User Interface Layer

Module-5

User interface layer and its execution Framework, Input /Output processes,

Understanding the UI Layer and Its Execution Framework

- Interactive applications are implemented and executed using the user interface (UI) software layers (collectively the UI layer).
- The UI layer refers to a set of software that operates above the core operating system (and underneath the application).
- It encapsulates and exposes system functions

- Fluid input and output (I/O)
- Facilitation of development of I/O functionalities (in the form of an application programming interface/library [API] or toolkit)
- Run-time management of graphical applications and UI elements often manifested as windows or graphical user inter- face (GUI) elements (in the form of separate application often called the window manager)

- the UI is what the user sees, the UI state is what the app says they should see. Like two sides of the same coin, the UI is the visual representation of the UI state. Any changes to the UI state are immediately reflected in the UI.

- A "user interface layer" example would be the visual elements and interactive components on a website or app that a user directly interacts with, such as buttons, text fields, dropdown menus, and icons, which allow them to navigate and perform actions within the application; essentially, the "front-end" of a software where users see and manipulate data

Key points about the user interface layer:

- **Direct interaction:**

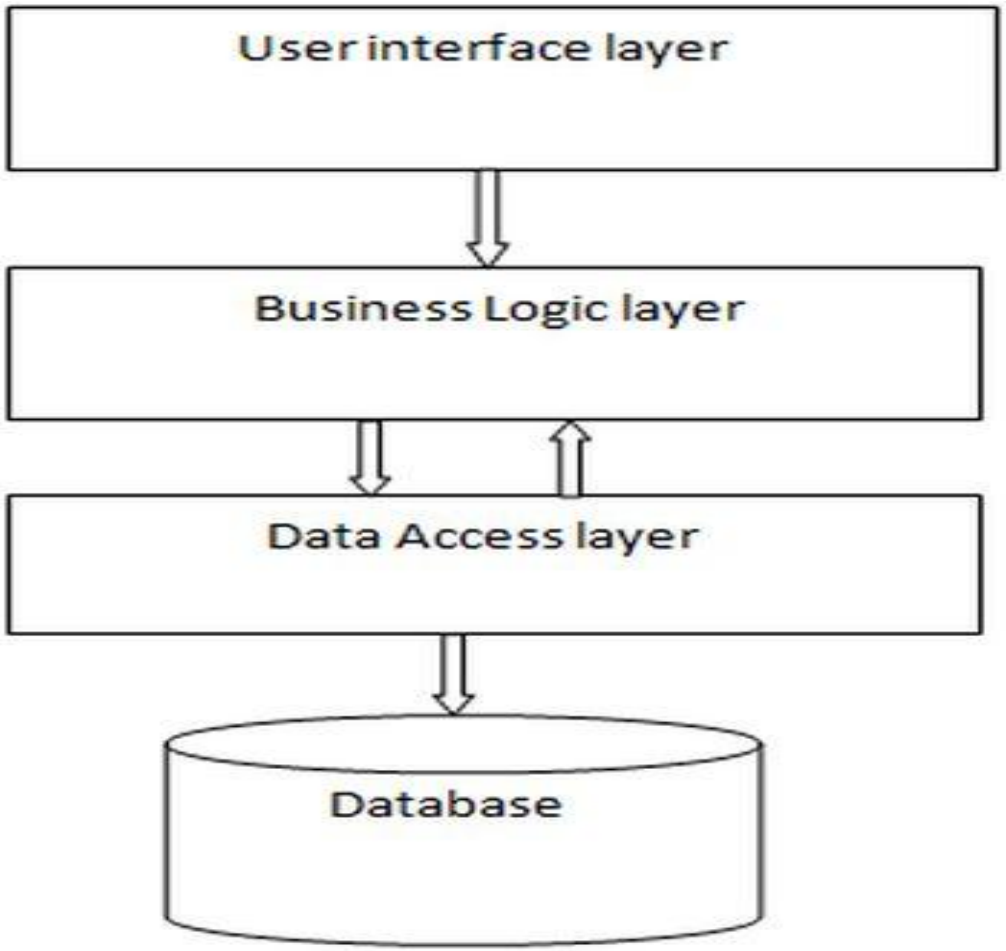
  This is the layer where users directly engage with the system through visual cues and input methods.

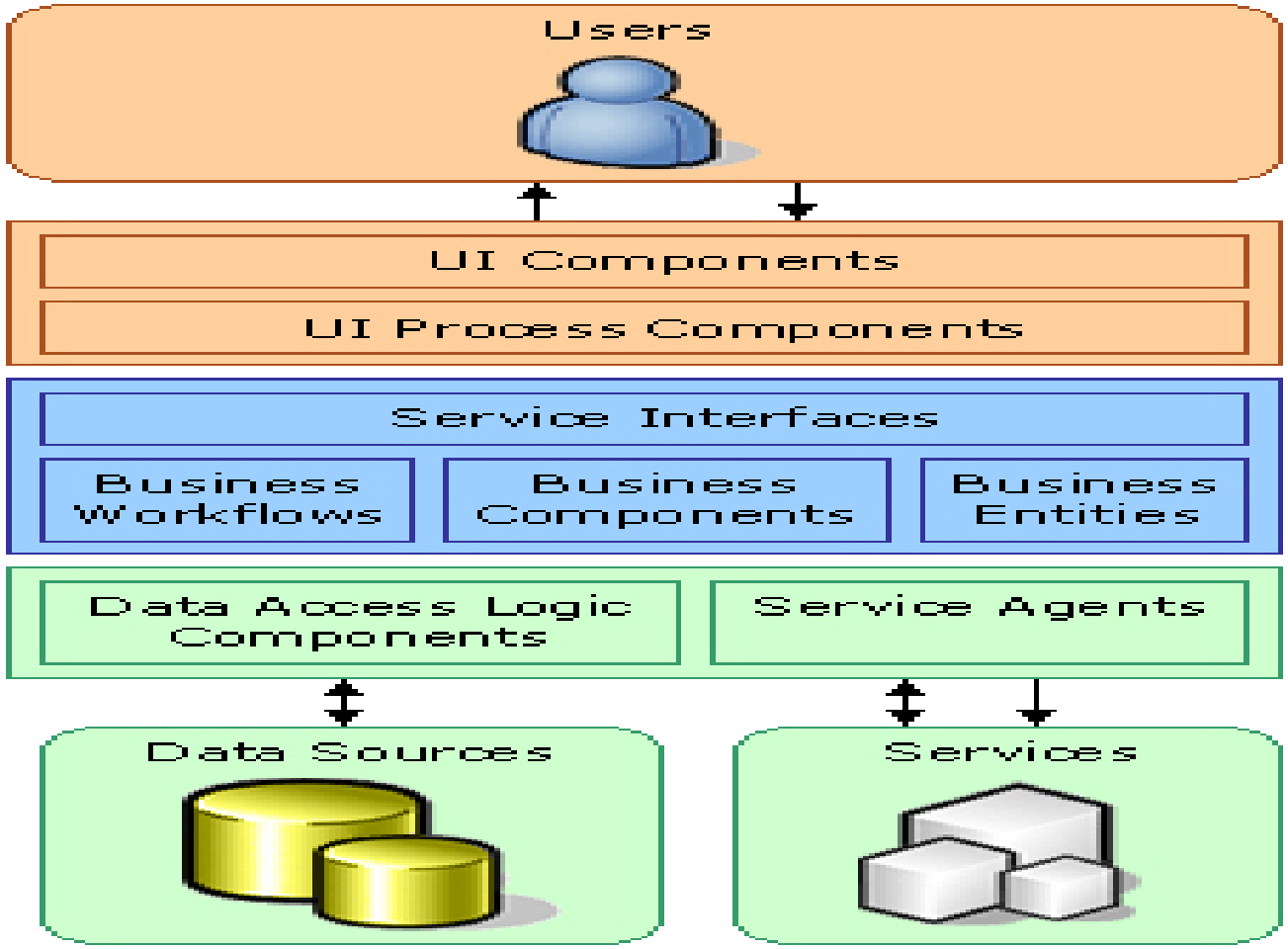
- **Presentation layer:**

It is responsible for presenting information to the user in a visually appealing and understandable way.

Examples of user interface layers in different contexts:

- **On a website:** The navigation bar, search bar, product listings, and "add to cart" buttons on an online store.
- **On a mobile app:** The home screen icons, touch-based controls in a game, and settings menus.
- **On a desktop application:** The file explorer window with folders and files, the toolbar with common functions, and dialog boxes for inputting information


User interface layer
Business Logic layer
Data Access layer
Database


Users
UI Components
UI Process Components
Service Interfaces
Business Workflows
Business Components
Business Entities
Data Access Logic Components
Service Agents
Data Sources
Services

- **Presentation**. The presentation layer provides the application's user interface (UI). Typically, this involves the use of Windows Forms for smart client interaction, and ASP.NET technologies for browser-based interaction.
- **Business**. The business layer implements the business functionality of the application. The domain layer is typically composed of a number of components implemented using one or more .NET - enabled programming languages. These components may be augmented with Microsoft .NET Enterprise Services for scalable distributed component solutions and Microsoft BizTalk Server for workflow orchestration.
- **Data** The data layer provides access to external systems such as databases. The primary .NET technology involved at this layer is ADO.NET. However, it is not uncommon to use some .NET XML capabilities here as well.
- Each layer should be structured as described in the following paragraphs.
- **Presentation Layer**
- For most business applications, a form metaphor is used to structure the presentation layer. The application consists of a series of forms (pages) with which the user interacts. Each form contains a number of fields that display output from lower layers and collect user input.
- Two types of components that implement forms-based user interfaces are:
- User interface components
- User interface process components

the Physical User Interface Layer

- The physical user interface layer includes physical files, such as templates, Siebel tags, style sheets, and other metadata that reference files.
- These files control the layout and the look and feel of the Siebel client.
- The physical user interface layer separates definitions for style and layout from user interface objects in the Siebel repository.
- This separation allows you to define definitions for style and layout across multiple objects of the Siebel client.

- Since most interfaces are graphical, the UI layer uses a two- or three-dimensional (2-D or 3-D) graphical system based on which GUI elements are implemented .
- UI layer is largely composed of

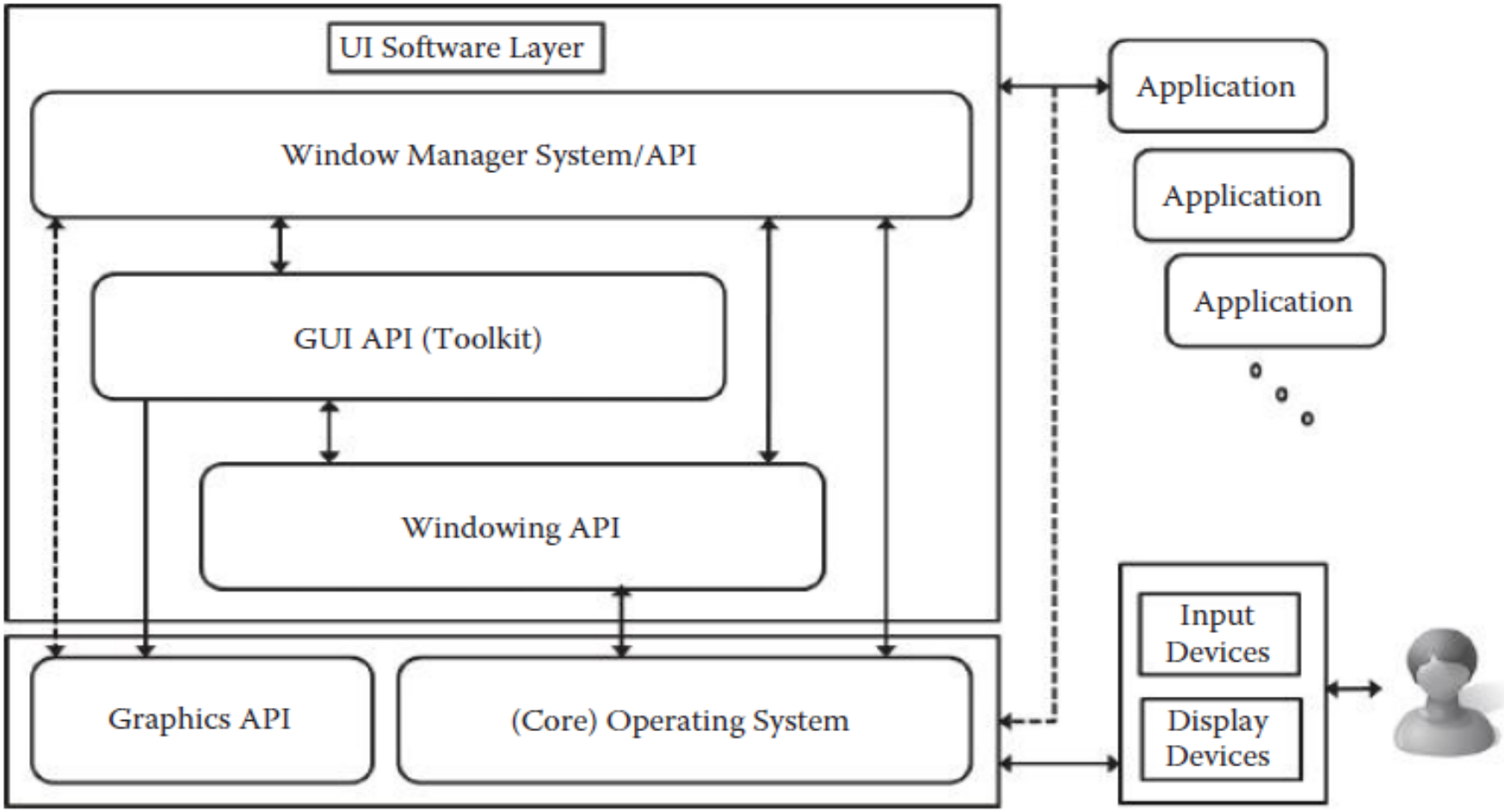
  (a) an API for creating and managing the user interface elements (e.g., windows, buttons, menus)

  (b) a window manager to allow users to operate and man- age the applications through its own user interfaces.

Example: E-commerce Application

- **Presentation Layer**: Displays products, handles user interactions (add to cart, login, etc.)
- **Application Layer**: Manages the flow of actions (e.g., adding products to the cart and processing orders)
- **Business Logic Layer**: Contains rules like calculating discounts, verifying payment, and managing inventory
- **Data Access Layer**: Interacts with the database to retrieve products and store user information
- **Database Layer**: Stores product details, user accounts, and transaction history

User interface software layer for a window-based multitasking UI.



- the UI layer as part of the system software in many computing platforms. The user interacts with the window/ GUI-based *applications* using various input and output *devices*.
- At the same time, aside from the general applications, the user interacts with the computer and manages multiple application windows/ tasks (e.g., resizing, focusing, cutting and pasting, etc.) using the (background running) *window manager*
- The window manager is regarded as both an application and API

- . User applications are developed using the APIs that represent abstracted I/O-related function- alities of the UI layer, such as those for window managing (resizing, iconifying, dragging, copy and paste, etc.), GUI elements and wid- gets (windows, menus, buttons, etc.), and basic windowing (creating/destroying window, deactivating window, etc.).
- These APIs are abstracted from the even lower-level APIs for 2-D/3-D graphics and the operating system.
- the architecture described here can be equally applied to nonwindow-based systems, such as those for layer- based systems* (e.g., mobile phones). Through such an architecture and abstraction, it becomes much easier to develop and implement interactive applications and their user interfaces.

Key points about the GUI layer

- **Visual presentation:**
  - It renders the visual elements on the screen, allowing users to interact with applications through graphical components like buttons, text boxes, and sliders.
- **Event handling:**
  - It manages user input events like mouse clicks, keyboard presses, and touch gestures, translating them into actions within the applications.
- **Window management:**
  - It controls the positioning, resizing, and overlapping of multiple windows on the screen, allowing for multitasking.

Difference between presentation layer and user-interface

- *Presentation layer* is term in the taxonomy of code and associated resources.
- *User Interface* is the implementation of the intended User Experience in terms of page layout, page transitions and page control elements. (I am using "page" loosely here - you can replace it with "form" or "window").

2.Input and Output at the Low Level

- At the lowest level, inputs and outputs are handled by the *interrupt* mechanism of the system software (operating system).
- An interrupt is a signal to the processor indicating that an event (usually an I/O event) has occurred and must be handled
- An interrupt signal is interpreted so that the address of its handler procedure can be looked up and exe- cuted while suspending the ongoing process for a moment.
- After the handler procedure is finished, the suspended process resumes again
- An arrival of an interrupt is checked at a very fast rate as part of the processor execution cycle

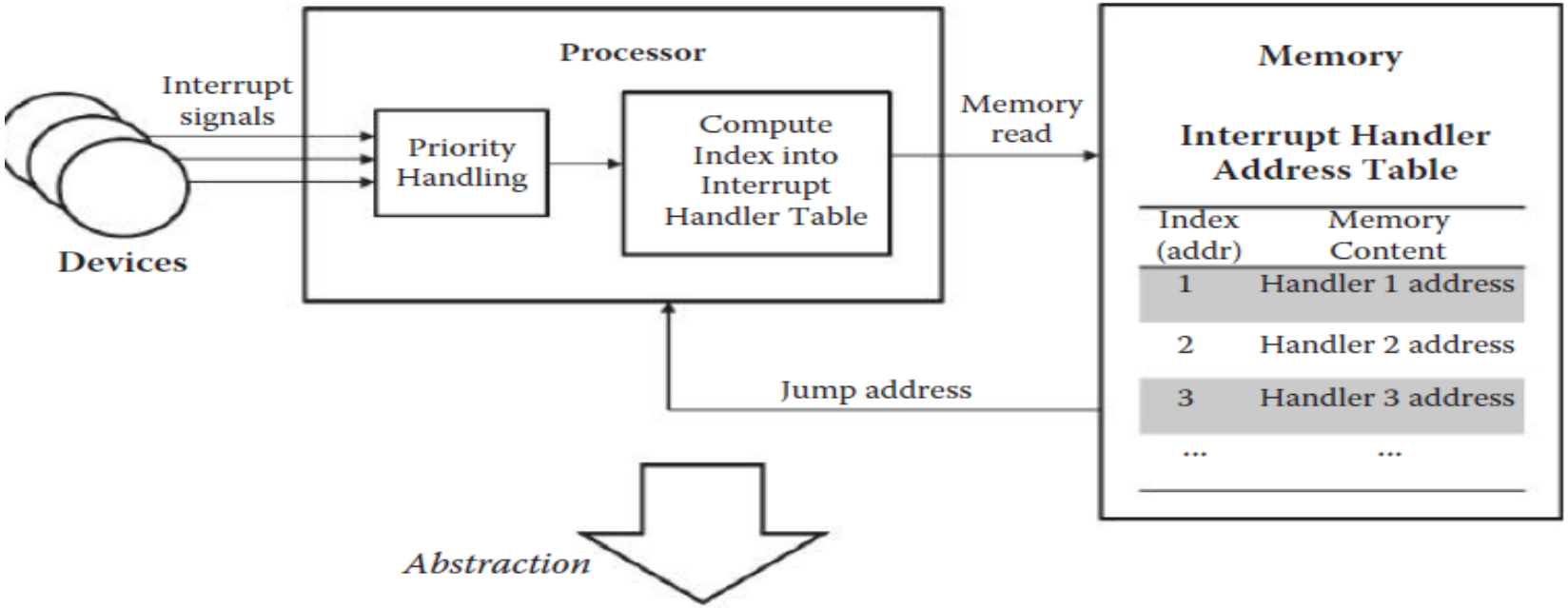
- In practice, this means that the proces- sor is always listening to the incoming events, ready to serve them as needed. The interrupt mechanism is often contrasted to *polling* (also
- known as *busy waiting*). In polling, the processor (rather than the I/O
- device) initiates input or output. In order to carry out I/O tasks, the processor enters a loop, continually checking the I/O device status to see whether it is ready, and incrementally accomplishes the I/O task. This form of an I/O is deficient in supporting asynchronous user- driven (anytime) I/O and wastes CPU time by blocking other non-I/O processes to move on.

- At a higher level, the I/O operation is often described in terms of *events* and *event handlers*, which is in fact an abstraction of the lower- level interrupt mechanism. This is generally called the *event-driven architecture* in which programs are developed in terms of events (such as mouse clicks, gestures, keyboard input, etc.) and their correspond-
- ing handlers. Such information can be captured in the form of a table and used for efficient execution. Figure 5.2 shows the rather compli- cated interrupt mechanism abstracted into the form of a simple event-handler table.

3. Processing the Input and Generating Output

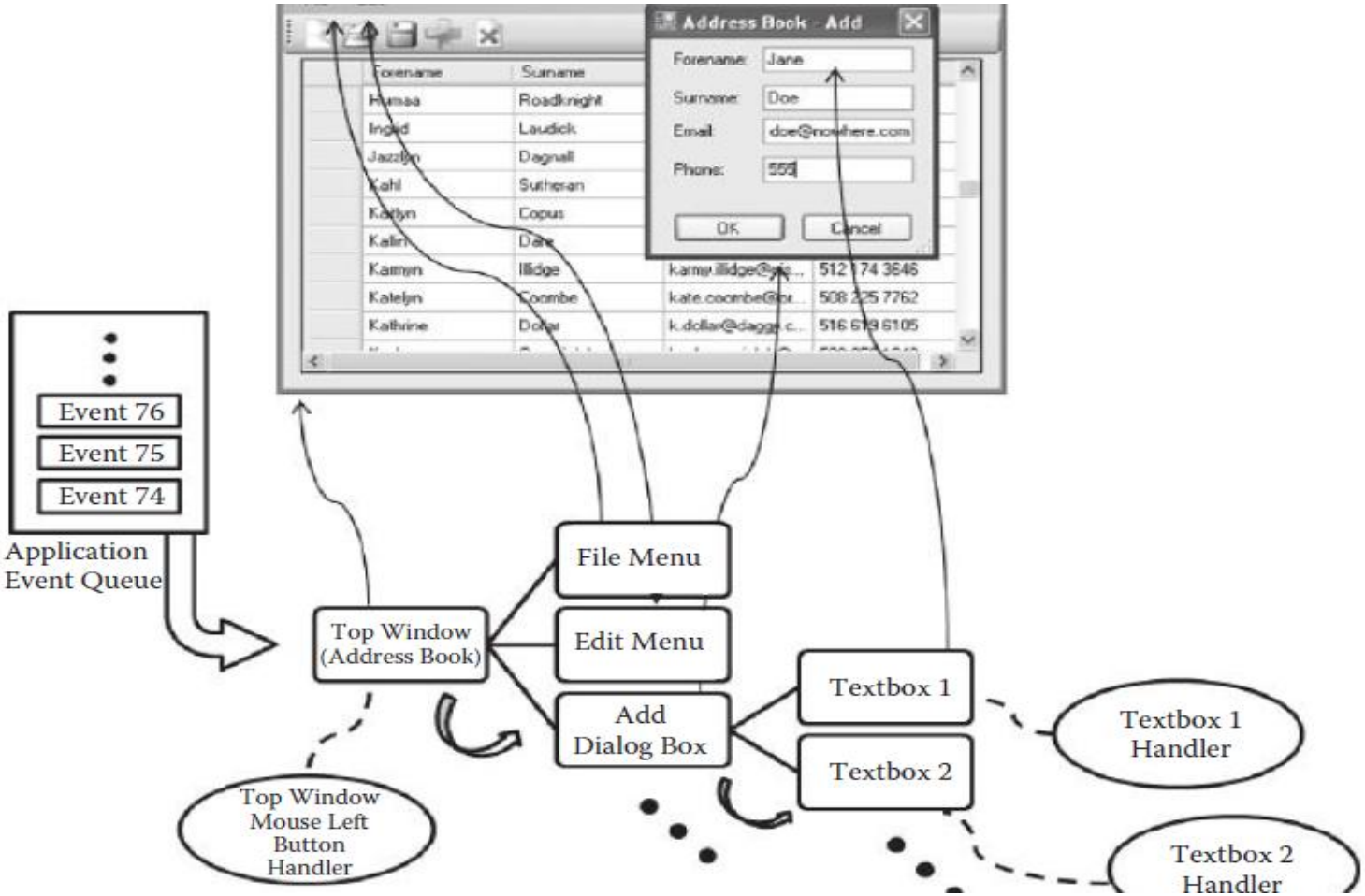
- *Events, UI Objects, and Event Handlers*
- *Event-Driven Program Structure*
- outputs

Events, UI Objects, and Event Handlers

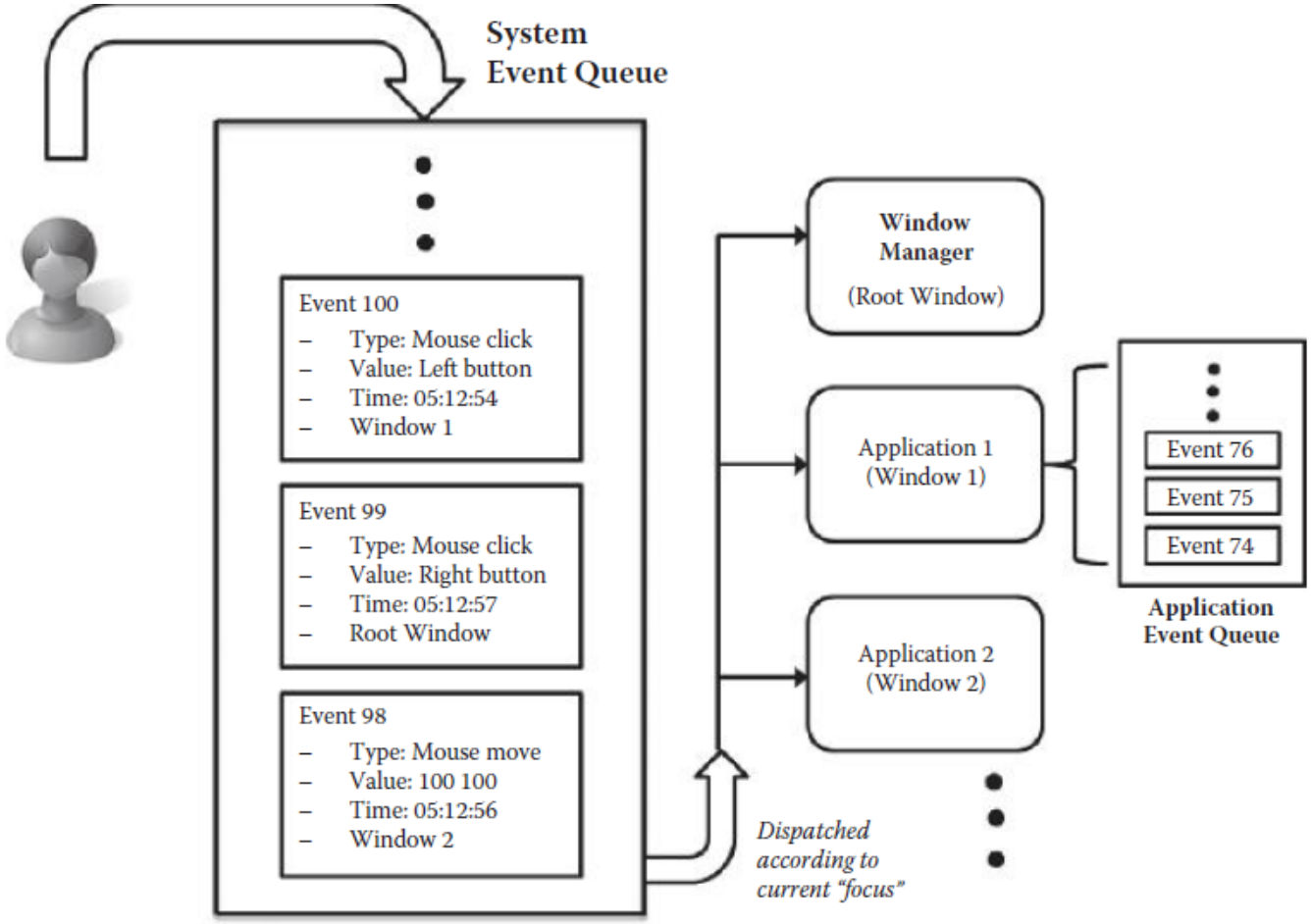


Event Id	Handler Table
1	Handler 1 address
2	Handler 2 address

Event being dispatched to the right UI object handler for a given application (organized as a set of UI objects and associated event handlers in a hierarchical manner) from the application-event queue.



Event queuing at the top application level.



Event-Driven Program Structure

- The first initialization part of the program creates the necessary UI objects for the application and declares the event-handler functions and procedures for the created UI objects.
- Then the program enters a loop that automatically keeps removing an event from the application event queue and invoking the corresponding handler (i.e., dispatching the event).

1. initialize application

define UI objects

define event handlers

...

...

2. run "system_loop"

```
system_loop ( ) {
 while (event != QUIT) {
  get next event from application queue
  find the target_object by traversing down the UI object tree
  invoke corresponding event handler for this target_object
  refresh screen and send commands to aural/haptic device
  }
} /* system provided event processing loop */
```

Output

- Interactive behavior that is purely computational will simply be carried out by executing the event-handler procedure
- A separate step for refreshing the display based on the changed screen data is called as the last part of the event processing loop. Analogous processes will be called for sending commands to output devices of other modalities as well