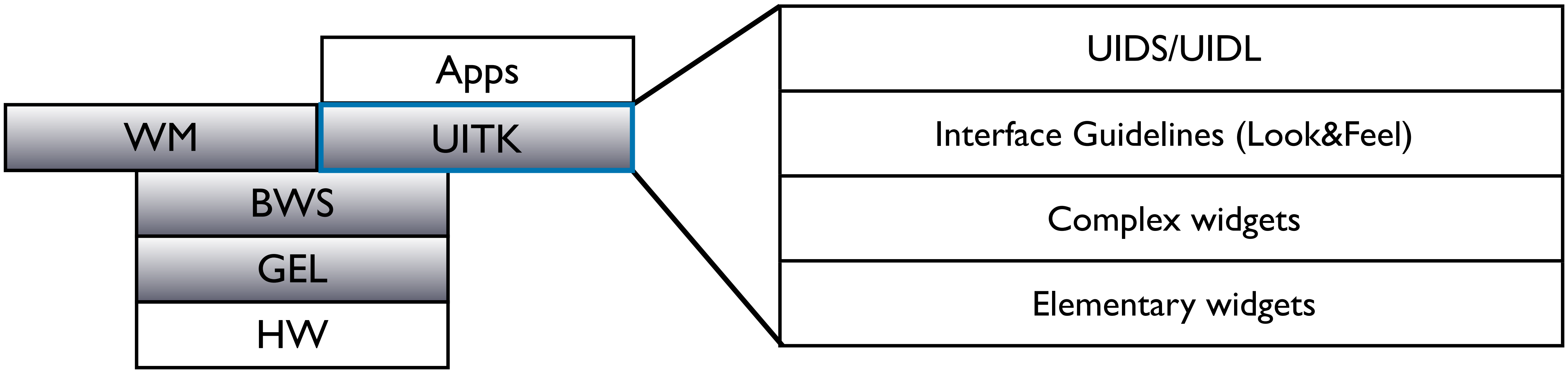
Designing Interactive Systems II

Computer Science Graduate Programme SS 2009

Prof. Dr. Jan Borchers
RWTH Aachen University

http://hci.rwth-aachen.de

User Interface Toolkit



- Motivation: Deliver API
  - problem/user-oriented instead of hardware/BWS-specific
  - 50–70% of SW development go into UI
    - UITK should increase productivity

UITK: Concept

- Two parts
  - Widget set (closely connected to WS)
  - UIDS (User Interface Design System to support UI design task)
- Assumptions
  - UIs decomposable into sequence of dialogs (time) using widgets arranged on screen (space)
  - All widgets are suitable for on-screen display (no post-desktop user interfaces)
  - Note: decomposition not unique

UITK: Structure

- Constraints
  - User works on several tasks in parallel → parallel apps
  - Widgets need to be composable, and communicate with other widgets
  - Apps using widget set (or defining new widgets) should be reusable
- Structure of procedural/functional UITKs
  - Matched procedural languages and FSM-based, linear description of app behavior
  - But: Apps not very reusable

UITK: Structure

- OO Toolkits
  - Widget handles certain UI actions in its methods, without involving app
  - Only user input not defined for widget is passed on to app asynchronously (as seen from the app developer)
    - Matches parallel view of external control, objects have their own “life”
  - Advantage: Subclass new widgets from existing ones
  - Disadvantage:
    - Requires OO language (or difficult bridging, see Motif)
    - Debugging apps difficult

UITK: Control Flow

- Procedural model:
  - App needs to call UITK routines with parameters
  - Control then remains in UITK until it returns it to app
- OO model:
  - App instantiates widgets
  - UITK then takes over, passing events to widgets in its own event loop
  - App-specific functionality executed asynchronously in *callbacks* (registered with widgets upon instantiation)
  - Control flow also needed between widgets

Defining Widgets

- Widget :

$$(W = (w_1 \ldots w_k), G = (g_1 \ldots g_l), A = (a_1 \ldots a_m), i = (i_1 \ldots i_n))$$

  - Output side: windows W, graphical attributes G
  - Input side: actions A that react to user inputs I
  - Mapping inputs to actions is part of the specification, can change even at runtime
  - Actions can be defined by widget or in callback
- Each widget type satisfied a certain UI need
  - Input number, select item from list,...

Simple Widgets

- Elementary widgets
  - Universal, app-independent, for basic UI needs
  - E.g., button (trigger action by clicking), label (display text), menu (select *1* of *n* commands), scrollbar (continuous display and change of value), radio button (select *1* of *n* attributes)

In-Class Exercise: Button

In-Class Exercise: Button

- What are the typical components (W, G, A, I) of a button?

In-Class Exercise: Button

- What are the typical components (W, G, A, I) of a button?
- Sample solution:
  - W=(text window, shadow window)
  - G=(size, color, font, shadow,...)
  - A=(enter callback, leave callback, clicked callback)
  - I=(triggered with mouse, triggered with key, enter, leave)

Simple Widgets

- Container widgets
  - Layout and coordinate other widgets
  - Specification includes list C of child widgets they manage
  - Several types depending on layout strategy
- Elementary & Container widgets are enough to create applications and ensure look&feel on a fundamental level

Complex Widgets

- Applications will only use subset of simple widgets
- But also have recurring need for certain widget combinations depending on app class (text editing, CAD,...)
    - Examples: file browser, text editing window
- Two ways to create complex widgets
    - Composition (combining simple widgets)
    - Refinement (subclassing and extending simple widgets)
    - Analogy in IC design: component groups vs. specialized ICs

Widget Composition

- Creating dynamic widget hierarchy by hierarchically organizing widgets into the UI of an application
  - Some will not be visible in the UI
- Starting at root of dynamic widget tree, add container and other widgets to build entire tree
  - Active widgets usually leaves
  - Dynamic because it is created at runtime
  - Can even change at runtime through user action (menus,...)

Widgets and Windows

- The dynamic widget tree usually matches geographical *contains* relation of associated BWS windows
- But: Each widget usually consists of several BWS windows

→ Each widget corresponds to a subtree of the BWS window tree!

→ Actions A of a widget apply to is entire geometric range except where covered by child widgets

→ Graphical characteristics G of a widget are handled using priorities between it, its children, siblings, and parent

Refinement of Widgets

- Create new widget type by refining existing type
- Refined widget has mostly the same API as base widget, but additional or changed features, and fulfills Style Guide
- Not offered by all toolkits, but most OO ones
- Refinement creates the Static Hierarchy of widget subclasses
- Example: Refining text widget to support styled text (changes mostly G), or hypertext (also affects I & A)

Late Refinement of Widgets

- App developer can compose widgets
- Widget developer can refine widgets
- → User needs way to change widgets
- → Should be implemented inside toolkit
- Solution: Late Refinement (see WM for discussion)
- Late refinement cannot add or change type of widget characteristics or the dynamic hierarchy
- But can change values of widget characteristics

Style Guidelines

- How support consistent Look&Feel?
  - Document guidelines, rely on developer discipline
    - E.g., Macintosh Human Interface Guidelines (but included commercial pressure from Apple & later user community)
  - Limiting refinement and composition possible
    - Containers control all aspects of Look&Feel
    - Sacrifices flexibility
  - UIDS
    - Tools to specify the dialog explicitly with computer support

Types of UIDS

- Language-oriented
  - Special language (UIL) specifies composition of widgets
  - Compiler/interpreter implements style guidelines by checking constructs
- Interactive
  - Complex drawing programs to define look of UI
  - Specifying UI feel much more difficult graphically
    - Usually via lines/graphs connecting user input (I) to actions (A), as far as allowed by style guide
- Automatic
  - Create UI automatically from spec of app logic (research)
- *Examples in upcoming lectures*