Overview

• Motivation and history
• Survey of current successful tools
• Tools and future UIs
Why Software Tools?

- As user interfaces become easier to use, they become harder to implement.
- Nearly all systems have a user interface
  - On average the UI is half of the code and takes half of the development effort.
- Tools reduce required development resources
  - The MacApp framework reduced time to produce Macintosh user interfaces by a factor of 4 or 5
  - Programmers using NeXTStep wrote 1/6 of the code and used half of the time.

Advantages of Tools

- Higher quality interfaces
  - Enable rapid prototyping
  - Ease modification
  - Tool effort can be greater than any one application
  - Allow design by UI specialists
- More economical to create and maintain
  - Easier to analyze UI specifications
  - Less code to write
  - Better modularization and separation of issues
  - More reliable since much more effort is put into the tool.
  - Lower skill needed by developers

History

- Many of the tools that are used today
  - Are based on research systems
  - Had their roots in the 70’s and 80’s
- Window managers
  - NLS, ‘67; overlapping windows, Kay ‘69: Smalltalk, ‘74
- Toolkits
  - Apple Macintosh Toolkit ‘86
  - Support consistency by providing a “path of least resistance”
History

- Event languages
  - University of Alberta, '85
  - HyperTalk, Visual Basic, Flash/ActionScript
- Graphical interface builders
  - Trillium '86, MenuLay '83
  - Usually paired with an event language
  - Visual Basic, Visual C++, JBuilder, Forte for Java
- Component systems
  - Andrew '88
  - ActiveX, OpenDoc, Java Beans

Scripting Languages

- Interpreted languages give:
  - Rapid revision and test
  - Immediate feedback
- Smalltalk '81
- Current systems: tcl/tk, Python
- HyperText
  - Bush '45; word coined '65.
  - Hyperties '86, 1st system with highlighted and embedded links
  - WWW as a UI

Current State

- Homogeneous interface hardware and style
- Hardware, networked workstation with:
  - Keyboard and pointer (mouse)
  - Color display between 1024x768 and 1600x1200
- Interface style
  - WIMP based on some Macintosh look-alike.
- Development Tools
  - Graphical interface builder to an event-based language
  - Integrated with a object-oriented language
  - Object browser
  - Symbolic debugger
Why it works?

- Homogenous hardware and software means:
  - Essentially the same program works on all platforms
  - Limited diversity of interaction methods
    - Menu select, button press
    - Form fill in
    - Drag and drop
- Development tools
  - Event organization well suited to user-centered, direct-manipulation interfaces
  - Object-oriented languages well matched to user interface programming

The Future

- Ubiquitous computing
  - A computer in every gadget
  - Wide variety of input devices
    - Stylus, key pad, voice, buttons
    - Displays from a few square centimeters to a few square meters
    - Ad hoc networks of computers cooperating
    - People don’t expect behavior like a computer
- 3D displays and virtual reality
- Mobil computing
- In other words, a heterogeneous environment

Implications of the Future

- The WIMP model doesn’t work well on
  - Very small screens (too little space for windows & menus)
  - Very large screens (pointless to manage a resource that isn’t scarce)
  - Systems without a pointer
  - Ad hoc groups of systems
    - Problems with data distribution
    - Problems with input distribution
    - Current interfaces assume the user is focused solely on them.
    - Compounded by heterogeneous environments
**Implications of the Future**

- Virtual reality
  - No standard I/O devices ⇒ custom development
  - No standard interaction techniques ⇒ toolkits impossible
- Ditto for 3D displays
- Mobil Computing
  - Context of current interaction important
  - Devices are small and lightweight
  - Services may be limited
    + Pointer may be more awkward than a mouse
    + No printer, slow network, …

**Current methods may no be adequate:**
- Graphics-based tools are aimed at 2D displays with fairly high resolution
  + Most devices won’t have one
- Based on events
  + Input by speech and gesture is not easy to model with events
- One user providing input
  + Doesn’t match multi-system environments with a lot of non-UI events

**Survey of Current Successful Tools**
Criteria for Evaluating Tools

- **Comprehensiveness:**
  - What parts of the UI are addressed?
  - What parts need support?
- **Threshold:**
  - What is the learning curve?
  - Are there multiple steps?
  - How quickly can you get something useful?
- **Ceiling:**
  - How much can be done?

Criteria for Evaluating Tools

- **Predictability**
  - Can I understand how the tool produces what it does?
  - Can I understand how to get the results I want?
- **Path of least resistance**
  - Does the tool produce good design with little effort?
  - Does the tool make bad designs difficult?
- **Mature technology**
  - Does the tool address issues relevant to current UI systems?
  - Is the type of UI well enough understood?

Toolkits

- **Raise the ceiling**
  - Provide a common interface
  - Reuse code
- **Lower resistance to consistency**
- **Have low threshold**
  - Windows are very similar to displays
  - Widgets provide simple interaction models
- **Mature, address issues with current UIs, which are well understood**
- **Not very comprehensive, but address a part of the UI that needs support**
- **For the future:**
  - Windows won't work in 3D, VR, and small display systems
  - Toolkits provide a model for building new UI types, but current widgets are inadequate for 3D and VR.
Event Languages

- Raise ceiling
  - Provide a common input model
  - Well matched to direct manipulation model
- Have moderate threshold
  - Disperse program execution thread
  - Difficult to debug
- Mature, issues w/current UIs, well understood
- Address a need for non-sequential response
- Future
  - Model many interaction types well
  - Can be extended to multiple sources
  - Do not handle speech, gestures, or other continuous inputs well

Graphical Interface Builders

- Don’t affect the ceiling
- Lower threshold for graphical layout
- Address a small area that is well understood and mature
- Make forms, buttons, and menus easy
  - other things are still hard
- For the future:
  - Of limited use in ubiquitous, 3D, and VR environments

Component Systems

- Raise the ceiling
  - Complex interactions can be packaged with simple interfaces
- Moderate to high threshold depending on packaging
- High “least resistance” value
- Address common special purpose behaviors
- For the future
  - When things mature they can become components
  - Need a better method to “plug into” applications
Scripting Languages

- Lower the threshold by providing rapid feedback
- Same threshold as programming languages
- Otherwise indistinguishable from other methods
- For the future
  - If they adapt, they will survive

Techniques that are Still in the Labs

Formal Languages

- Targeted at dialog management
- Two types:
  - State machine based
  - Grammar based
- Good at describing sequential systems
  - WIMP systems aren’t sequential
  - Grammars are especially sequential
  - State machines can be extended to handle parallel dialogs
- State machines are good at specifying finite state widgets
Secure Switch Regions

On Mid Off
Out

Interaction Object Graph
Secure Switch

Formal Languages

- High threshold
  - New way of thinking, not code
- Tend to encourage stiff, sequential designs → path of least resistance leads to bad interfaces
- Future:
  - Heterogeneous systems may require dialog specification and retargeted presentation
  - Designing new interaction methods can use a higher level of abstraction than code
Constraints

- Declarative specification of relationships, system maintains them
- Currently successful for geometry management
- Evaluation
  - High threshold: new way of thinking, hard to debug
  - Unpredictable
- Future
  - Useful for specifying model-UI parameter relationships, widgets and UI could be redone with semantics maintained
  - Natural for VR and other interactions specifying continuous relationships

Model-Based Techniques

- Automatically generate a UI from a specification of the data model
- Evaluation:
  - High threshold: new way of thinking
  - Low ceiling: UIs usually worse than code-built
  - Unpredictable
- Future
  - Separates I/O specification from interaction, unimportant with WIMP, but can be valuable for heterogeneous environments

Tools and Future UIs
Ubiquitous Computing

- The “standard” platform disappears:
  - Techniques for device independent specification needed
  - Model-based techniques
  - Dialog specification with formal methods
  - 3D displays will require entirely new interaction techniques
- Coordinating multiple devices becomes important
  - Whole new methods needed
  - Context sensitivity is very important

Speech Recognition

- Speech requires coordination between the context and the recognizer for best results
  - Disambiguation is easier when grammar eliminates some choices
  - Limited vocabularies improve accuracy

Gesture Recognition and Multi-Modal Interaction

- Gestures may require feedback and beginning interaction before a complete gesture is recognized
- Multi-modal interactions imply:
  - Command/object sequencing is different
  - Command/object sequencing is not fixed
- Current interface techniques do not require much knowledge of the application’s content
3D Interfaces

Basic issues remain to be worked out:
- Navigation
  - Immersion is usually slower than “fish tank”
  - No set of “standard” interactions
  - New, non-paper metaphors
- Scene graph appears to be a reasonable building block

End-User Programming

Largely unknown outside of spreadsheets
Threshold problem
  - Need for incremental learning without “cliffs”
  - Boolean conditions
  - Iteration over a set (looping)
  - Procedures or functions
  - Abstract parameters
End users work in communities, but tools are designed for individuals
  - Remember your experience with Flash
Solutions to these problems can benefit everyone

Revisiting Comprehensiveness

Parts of UI development performed by different tool types

<table>
<thead>
<tr>
<th>System Type</th>
<th>End-User Model</th>
<th>Task Analysis</th>
<th>Map to Computer Objects</th>
<th>Define Layout</th>
<th>Define Dialog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>No</td>
<td>No</td>
<td>Manual</td>
<td>Some</td>
<td>Yes</td>
</tr>
<tr>
<td>Framework</td>
<td>No</td>
<td>No</td>
<td>Manual</td>
<td>No</td>
<td>Some</td>
</tr>
<tr>
<td>UI Builder</td>
<td>No</td>
<td>No</td>
<td>Manual</td>
<td>Yes</td>
<td>Some</td>
</tr>
<tr>
<td>Model-Based</td>
<td>Some</td>
<td>Some</td>
<td>Auto</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Other Tool Issues

- Support for "challenged" users
  - Aging population means more
- Support for overlapping, non-rectangular, semi-transparent components
- Support for finding and using the right component
- Support for inattentive use
- Support for evaluation of interfaces
- Operating system issues
  - Time in the interface
  - No differentiated memory model
  - Undos/reds across sessions, infinite versioning
  - Better system status checking

Questions?