# Visual Languages/Notations

- Aims of this section
  - Introduce use of diagrammatic/visual approaches to programming
  - Lok at several example visual languages
    - Chimera (programming by demonstration)
    - Forms/3 (spreadsheet-based)
    - Prograph (OO visual dataflow)
    - Kidsim (visual rule based)
    - UML (covered in more detail later)
  - Introduce approaches for evaluating visual notations and environments
    - Cognitive Dimensions
    - Attention Investment
    - Champagne Prototyping
- Tomorrow
  - Domain specific visual languages
- Later
  - UML
  - Pounamu meta tool for constructing VL editors



# Resources

- Much material in thise lecture from:
  - "Visual Programming," Margaret Burnett, in Encyclopedia of Electrical and Electronics Engineering (John G. Webster, ed.), John Wiley & Sons Inc., New York, 1999
  - "Scaling Up Visual Programming Languages", Margaret Burnett, Marla Baker, Carisa Bohus, Paul Carlson, Sherry Yang, Pieter van Zee, Computer, March 1995, 45-54.
  - Cognitive Dimensions website http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/



# What is a Visual Language?

- "some visual representations (in addition to or in place of words and numbers) to accomplish what would otherwise have to be written in a traditional one-dimensional programming language"
  - Shu, N *Visual Programming*, Van Nostrand Reinhold, NY, 1988
- Visual programming is programming in which more than one dimension is used to convey semantics. Eg:
  - multi-dimensional objects
  - use of spatial relationships
  - use of the time dimension to specify "before-after" semantic relationships.
- A Visual Programming Environment allows visual specification and generation of code
- NB some use of 2-D in conventional PLs
  - use of indentation/layout to convey semantic info



# History

- Early work didn't scale
  - Executable flowcharts
  - Programming by demonstration
- Followed by work in
  - Programming environments that replaced some textual programming by visual (eg VisualWorks, Visual Basic)
    - Won't consider here
  - CASE tools – programming in the large
  - General purpose VLs – the original nirvana
  - Domain Specific VLs – constraining the task
    - look at next lecture

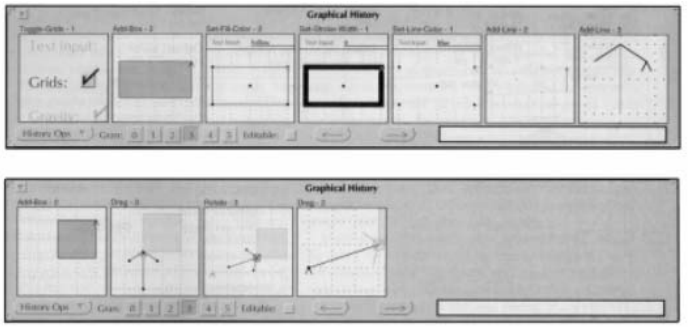
# Goals and Strategies of VP

- Goals
  - make programming more accessible to some audience (often end users)
  - improve correctness of performing programming tasks
  - improve speed of performing programming tasks.
    - NB what's a "programming task" – see attention investment later
- Strategies
  - Concreteness: express program using specific instances
  - Directness: feeling of directly manipulating object
  - Explicitness: making relationships explicit
  - Immediate visual feedback or liveness: automatic display of effects of manipulations, even to the extent of editing "code" of running programs (cf spreadsheets)
  - Small number of concepts
- Metaphor is important



# Example VPLs: Chimera

- Chimera (Kurlander, 1993) is an example of a programming by demonstration environment using comic book metaphor
- Captures concrete GUI editing operations and allows conversion to macros by selecting from comic strip history





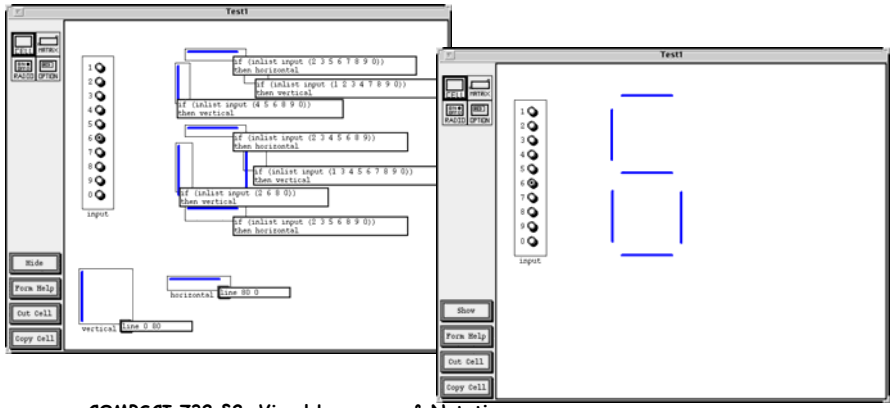

# Example VLs: Forms/3

- Forms/3 (Burnett 1995,98) uses a spreadsheet metaphor
- Programmer constructs forms with free format cells (not fixed to a grid) using direct manipulation
- Each cell has a formula which may refer to contents of other cells, possibly in other forms
- Linked formulae create a one-way constraint network – consistency is maintained
- Can construct types and instantiate them (prototype approach to OO) – cells can reference instances
- Can sketch shapes



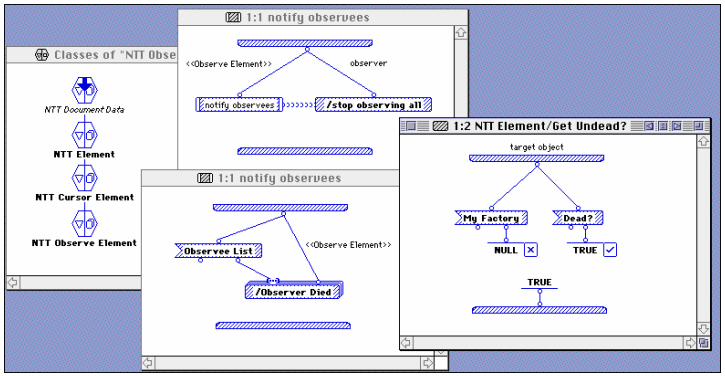
# Forms/3

- Aimed at non-programmers
- Much recent work on adding test tools (see EUSES project)



# Example VLs: Prograph

- Prograph (Cox et al 1989) uses a visual dataflow metaphor
  - dataflow metaphor very popular in VL – nodes for processing elements, arcs for dataflows






# Prograph

- Has a well developed OO framework
  - Dataflow "methods" for classes
  - GUI library framework allows rapid prototyping of applcns
- Has extensive debugging support
  - Reuses dataflow diagrams during execution with values instantiated to visualise execution behaviour
- Probably the only "successful" commercial general purpose visual programming language



# Example VLs: KidSim/Cocoa

- Cocoa (Smith et al 1994) uses a rule based metaphor combined with a 2-D cellular grid
  - Rules are specified using programming by demonstration
  - Aim is to make programming accessible to kids

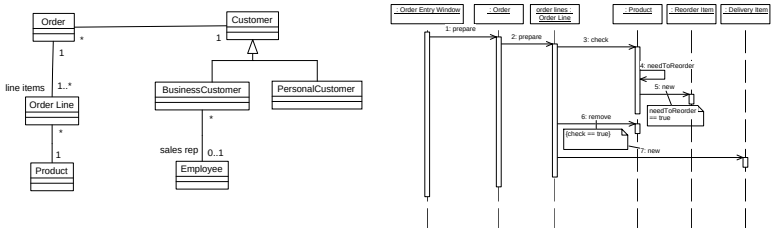



# KidSim/Cocoa

- Characters are defined
- Rule preconditions specify character proximities/ orientations
- Rule actions may remove or relocate characters, introduce new characters, etc
- Order-based disambiguation of rules if multiple rules for a character can fire
- Developed into commercial product: Stagecast Creator
- Several other similar languages, most notable of which is AgentSheets (Repenning). Alice has similarities.

# Example VLs UML

- UML is a collection of visual notations used for programming in the large
- Will explore in more detail in later lectures






# Designing and Evaluating VLs

- How "good" are the languages we have just looked at?
- How can we design such languages so they meet users needs?
- Difficult:
  - Combination of psychology, user interface design, abstraction skills, expressability, narrowness of task, etc, etc
  - Typical usability studies are VERY expensive
  - Need some lightweight "tools" to help us understand the impact of design decisions
- Look at:
  - Cognitive Dimensions
  - Attention Investment
  - Champagne Prototyping



# Cognitive Dimensions Framework

- Green and Petre 1996 (since developed by Blackwell)
- Establishes a set of "dimensions" to think about the tradeoffs made in implementing visual programming environments
  - Has had very strong influence on the VL community
  - Means of explaining effects of design decisions
- Comes out of cognitive psychology community
- Lightweight – doesn't need large usability studies to get useful insight
- Can be used for evaluation and also as a design aid



# Cognitive Dimensions

- 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?

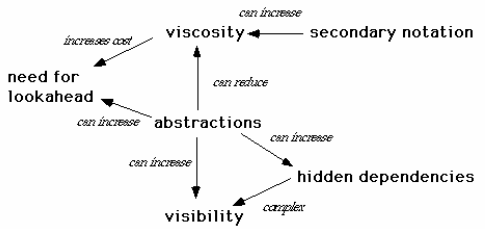
# Cognitive Dimensions

- 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?



# Use of Cognitive Dimensions

- Note the tradeoffs that occur
  - May add an abstraction that makes it easier to change things (reduced viscosity) but increases the difficulty of understanding (increased abstraction gradient and increased hidden dependencies).
  - See Green and Petre paper for several examples illustrating tradeoffs made




- Burnett provides a set of representation benchmarks that assist in operationalising the use of the CD framework.
  - See Burnett paper



# Cognitive Dimensions provides vocabulary

Verbatim transcript from a newsgroup discussion (real words from real users).

NB: this discussion referred to a version of Framemaker that is now obsolete.

- *A:* ALL files in the book should be identical in everything except body pages. Master pages, paragraph formats, reference pages, should be the same.
- *B:* Framemaker does provide this ... File -> Use Formats allows you to copy all or some formatting categories to all or some files in the book.
- *A:* Grrrrrrrrr ..... Oh People Of Little Imagination !!!!!!
- Sure I can do this ... manually, every time I change a reference page, master page, or paragraph format .....
- What I was talking about was some mechanism that automatically detected when I had made such a change. ( ..... ) Or better yet, putting all of these pages in a central database for the entire book .....
- *C:* There is an argument against basing one paragraph style on another, a method several systems use. A change in a parent style may cause unexpected problems among the children. I have had some unpleasant surprises of this sort in Microsoft Word.

Improved Discussion

- *A:* Framemaker is too viscous.
- *B:* With respect to what task?
- *A:* With respect to updating components of a book. It needs to have a higher abstraction level, such as a style tree.
- *C:* Watch out for the hidden dependencies of a style tree.
- *(further possible comments)*
- The abstraction level will be difficult to master; getting the styles right may impose lookahead.

From: An Introduction to the Cognitive Dimensions Framework, T R G Green

http://homepage.ntlworld.com/greenery/workStuff/Papers/introCogDims/index.html



# Attention Investment

- Theory to explain why people spend time doing programming
- Programming defined very broadly
- Defines "attention units": nominal amount of "concentration" applied
- Applies a cost benefit analysis approach to programming activities
  - Programming => automation to save time in the future
  - Has Cost: attention units to do the job
  - Investment: attention units expended towards a potential reward
  - Pay-off: reduced future cost from investment
  - Risk: probability that no pay-off or –ve pay-off results
- See Blackwell's paper.

# Champagne Prototyping

- A "cheap" method for early design evaluation
- Combines:
  - simple prototyping
    - used overlays and "look don't touch" approach
  - cognitive walkthroughs with credible participants
  - cognitive dimensions & attention investment for analysis

  to assist in answering questions at early design phase of visual environments
- Blackwell, Burnett and Peyton Jones, Champagne Prototyping: a research technique for early evaluation of complex end user programming systems, IEEE VL/HCC, 2004, 47-54



# Summary

- Have looked at a variety of VLs/VPEs
- Wide variety of metaphors and approaches used
  - Some are executable, some are just design notations
  - Some aimed at programmers, some at non programmers
- Have examined several approaches to evaluating visual language/environment design
  - Emphasis on "low cost" methods
- Will explore domain specific visual languages in more depth in next lecture
- Lead on to later sections
  - UML and the concept of meta modelling
  - Pounamu meta modeller