphs into higher-level graphs that can then be composed into even higher-level graphs, and so on.

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

Industrial Example: Clarity

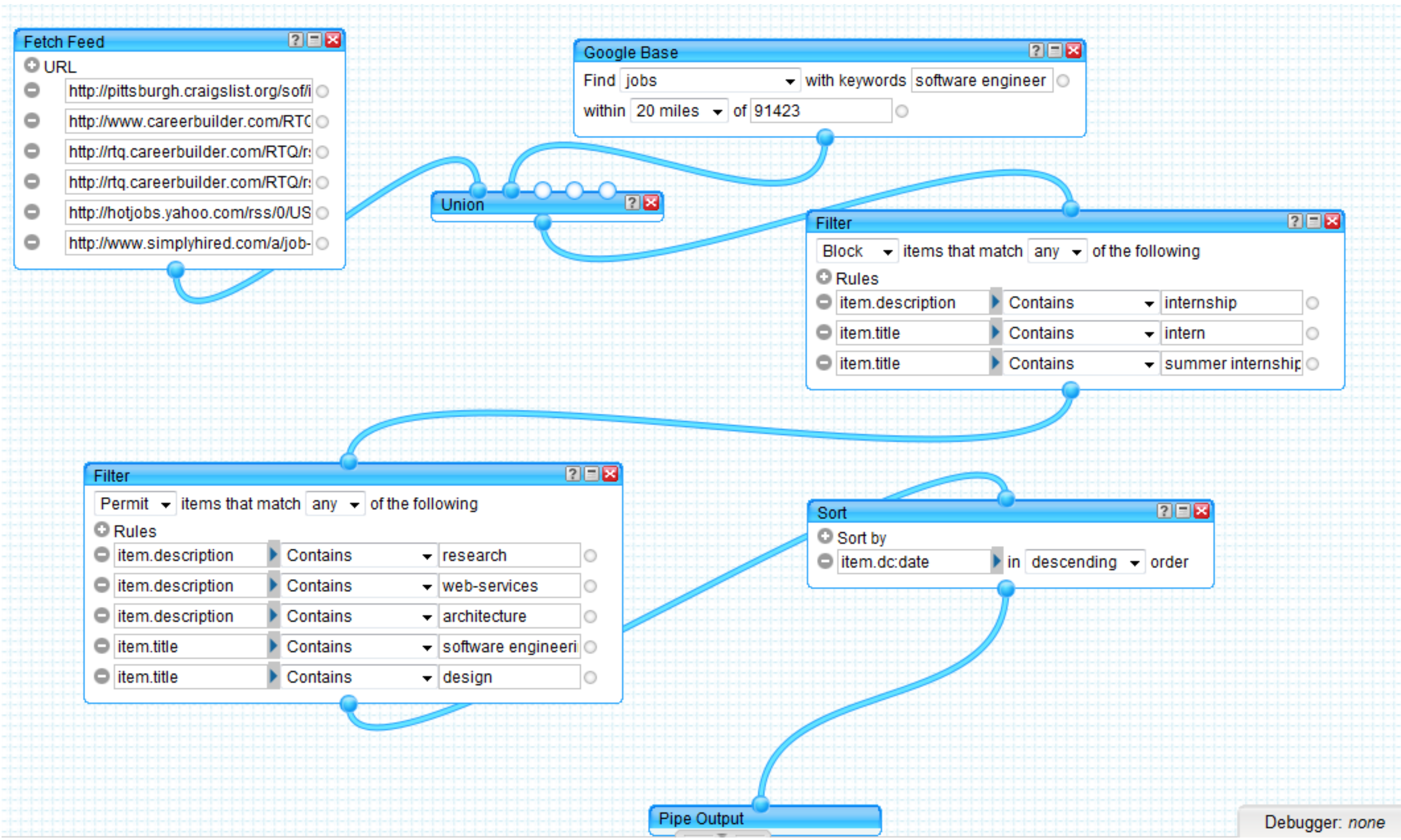
- “Clarity is a schematic functional programming environment that allows you to design and implement programs by drawing them. The picture below shows an example of the hypotenuse function that expresses Pythagoras' theorem.”





http://www.clarity-support.co.uk/products/clarity/

Industrial Example: Yahoo Pipes

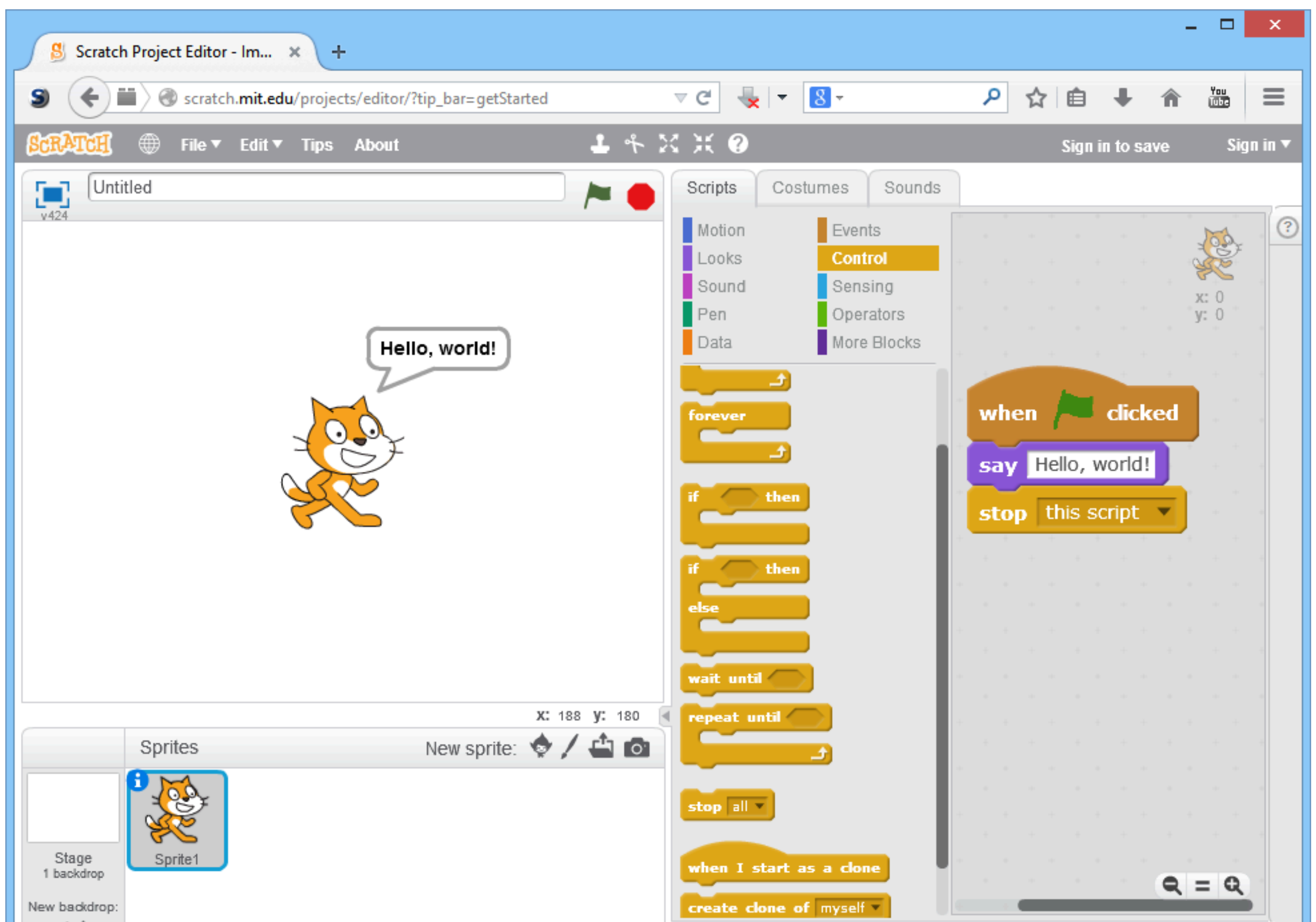


https://en.wikipedia.org/wiki/Yahoo!_Pipes

https://www.youtube.com/watch?v=Xv-4TOit5_g

Structured editors

- Structured editors that utilize extra dimension to capture program semantics can be considered visual programming languages
  - e.g., Alice, Scratch



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

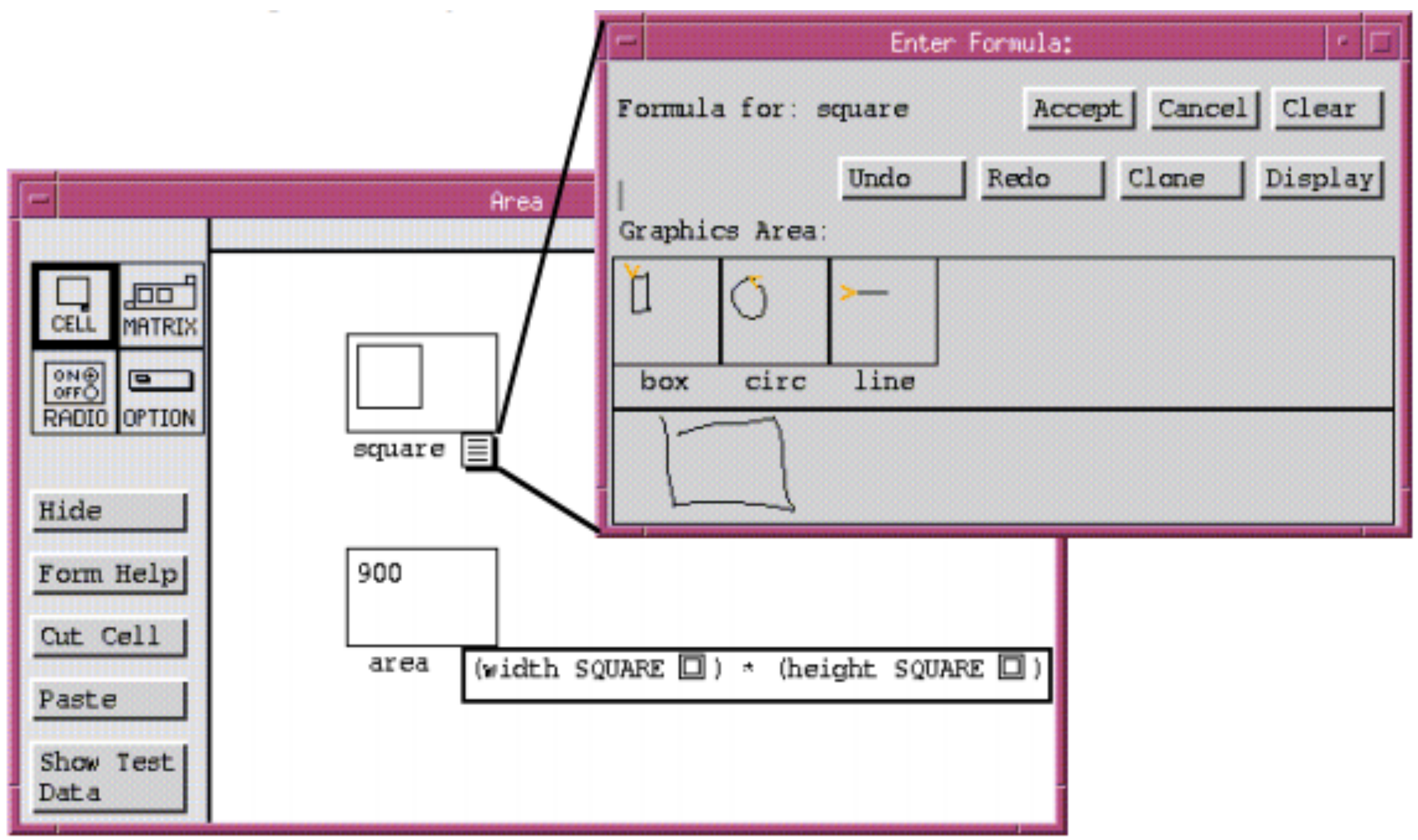
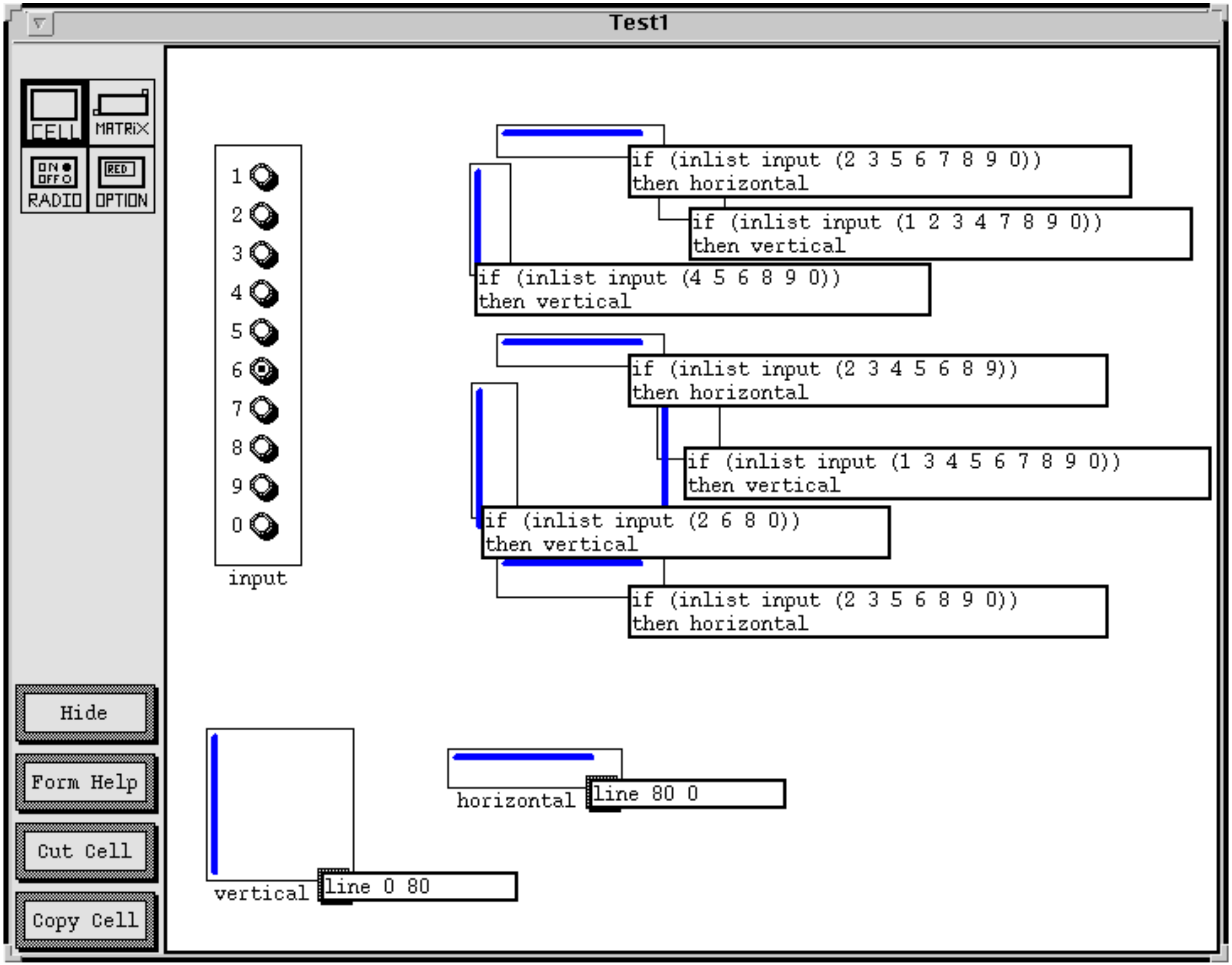


Figure 2: Defining the area of a square using spreadsheet-like cells and formulas in Forms/3. Graphical types are supported as first-class values, and the programmer can enter cell square's formula either by sketching a square box or by typing textual specifications (e.g., "box 30 30").

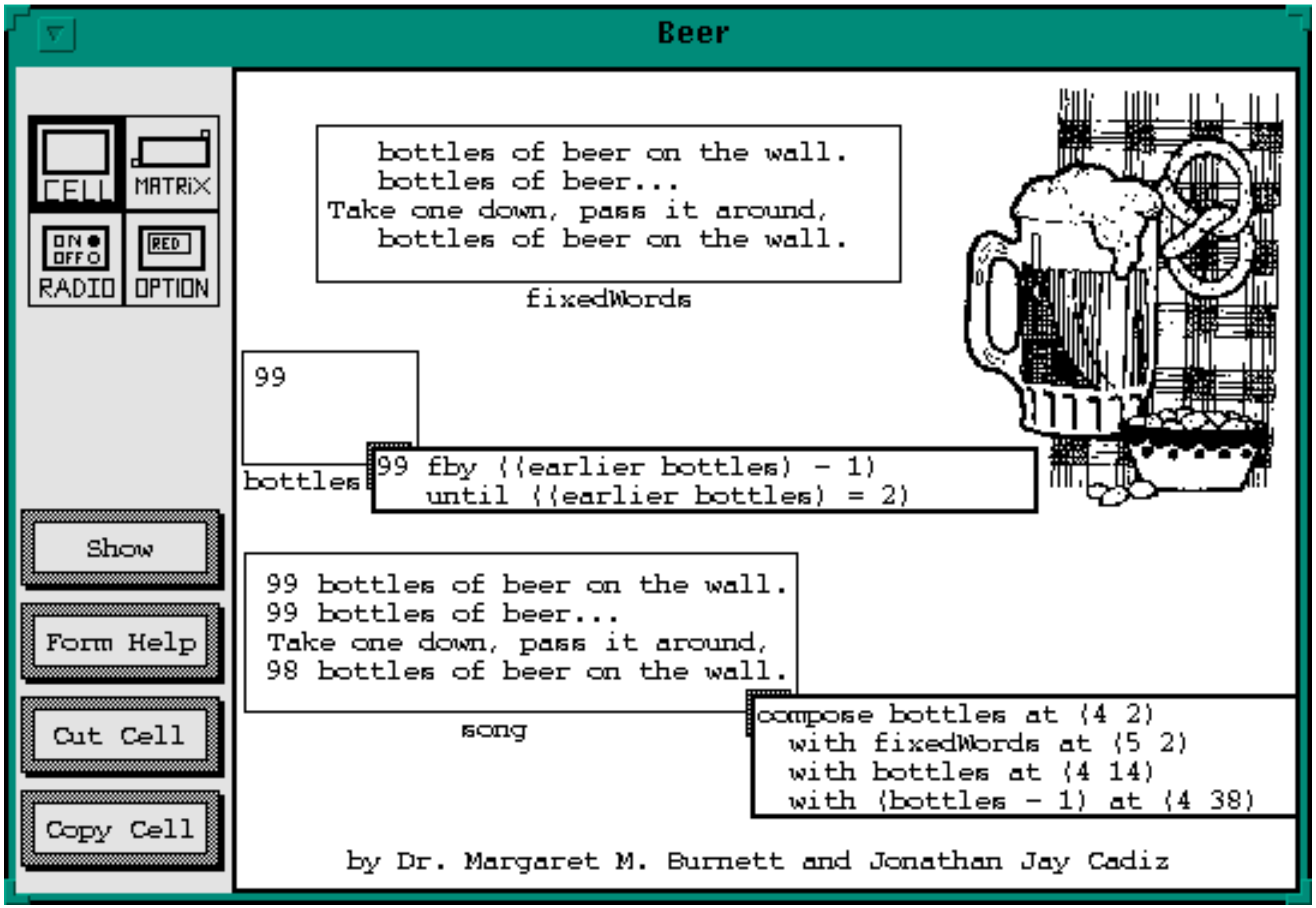
Margaret M. Burnett, "Visual Programming" In the Encyclopedia of Electrical and Electronics Engineering (John G. Webster, ed.), 1999

Forms/3 Example



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

Forms/3 Example



Interstate

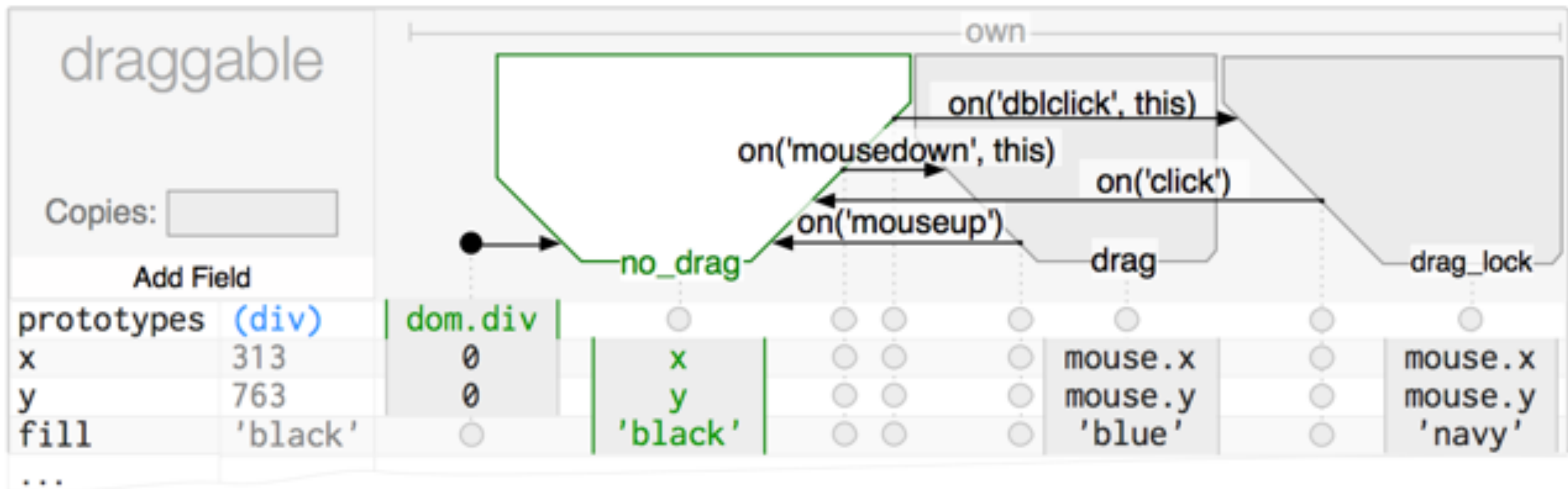


Figure 1: A basic InterState object, named `draggable`, which implements draggable and drag lock behaviors. Properties that control `draggable`'s display are represented as rows (e.g. `x`, `y`, and `fill`). States and transitions are represented as columns (e.g. `no_drag` and `drag`). An entry in a property's row for a particular state specifies a constraint that controls that property's value in that state. Here, while `draggable` is in the `drag` state, `x` and `y` will be constrained to `mouse.x` and `mouse.y` respectively, meaning `draggable` will follow the mouse.

http://interstate.from.so/

https://www.youtube.com/watch?v=M--9jsuDZis

Stephen Oney, Brad Myers, and Joel Brandt. 2014. InterState: a language and environment for expressing interface behavior. *Symposium on User interface software and technology*, 263-272.

Assessing Usability

- Empirical techniques assess usability through studies gathering data
- Analytical techniques use principles & guidelines to estimate the usability of a system

- Will look at a technique for analytical usability evaluation here

Cognitive Dimensions of Notations

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

Abstraction gradient	What are the minimum and maximum levels of abstraction? Can fragments be encapsulated?
Closeness of mapping	What 'programming games' need to be learned?
Consistency	When some of the language has been learnt, how much of the rest can be inferred?
Diffuseness	How many symbols or graphic entities are required to express a meaning?
Error-proneness	Does the design of the notation induce 'careless mistakes'?
Hard mental operations	Are there places where the user needs to resort to fingers or penciled annotation to keep track of what's happening?
Hidden dependencies	Is every dependency overtly indicated in both directions? Is the indication perceptual or only symbolic?
Premature commitment	Do programmers have to make decisions before they have the information they need?
Progressive evaluation	Can a partially-complete program be executed to obtain feedback on "How am I doing"?
Role-expressiveness	Can the reader see how each component of a program relates to the whole?
Secondary notation	Can programmers use layout, color, or other cues to convey extra meaning, above and beyond the 'official' semantics of the language?
Viscosity	How much effort is required to perform a single change?
Visibility	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?

T. Green and M. Petre, Usability analysis of visual programming environments: a 'cognitive dimensions' framework. Journal of Visual Languages and Computing 7(2): 131-174, June 1996

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

T. Green and M. Petre, Usability analysis of visual programming environments: a 'cognitive dimensions' framework. Journal of Visual Languages and Computing 7(2): 131-174, June 1996

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