CS 66-460
Introduction to Graphical Human Machine Interfaces

An Overview of User Interface Software

Advanced Techniques

WEDNESDAY, MARCH 19, 1997

Instructor: G. Bowden Wise

Rensselaer Polytechnic Institute
Spring 1997

Novel Approaches are Needed for GUI Development

Software tools can reduce the effort required to build GUIs:

- Toolkits
- Virtual Toolkits
- Application Frameworks
- Interface Builders
- Prototyping Tools

We still have a long way to go to help with things like:

- the "insides of applcation windows"
  e.g., drawing application specific objects
- maintaining relationships in the interface
  e.g., widget layout, views and data couplings, etc
- cross-platform development
- built-in support for help, undo, abort
- automatic creation of parts of the interface from a specification
- evaluation of a resulting interface

Advanced Techniques

Research is showing promise for novel approaches:

- Demonstrational interfaces
  - allow the user interface designer to **draw** example graphics of the interface, and then to **demonstrate** the interactive behaviors to show how the interface should react to the user

- Constraint systems
  - Relationships between objects are declared once and then maintained by the system

Motivation: Need for Interactive, Graphical Tools

Direct Manipulation

- Allows the manipulation of objects by directly moving objects with the mouse and setting properties
  - Example: Interface builders, VisualBasic
- Limited to the static parts of the interface
  - No way to point at objects created or drawn by the user

Interactive Tools Improves Productivity

- Frameworks can improve productivity by factors of 3–5
- Interactive editors (e.g., interface builders) improve productivity by factors of 10–50 (!)
- It is more natural to specify graphical parts of an application using a graphical editor
- Graphical interaction does not require programming; can be used by non-programmers like graphic designers

Demonstrational Interfaces

- allow users to perform actions on concrete "example objects" (often by direct manipulation)
- the actions "demonstrate" the desired result using example values
- the system then generalizes those actions to produce a general purpose procedure or prototype
- very hard problem: no commercial systems yet

Demonstrational Interfaces Illustrative Examples

delete all postscript files in a directory

- user drags "file1.ps" to the trashcan
- user drags "file2.ps" to the trashcan
- the system infers that a similar action is being performed twice and pops up a dialog box asking the user if he wants all postscript files to be deleted

automatic widget layout

- user draws examples of the objects that will be created at run-time
- user draws objects in approximately the right places and the system
- the system then generalizes those operations to produce a general purpose procedure or prototype

Peridot
(Meyers, 1986–1988)

Programming by Example for Real-time Interface Design Obviating Typing

- First demonstrational tool, used by-example techniques to allow the creation of new widgets
- Goal: users can create interface without programming
- The user draws what the widgets look like, then moves the mouse around pressing buttons to demonstrate behavior
- From the drawing, it uses AI techniques to infer:
  - graphical constraints among the objects
    e.g., boxes should be the same size as the text
  - control structures
    e.g., iteration over all the items in a menu
  - how the mouse effects the graphics
    e.g., a check mark should follow the mouse
- Peridot then generalizes from the given examples to create a general purpose program

Peridot (continued)

- Uses **parameterized procedures** which allows parts of the interface to depend on values of parameters
  - e.g, a pop-up menu shrinks to fit around the menu item strings
  - Peridot asks designer for example values for each parameter
- Two types of constraints are used
  - **graphical**: ties various graphical objects together so that when one changes the other is updated
  - **data**: insures graphical objects are updated when any special variables (called Action Values) are updated
- techniques used:
  visual programming
  programming by example
  constraints
  plausible inferencing
- However, lacks a programming interface and offers no way to use existing toolkit items or create application specific objects

Constraint Systems

- Large, complex applications have thousands of interdependent relationships
- A graphical application must
  - lay out objects
  - display feedback for input operations
  - keep views consistent with underlying data
- Constraints provide a convenient way to specify relationships
- Constraint solvers maintain the constraint relationships
- Sketchpad pioneered the use of constraints in a drawing editor

Constraint Systems

- Types of constraints:
  - one-way: only one object in the constraint may change
  - multi-way: allow any object to change
- Constraint solving:
  - lazy:
    evaluate a constraint only if it affects a result that the user requests
    * may contain out-of-date values in the constraint; need bookkeeping, may cause delays
  - eager:
    evaluates a constraint immediately when a value changes
    * may update unneccesary objects (off screen) but everything is kept up-to-date

Garnet (Myers et al, 1988+)

Generating an **a**malgam of <u>r</u>eal-time **n**ovel **e**ditors and **t**oolkits

- Garnet is a tool to aid in the design and implementation of highly interactive, graphical direct manipulation user interfaces
- particularly well suited for:
  - box and arrow diagram editors (MacProject)
  - conventional drawing programs (MacDraw)
  - icon manipulation programs (Macintosh Finder)
  - graphical programming languages
  - tree and graph editors
  - board games
  - simulation/process monitoring programs
  - UI construction tools (Garnet)
  - some forms of CAD/CAM
- Primarily influenced by Peridot

Garnet (continued)

- comprehensive user interface development environment
- uses Lisp and runs on X Windows platforms
- consists of
  - an object-oriented programming system
  - a constraint system
  - a graphical object system
  - a system for handling input
  - two complete toolkits: one is Motif-like
- with many high-level tools:
  - Gilt: interface builder
  - Lapidary: interactive design tool
  - C32: spreadsheet system for managing constraints
  - Jade: dialog box system

Garnet
Object-Oriented Programming System

- uses a prototype-instance model rather than inheritance model
- does not distinguish between instances and classes
- any instance can be a prototype
- all data and methods are held in slots
- slots that are not overridden inherit their values from their prototypes
- slots can be added and removed at run-time
- instances can add any number of new slots; if a program sets a value for a slot that does not exist, the slot is created
  - advantage: easy to create local data
  - disadvantage:
    no type checking of slots (can hold any value);
    no way to check slot accessing/setting at compile/run time since all slot names are legal

Garnet
Prototype Instance Example

Garnet
Object-Oriented Programming System (continued)

Advantages of prototype-instance model

- more dynamic and flexible than familiar class instance models
- changing the prototype automatically propagates to all instances
- example: user edits the menu prototype in an interface builder, all menus in the system immediately change a

Potential disadvantage of prototype-instance model:

- retrieving a slot's value could be costly, since it may require a search up the entire inheritance hierarchy
  - Garnet alleviates this problem by keeping local caches of inherited values
- dynamic addition of slots means that a fixed storage scheme like a structure or record cannot be used to hold slots
  - Garnet keeps commonly accessed slots in records, but newly added ones in lists

Garnet
Constraint System

- A constraint is a relationship among objects that is maintained when any of the objects changes
- Natural way to express common relationships in GUIs
- Example: keeping an arrow line attached to two boxes, even when the objects are moved
- One-way constraints: if the other object changes, the object with the formula is reevaluated, but not vice versa
  - if either box is moved, the arrow moves; but, moving the arrow does not move the boxes

Garnet
Constraint System (continued)

- Garnet can handle cycles in certain cases: A can depend on B and B can depend on A
  - Garnet makes sure it only goes around once
- Garnet also allows objects referenced in the constraint to be indirectly accessed via a variable
  - menu example: reverse video rectangle moves whenever the variable `:obj-over` changes
- Constraints can also be used to:
  - to control behaviors (e.g., resize or move an object based on which button is down)
  - to connect graphical objects to application specific objects (e.g., constrain a value of a gauge to a data value in the application)

Garnet
Graphical Object System

Opal is Garnet's graphical object system

- provides default values for all object properties
- simple objects can be designed by specifying only the necessary parameters

- to create a rectangle at (10,20) size 30x40:

```
(create-instance 'myrect rectangle
  (:left 10) (:top 20)
  (:width 30) (:height 40)
)
```

- integrated with the constraint system, so any object property can be specified with formulas instead of numbers
- automatically handles drawing and erasing (only when necessary)

Garnet
Graphical Object System (continued)

Aggregates can be used to group graphical objects into collections

- When an instance of an aggregate is created, an instance of each component of the aggregate is created
- Changes to the aggregate are immediately reflected in all instances
  - Example: menu items

Aggrelists are used to lay out a list or table of elements

- contain a single prototype for all the elements
- each element uses a different value for some value of the prototype
- Aggrelists allow items to be displayed horizontally, vertically, or in multiple rows, or in a custom layout (e.g., trees?)
  - Example: menus as aggregates

Garnet
Input Handling

- Handling keyboard, mouse, and other input is hard in GUIs
- Programmers using toolkits usually must write code to deal with the stream of events the window manager dispatches to the application

- Garnet uses **interactors** to encapsulate input device behaviors
- There are just a few distinct behaviors in GUIs (e.g., select from a menu, resize a window, input a string of text, etc)

Garnet
Types of Interactors

- **Interactors** capture common behavior and make them available in a central place, while still allowing customization by applications
  - look and feel independent
  - let designers separate the details of object behavior from the application and the graphics
  - support multiple input devices in parallel
  - let users operate on different applications (separate windows)
  - hide the complexities of the underlying window system

Only six:

- Menu-Interactor
- Move-Grow-Interactor
- New-Point-Interactor
- Angle-Interactor
- Trace-Interactor
- Text-String-Interactor

Garnet
Interactors (continued)

Interactors:

- are parameterized so that the programmer can control
  - the mouse and keyboard events that initiate it
  - optional parameters to be called upon completion
  - objects on which the object operates
  - (optional) object that receives feedback

Similar to the **model-view-controller** of Smalltalk:

- model: the application objects
- view: Opal graphical objects
- controller: interactors

Garnet
Widgets and Gadgets

- On top of Opal and interactors are a collection of widgets
- Garnet provides two complete toolkits; one looks like Motif

Garnet
Debugging Tools

- Defining interfaces with constraints and interactors is natural and effective
- but... difficult to find bugs in the code
- with constraints, a bad value, can lead to bad values all over
- Garnet provides a large number of debugging tools:
  - tracing mechanisms (how a slot got its value)
  - where objects are on the screen (or why not visible)
  - event monitor (what happens when the mouse or keyboard is used)
  - object inspectors/browsers

Garnet
Lapidary

Lapidary is the interface builder

- provides a graphical front end to most of the underlying toolkit features so that a program's graphical aspects can be specified pictorially

A designer

- draws pictures of application-specific objects that the application will create and maintain at run-time
- including:
  - the graphical entities the end-user will manipulate
  - the feedback showing which objects are selected (handles around objects)
  - dynamic feedback objects (hairline boxes as object is dragged)
- the designer creates prototypes in Lapidary
- the application program creates instances from the prototypes
- also supports use of pre-defined widgets

Garnet
Lapidary (continued)

Behavior can be specified by:

- attaching constraints to objects using iconic menus (e.g., check marks by selected menu item)

- by demonstration
  - the designer draws one state, then another, and Lapidary automatically constructs the constraints to change the object between the two states

Garnet
Lapidary (Graph-Editor Example)

Garnet
Jade Dialog Box Creator

- creates a dialog from a specification of its contents
- the specification is independent of any particular look-and-feel
- textual specification has a lisp-like syntax, and specifies
  - the text that appears in the dialog box
  - the type of input required (choice in a set, number in a range)
- when Jade is given a spec and a particular look and feel
  - it chooses the correct interaction techniques
  - heuristics for placement of objects in the dialog are specific to a particular look and feel
  - heuristics attempt to create an attractive display based on graphic arts principles
- once the dialog is generated, the designer may edit the result
  - the changes are saved in an exception file

Garnet
C32 Spreadsheet

- Allows for specification of more powerful constraints
- Allows designers to enter arbitrary Lisp constraint expressions
- Also let's designer monitor and debug interfaces by watching spreadsheet values while the user interface is running