Putra Pandu Adikara, S.Kom

Interaksi Manusia dan Komputer

UI Architecture & Implementation

Tujuan

- Memahami lapisan alat pengembangan: Windowing system, Toolkit, dan UIMS.
- Memahami konseptual dan implementasi pemisahan pada UIMS
- Memahami dan menggunakan Widget
- Mengimplementasikan antarmuka yang baik sesuai teknik, metode, standar, garis pedoman.

Pendahuluan

- Bagaimana interaksi manusia dan komputer mempengaruhi programmer?
  - Dengan adanya kemajuan di coding, maka ikut menaikkan level pemrograman.
  - Perangkat keras tertentu juga berimbas pada teknik interaksi tertentu pula.

Pendahuluan

- Terdapat tiga lapisan pada alat pengembangan, antara lain:
  - **Sistem Windowing (Windowing system)**
    - dukungan inti untuk aktifitas pengguna-sistem secara terpisah atau bersamaan.
  - **Alat bantu interaksi (Interaction toolkits)**
    - membawa pemrograman mendekati ke persepsi pengguna.
  - **Sistem manajemen antarmuka pengguna (User Interface Management Systems/UIMS)**
    - mengkontrol hubungan antara presentasi dan fungsionalitas

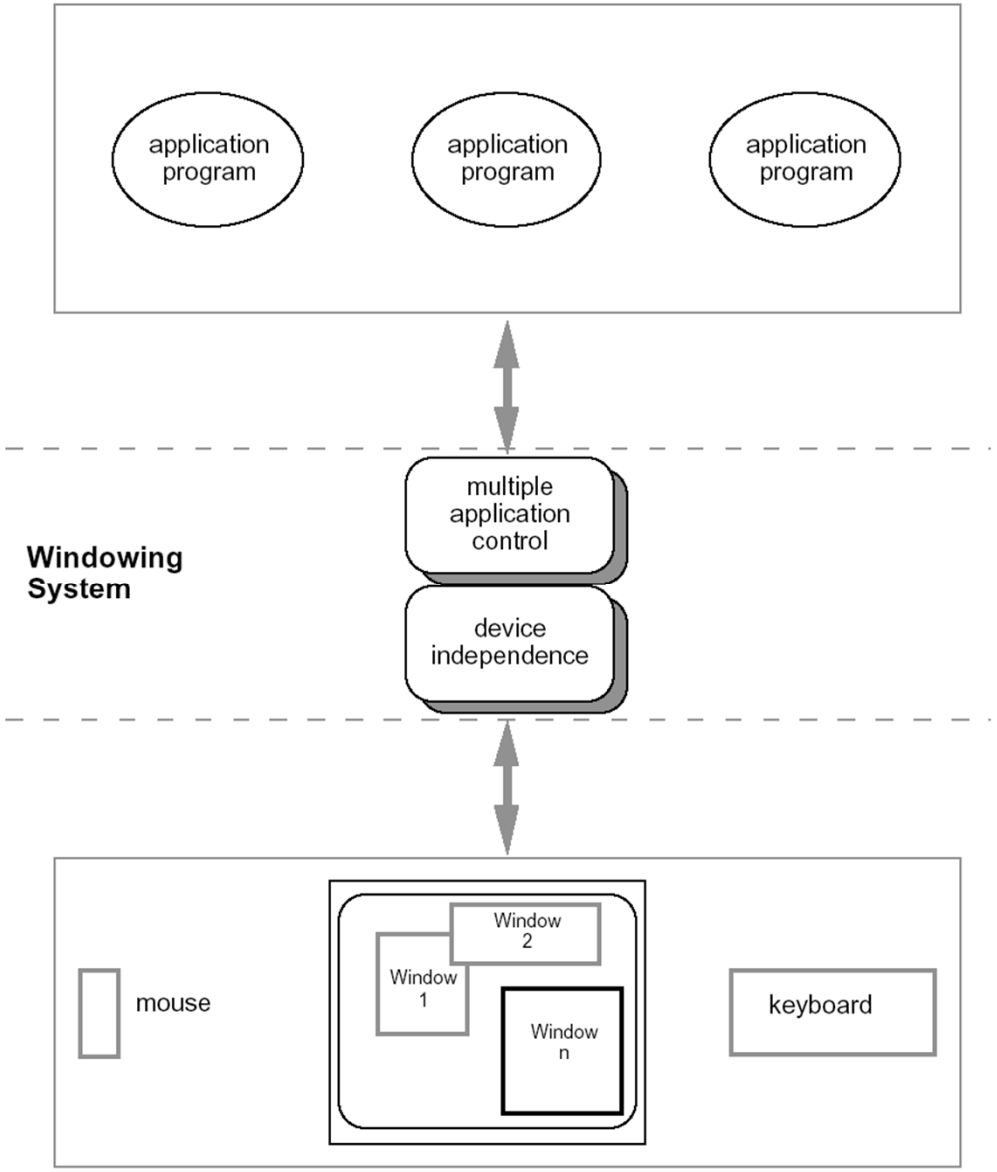
Sistem Windowing

- Pusat lingkungan untuk programmer dan penggguna dari sistem interaktif yang memungkinkan satu workstation tunggal mendukung sistem-pengguna dengan aksi-aksi secara bersamaan.
- Sistem Windowing mengkordinasikan berbagai macam perangkat input-output.

Elemen-elemen Sistem Windowing

- Kemandirian perlengkapan (device independence)
  - Pemrograman driver alat terminal abstrak.
  - Input dan output untuk model image/graphic:
    - Pixel
    - PostScript (MacOS X, NextStep)
    - Graphical Kernel System (GKS)
    - Programmers' Hierarchical Interface to Graphics (PHIGS)
- Berbagi-pakai sumber daya
  - Mencapai tugas dari pengguna secara bersamaan.
  - Sistem jendela mendukung proses mandiri.
  - Isolasi dari aplikasi individual.

Peranan Sistem Windowing



- Terkadang diperlukan bekerja melalui low-level pada windowing system, misalnya pada platform baru, contohnya pada loop di dalam main program.

```
main() {
   InitializeSystem();
   SetInitialState();
   DisplayInitialGraphics();
   while(true) {
        Event event = readNextEvent();
        switch(event.type) {
        case EVENT_REDISPLAY: redisplay(); break;
        case EVENT_PEN_DOWN: doPenDown(event.x, event.y); break;
        case EVENT_CHAR: doInputChar(event.detail); break;
        …
        default: doSystemDefault(event); break;
        }
   }
}
```

Arsitektur dari Sistem Windowing

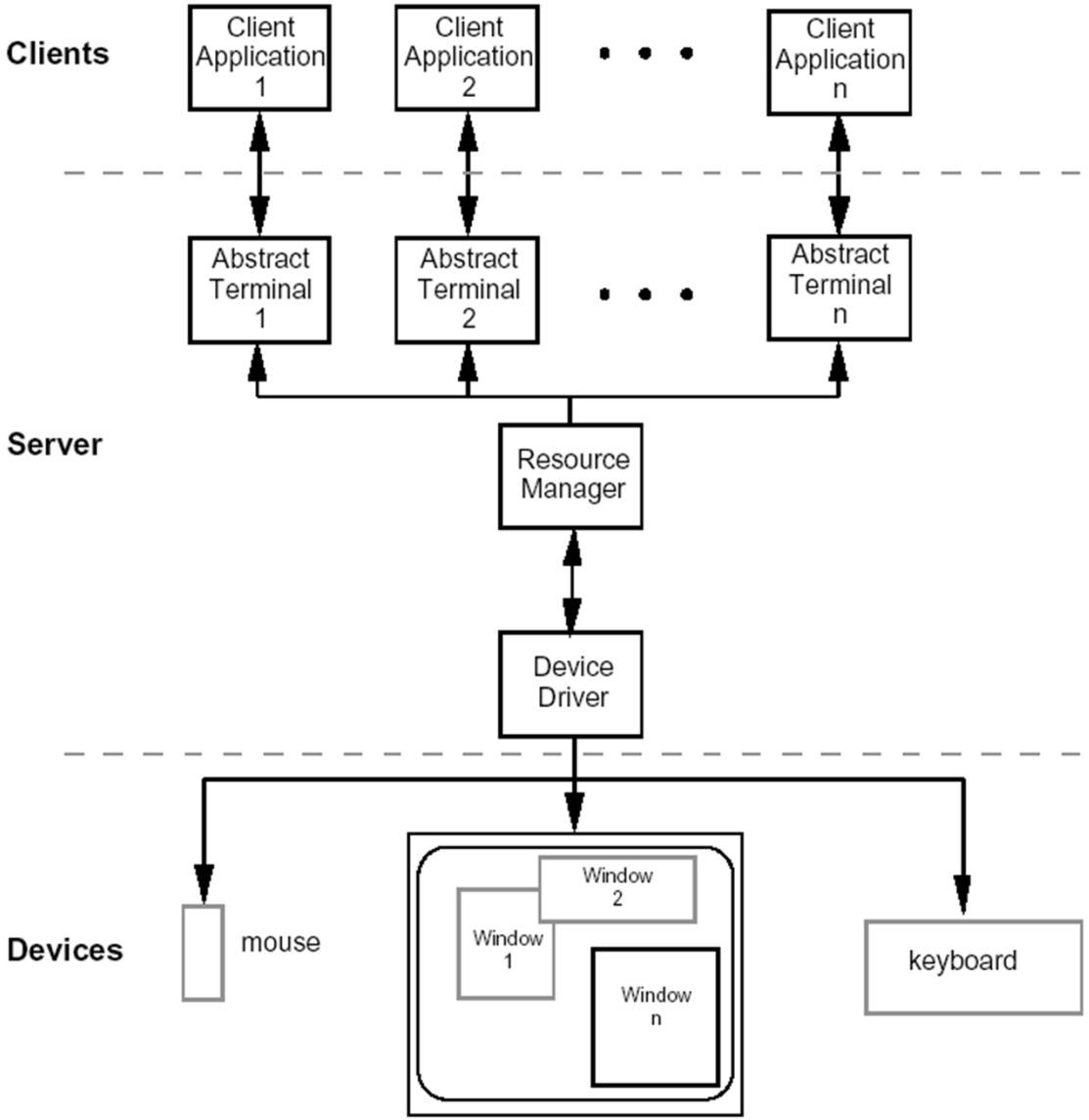
Ada tiga kemungkinan arsitektur perangkat lunak dimana diasumsikan seluruh driver alat tersebut terpisah dan berbeda pada bagaimana manajemen banyak aplikasi diimplementasikan.

- Tiap aplikasi mengatur seluruh proses
  - Tiap orang mengkhawatirkan tentang sinkronisasi
  - Mengurangi portabilitas dari aplikasi

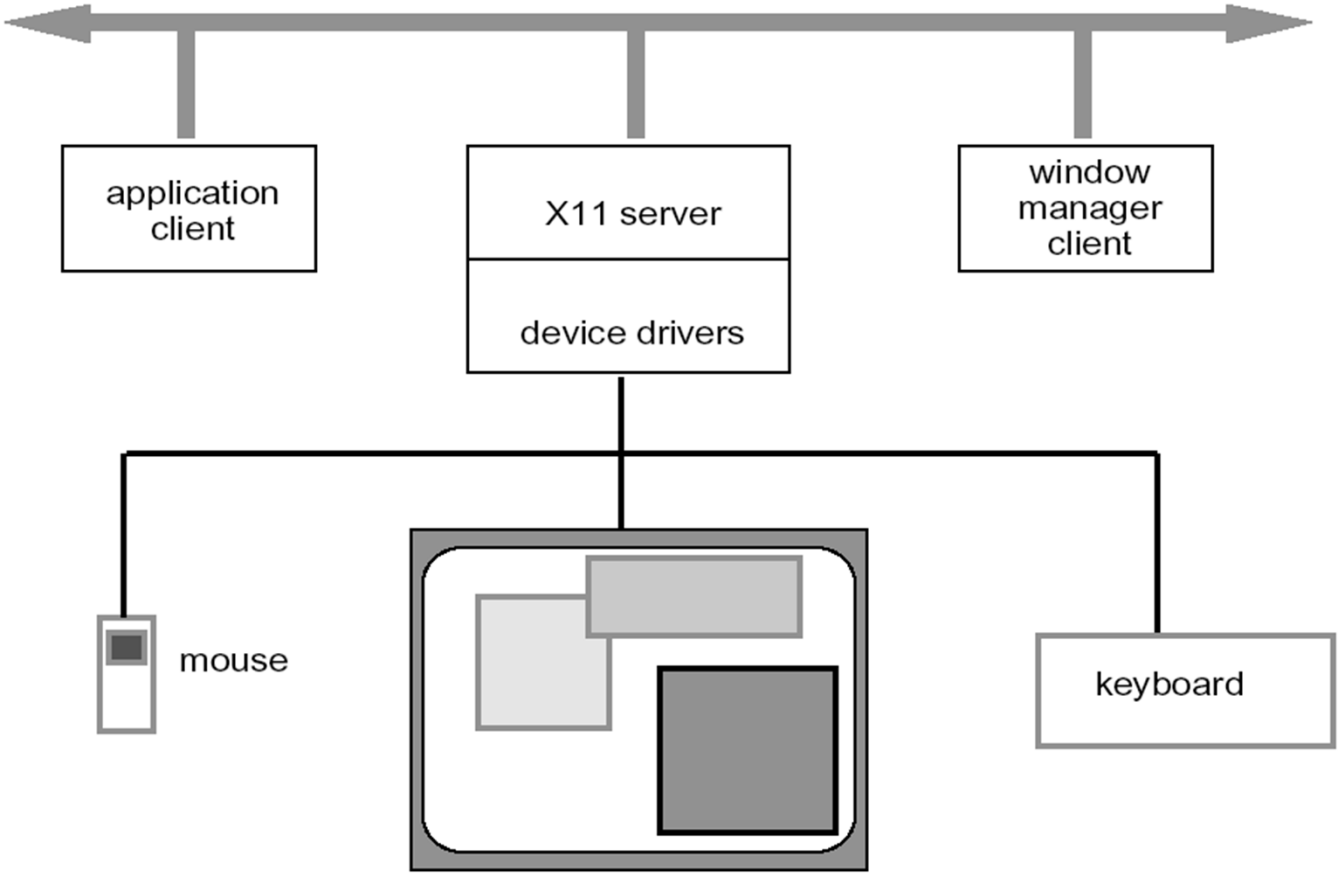
- Pembagian peran manajemen dalam kernel dari sistem operasi: aplikasi terhubung dengan sistem operasi

- Pembagian peran manajemen sebagai aplikasi terpisah: maksimum portabilitas dari suatu aplikasi

Arsitektur Client-Server



Arsitektur X-Window

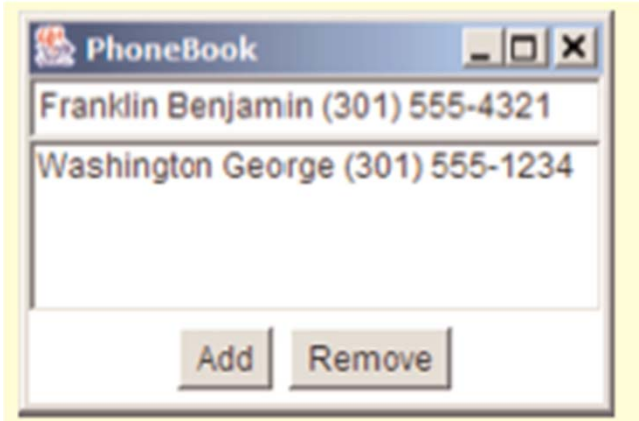


Interaction Toolkit

- Alat bantu interaksi memberikan abstraksi untuk memisahkan perangkat input dan output secara fisik.
- Alat bantu ini memungkinkan programmer untuk membuat perilaku object pada tingkat yang sama dengan bagaimana user merasakannya.
- Pada lapisan alat bantu antarmuka pengguna grafis, contohnya:
  - Widget Toolkit, seperti windows, scrollbars, pull-down atau popup menu, dll.

Contoh GUI Toolkit pada Java

```
class PhoneBook {
  public static void main(String[] args) {
    Frame frame = new Frame("PhoneBook");     // create the main window
    final TextField entry = new TextField(); // create a text field
    frame.add(entry, BorderLayout.NORTH);     // add the text field on the upper
                                              // side of the window
    Panel panel = new Panel();                // create a composite panel to store
                                              // the two "Add" and "Remove" buttons
    panel.setLayout(new FlowLayout());        // specify how the widgets will be
                                              // laid out in the panel
    Button add = new Button("Add");           // create the "Add" button
    panel.add(add);                           // add it to the panel
    Button remove = new Button("Remove");     // create the "Remove" button
    panel.add(remove);                        // add it to the panel
    frame.add(panel, BorderLayout.SOUTH);     // add the panel on the lower side
    final List list = new List();             // create a new item list widget
    frame.add(list, BorderLayout.CENTER);     // add it to the  window's center
    list.add("Washington George
      (301) 555-1234");                       // add one item to the list
    frame.pack();                             // compute the final widgets' sizes
                                              // and positions
    frame.setVisible(true);                   // show the window
    add.addActionListener(new
      ActionListener() {                      // add action to the "Add" button
public void actionPerformed(ActionEvent e)
{ list.add(entry.getText()); }                // add the text field's text
      });                                     // to the list
    remove.addActionListener
    (new ActionListener() {                   // add action to the "Remove" button
    public void actionPerformed
    (ActionEvent e)
{ try { list.remove(entry.getText()); } catch(Exception ex) {}}
      });
  }
}
```



Widget Toolkit

- Alat bantu antarmuka pengguna mengkombinasikan antarmuka objek dan kelakuan manajemen.
- Umumnya berbasiskan object oriented.
- Banyak library toolkit (widget library) dari komponen sistem dan routine yang dimasukkan oleceh programmer untuk memudahkan pembuatan user inter

Widget Toolkit

- **Berdasarkan platform**
- **X-Windows:** X-Toolkit & X-Motif
- **Macintosh:** MacToolbox/Carbon, MacApp, Cocoa
- **Windows:** Microsoft Foundation Classes, Windows Forms, Windows Presentation Foundation (WPF), Visual Component Library (VCL)
- **Cross Platform:** GTK+, Qt, Tk, WxWidgets
- **Java:** Swing, Abstract Window Toolkit (AWT)

Widget Toolkit

- Berdasarkan bahasa
- **XML, AJAX, SVG**: jQuery, script.aculo.us, Dojo Toolkit, Yahoo! UI Library, Google Web Toolkit, XAML
- **Java:** AWT, SWT, Swing, Qt Jambi
- **C/C++:** FOX Toolkit, GTK+, Qt, Wt, Tk, WxWidgets, Xforms
- **Object Pascal:** VCL, CLX IP Pascal, Lazarus, fpGUI
- **Python:** Pyjamas, PyQt, wxPython, PyGUI, PySide, Tkinter
- **Objective C:** GNUstep

UIMS

- **Sistem Manajemen Antarmuka Pengguna (*User Interface Managemen System/UIMS*)** adalah level akhir dari alat pendukung pemrograman.
- Memungkinkan desainer dan programmer mengkontrol hubungan antara obyek presentasi dari toolkit dengan fungsi semantiknya dalam aplikasi sebenarnya.
- UIMS menambahkan level di atas toolkit karena toolkit terlalu sulit untuk bagi yang bukan programmer.
- Dimungkinkan menggunakan UI Development System (UIDS), UI Builder, atau UI Development Environment (UIDE) sebagai arsitektur konseptual teknik implementasi dan dukungan
  - seperti Netbeans, Eclipse, Visual Studio, Delphi, dll.

Fungsi UIMS

- Dengan melakukan pemisahan antara semantik aplikasi dan presentasi yang disebut *software architectural pattern*, akan meningkatkan:
- **Portabilitas**: menjalankan pada sistem berbeda
- **Penggunaan kembali:** komponen yang digunakan kembali dengan memotong biaya dari tingkat kegunaan
- **Beberapa antarmuka:** memiliki fungsi yang sama dalam masalah akses .
- **Kustomisabilitas:** oleh desainer dan pengguna
- **Mengidentifikasi peranan:** contohnya Seeheim
- **Komponen presentasi**
- **Kontrol dialog**
- **Model antarmuka aplikasi**

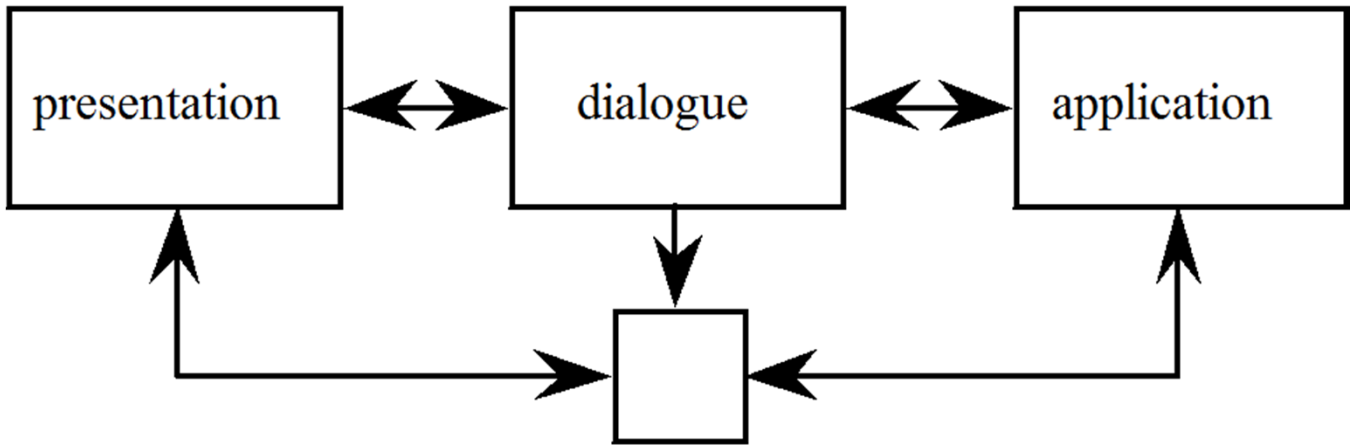
UIMS

- Pemisahan ini biasanya karena memperhatikan hal berikut ini

Aplikasi	Antarmuka
• **Fungsionalitas inti**	• **Komponen antarmuka**
• **Operasi**	• **Graphic**
• **Data**	• **I/O**

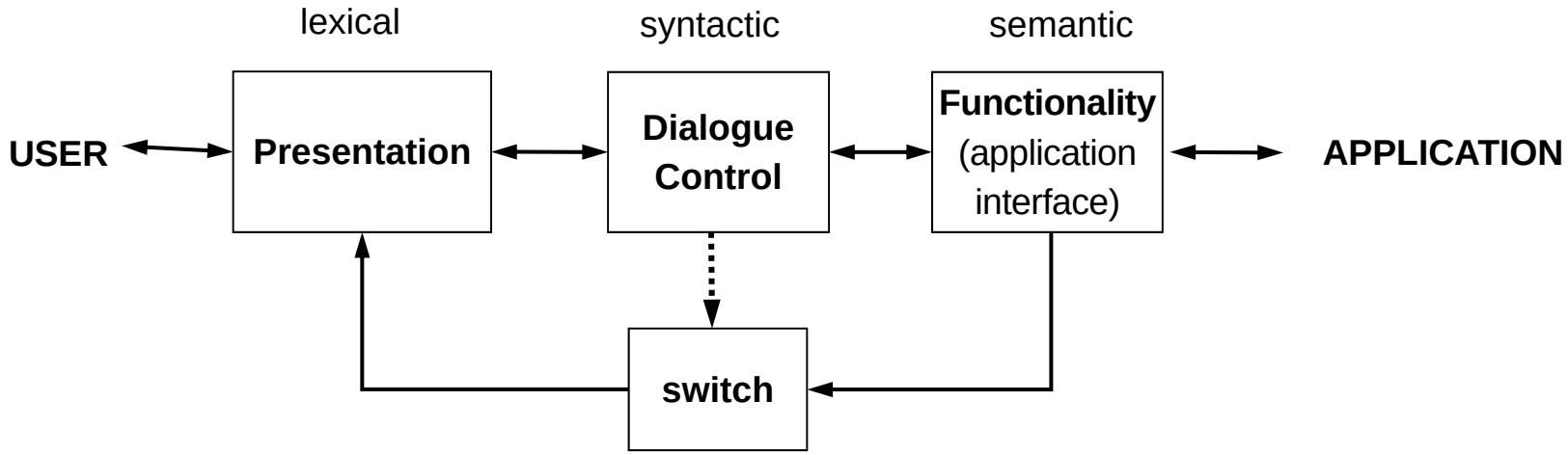
UIMS Model

- Tradisi UIMS, yaitu layer interface atau komponen logika seperti:
- **Linguistik**: leksikal, sintaktik, semantik.
- **Seeheim**



Model konseptual Seeheim

UIMS Model

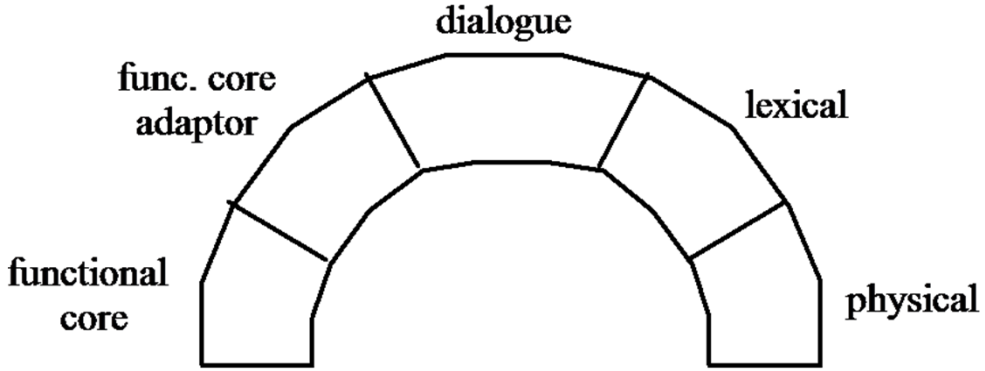


Model implementasi Seeheim

- Terdapat beberapa macam umpan balik semantik yaitu:
  - Lexical, misalnya gerakan mouse
  - Syntactic, misalnya *highlight* suatu menu
  - Semantic, misalnya sejumlah perubahan setelah klik mouse
- Adanya umpan balik semantik cepat menyebabkan pada implementasi dibutuhkan kontrol dialog (*switch box*) yang mengatur komunikasi langsung pada aplikasi dan presentasi.

UIMS Model

❖ Arch/Slinky



- Lebih banyak lapisan pada model ini yang membedakan leksikal dan fisik.
- Lapisan yang berbeda dapat lebih tebal/banyak (lebih penting) pada sistem yang berbeda atau komponen yang berbeda.

UIMS Model

Model-View-Controller (MVC)

- Pada model design-pattern MVC, hubungan antara aplikasi semantik dan presentasi dibangun dari tiga serangkai (*triad*) Model-View-Controller
- Memisahkan antara model domain, presentasi, dan aksi dari pengguna.
  - **Model** menangani keadaan internal logis dari komponen, mengelola perilaku dan data dari domain aplikasi, menanggapi request informasi mengenai keadaan-nya (biasanya dari view), dan menanggapi instruksi untuk mengubah keadaan (biasanya dari controller), misalnya koneksi database, query database, implementasi business rule.
  - **View** menangani bagaimana direnderi di layar.
  - **Controller** memproses input pengguna.

Diagram MVC

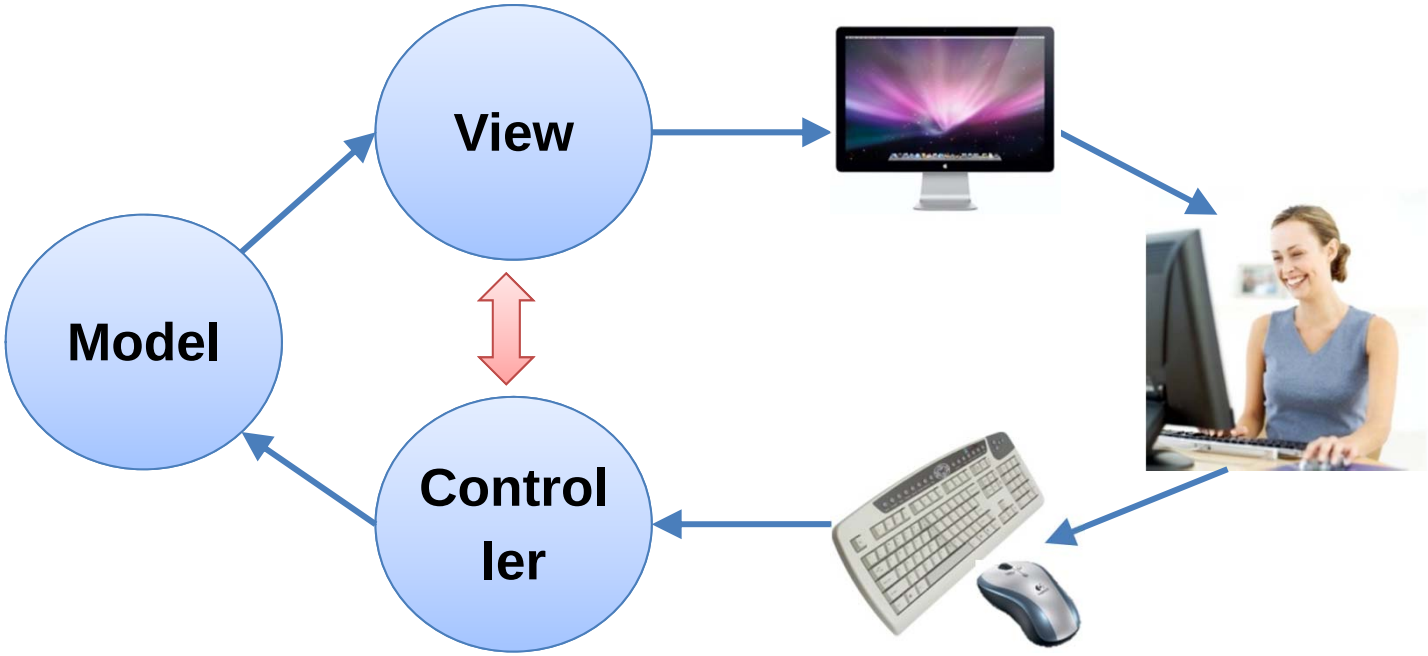
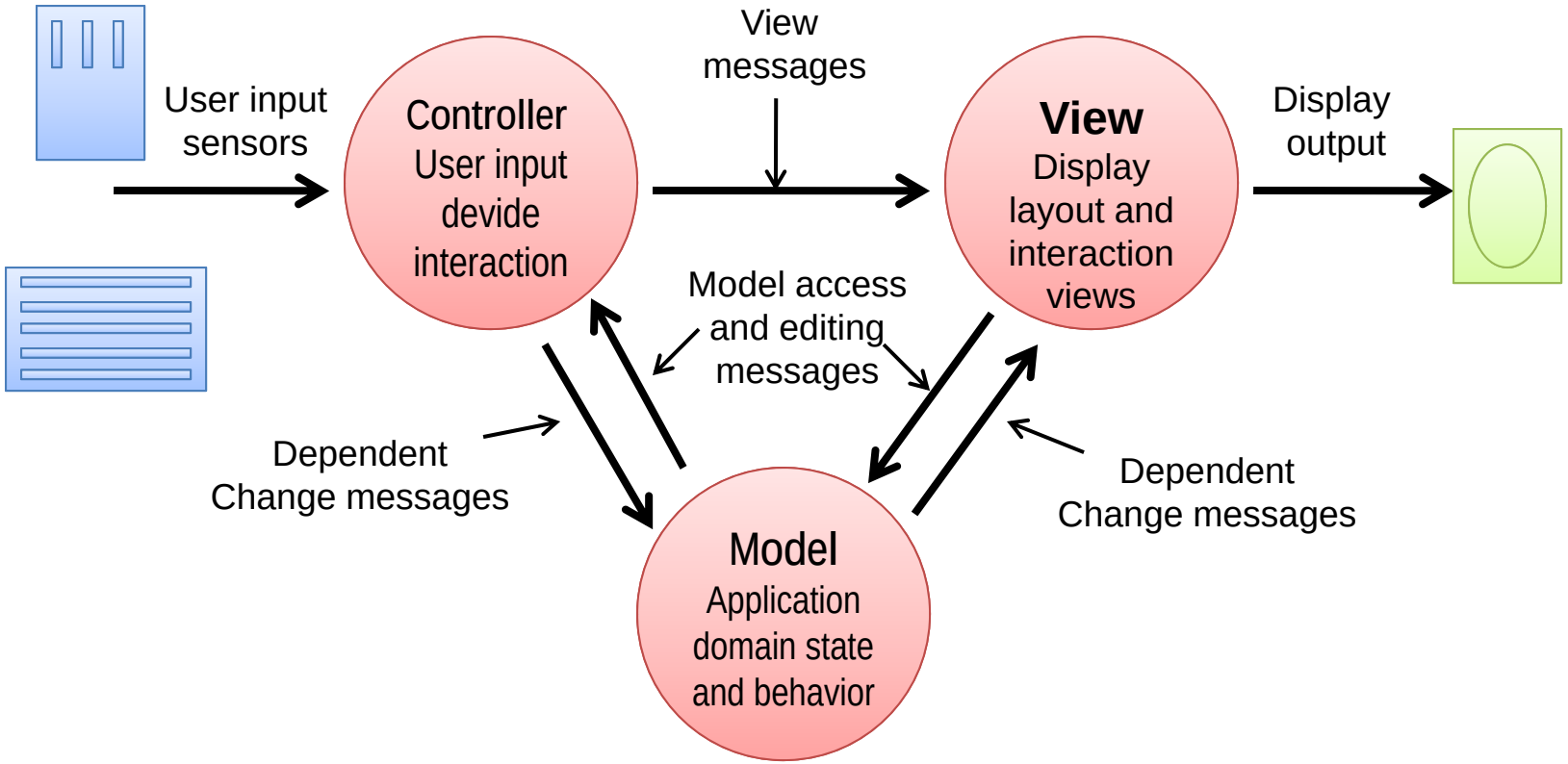


Diagram MVC


User input
sensors
Controller
User input
devide
interaction
View
messages
View
Display
layout and
interaction
views
Display
output
Model access
and editing
messages
Dependent
Change messages
Dependent
Change messages
Model
Application
domain state
and behavior

Contoh Architectural Pattern MVC

- **Web based framework**
  - **Pada aplikasi JSP:**
    - Model: JavaBean yang terkoneksi ke Data Sources
    - Controller: Servlet, yang menangani request
    - View: halaman JSP, yang menangangi response
  - **Pada aplikasi PHP:** framework Code Igniter, terdapat kelas-kelas dan folder terstruktur untuk MVC.
    - Model: berupa file untuk koneksi dan operasi pada Data Sources.
    - View: berupa file HTML.
    - Controller: berupa file berisi fungsi atau prosedur.
  - Contoh lain:
    - PureMVC framework untuk ActionScript, Wt untuk C++, Oracle Application Framework,Kohana PHP , CakePHP, Yii PHP, Zend Framework, Drupal, Django, Ruby on Rails, PureMVC untuk ColdFusion/Java/Flex/PHP/Ruby/ Python.

Contoh Architectural Pattern MVC

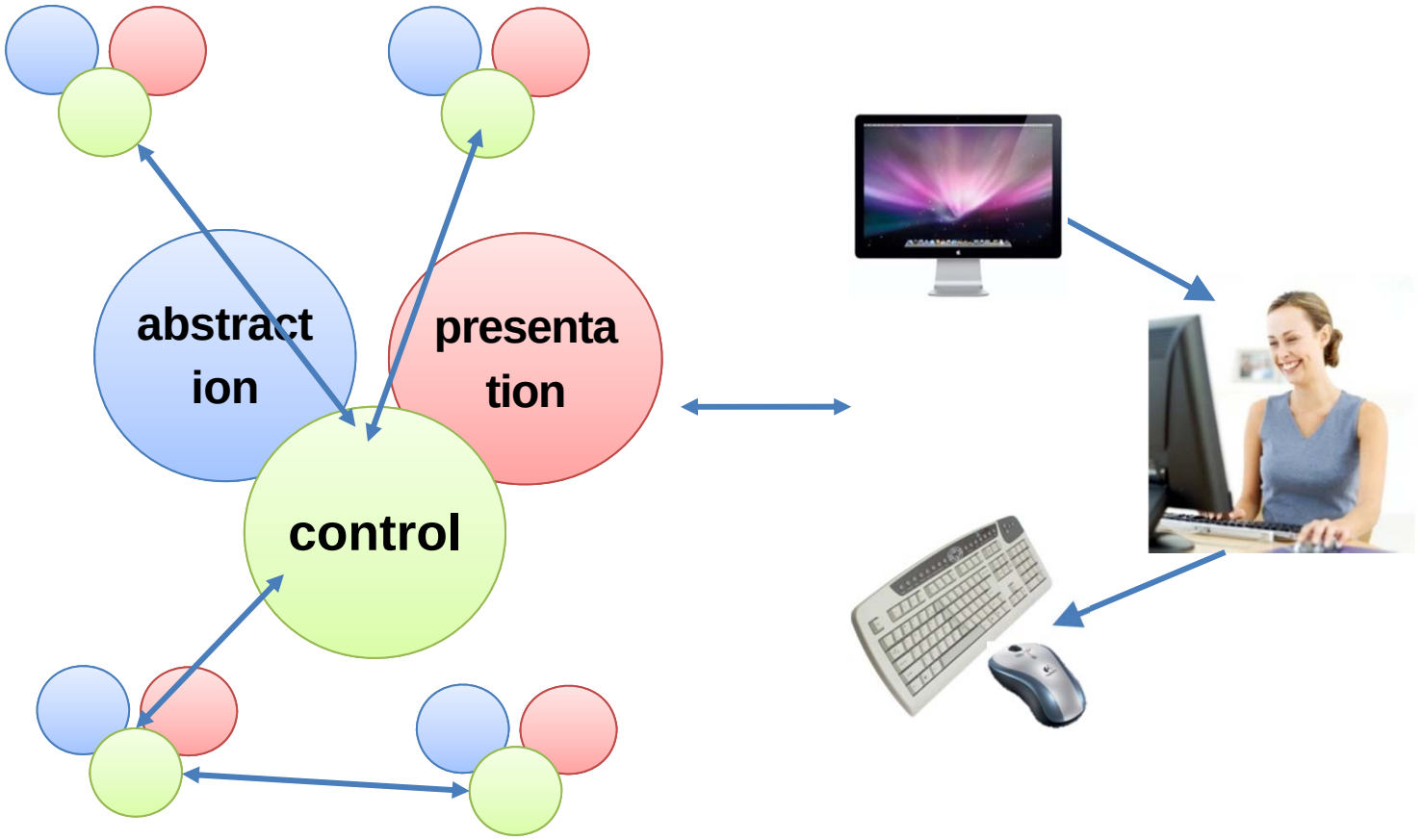
- **Desktop GUI framework**
  - **Pada aplikasi .NET (**Windows Forms, Windows Presentation Foundation/WPF)
    - Model: berupa kode implementasi untuk koneksi database, query database, atau yang menghasilkan data seperti DataSet, dll.
    - Controller: berupa kode implementasi di dalam file .CS atau .VB.
    - View: dapat berupa Windows Forms, WebForms (.ASPX/.ASCX), HTML, XML/XLST, XTML, WML
  - **Pada Java:** Java Swing, AWT
  - Contoh lain
    - GTK+, JFace, XPages, Qt, Microsoft Foundation Class library (MFC), Java Swing,Adobe Flex, Wavemaker,.

Contoh Architectural Pattern MVC

- Beberapa design pattern menggunakan istilah berbeda walau berdasarkan dari MVC
  - Misalnya PureMVC
    - Data object diatur oleh **Proxies**
    - View component yg membuat UI diatur oleh **Mediators**
    - **Command** yang dapat berinteraksi ke Proxies, Mediators, serta Command lainnya

UIMS Model

- **Presentation-Abstraction-Control (PAC)**



UIMS Model

- **Presentation-Abstraction-Control (PAC)**
  - PAC mirip dengan MVC. PAC digunakan sebagai struktur hirarki agen
  - Tiap agen terdiri dari tiga serangkai (triad): presentasi, abstraksi dan bagian kontrol.
  - Agen (atau tiga serangkai) berkomunikasi satu sama lain hanya melalui bagian kontrol dari masing-masing triad.
  - Ini juga yang membedakan dari MVC, dimana di dalam masing-masing triad, sepenuhnya memisahkan presentasi (view dalam MVC) dan abstraksi (model dalam MVC).
  - Cara ini menyediakan pilihan untuk multithread secara terpisah antara model dan view yang dapat memberikan pengalaman pengguna atas program yang cepat waktu mulainya, karena user interface (presentasi) dapat ditampilkan sebelum abstraksi sepenuhnya diinisialisasi.

UIMS Model

- Bagian atau variasi dari PAC dengan nama **Hierarchical-Model-View-Controller (HMVC)** dipublisasikan pada sebuah artikel JavaWorld Magazine, penulis sepertinya tidak menyadari adanya PAC yang dipublikasikan 13 years sebelumnya.
- Perbedaan utama pada HMVC dan PAC adalah HMVC kurang ketat karena membolehkan view dan model tiap agen berkomunikasi secara langsung, tanpa melalui controller

- Variasi model UIMS lain
  - Model View ViewModel (MVVM) → WPF, Silverlight, dll
  - Model View Presenter (MVP)
  - Model View Adapter (MVA)

Pengembangan Antarmuka

- Menemukan alat yang tepat adalah tradeoff antara enam kriteria utama:
  - Bagian dari aplikasi yang dibangun menggunakan perangkat tersebut.
  - Waktu pembelajaran
  - Waktu pembangunan/pengembangan
  - Metodologi yang dikenakan atau disarankan
  - Komunikasi dengan subsistem lain
  - Ekstensibilitas dan modularitas

Pemilihan Alat Bantu Pengembangan Antarmuka

Software Layers		Visual Tools	Contoh
4	Application	Model Based Building Tools	Microsoft Access Sybase PowerDesigner
3	Application Framework Specialized Language	Conceptual Building Tools	Macromedia Director, Tcl/Tk, Microsoft MFC
2	GUI Toolkit	Interface Builder	Borland JBuilder Microsoft Visual Studio
1	Windowing System	Resources Editor	Windows Graphical User Interface Apple Quartz X11 Windowing System

Pemilihan Alat Bantu Pengembangan Antarmuka

Layers	Part of the application built	Learning Time	Building Time	Methodology Imposed or Advised	Communication with other subsystems	Extensibility and modularity
4	All for a specific domain	Long	Short	Specification first, then visual, then programming (if required)	Very good for the specific domain of the tool	Very good
3	Presentation, interaction	Short (days)	Short	Visual first	Depends on the tool	Languages: Bad Framework: Good
2	Presentation	Long (weeks)	Long	Visual first with tools, none otherwise	Good	Medium/Good
1	All	Very long (Months)	Very long	None	Very good	Very bad

MVC Pattern

Model-View-Controller Pattern

- MVC memisahkan urusan frontend dan backend
- Memisahkan input dari output
- Manfaat:
  - Mengizinkan beberapa view dalam data aplikasi yg sama
  - Mengizinkan view/controller digunakan ulang untuk model yg lain
- Contoh: text box (widget)
  - Model: string yang bisa diubah-ubah
  - View: kotak dengan teks yg ‘digambar’ didalamnya
  - Controller: keystroke handler

Model

- Bertanggung jawab terhadap data
  - Mengelola application state (data fields)
  - Implementasi perlakuan state-changing
  - Memberitahukan view/controller yang sesuai ketika terjadi perubahan (*observer pattern*)

- Design issues
  - How fine-grained are the change descriptions?
    - “The string has changed somehow” vs. “Insertion between offsets 3 and 5”
  - How fine-grained are the observable parts?
    - Entire string vs. only the part visible in a view

View

- Bertanggung jawab terhadap output
  - Menempati luasan layar (posisi, ukuran)
  - Ditampilkan di layar
  - Memperhatikan perubahan terhadap model
  - Menggambarkan hasil dari model

- Satu view mempunyai satu model
  - Tapi model bisa memiliki banyak view

View Hierarchy

- View disusun menjadi sebuah hirarki
- Container
  - Window, panel, rich text widget
- Component
  - Canvas, button, label, textbox
  - Containers juga termasuk components
- Tiap sistem GUI mempunyai hirarki view, dan hirarki view digunakan dalam banyak cara
  - Output
  - Input
  - Layout

View Hierarchy: Output

- Drawing
  - Request penggambaran dijalankan dari atas ke bawah sesuai hirarkinya
- Clipping
  - Parent container mencegah child component digambarkan diluar batas luasannya
- Z-order
  - Children (biasanya) digambarkan di atas parent-nya
  - Child order menentukan urutan penggambaran antar sesama sibling
- Coordinate system
  - Setiap container mempunyai sistem kordinat sendiri (awalannya biasanya dari atas kiri)
  - Posisi child dinyatakan sesuai koordinat dari parent

View Hierarchy: Input

- Event dispatch and propagation
  - Event input mentah (penekanan tombol, gerakan mouse, klik mouse) dikirimkan ke komponen terendah dlm hirarki
  - Event disalurkan ke atas di dalam hirarki hingga komponen-komponenn di dalamnya menanganinya
- Keyboard focus
  - Satu komponen di dalam hirarki mempunyai fokus (secara implisit, ancestornya juga)

View Hierarchy: Layout

- Automatic layout: children diposisikan dan diatur ukurannya oleh parent
  - Memungkinkan perubahan ukuran window
  - Mampu berurusan dgn internationalization dan perbedaan platform (e.g. fonts or widget sizes)
  - Mengurangi beban programmer dalam mengelola ukuran dan posisi
    - Although actually just raises the level of abstraction, because you still want to get the graphic design (alignment & spacing) right

Controller

- Bertanggung jawab terhadap input
  - Memperhatikan adanya event pada mouse dan keyboard
  - Menginstruksikan model atau view untuk berubah yg berkesesuaian
    - e.g., character is inserted into the text string

- Controller mempunyai satu model dan satu view

Problem: Controller Needs Output

- Menu jelas terkait dengan suatu controller
  - e.g. menu klik kanan pada text-field

- Tapi sebuah menu perlu di ‘gambar’kan
  - Menu adalah sebuah model-view-controller itu sendiri, digunakan sebagai subkomponen

Problem: Who Manages Selection?

- Harus ditampilkan oleh view
  - As blinking text cursor or highlighted object
- Harus diperbaharui dan digunakan oleh controller
  - Clicking or arrow keys change selection
  - Commands modify the model parts that are selected
- Apa sebaiknya selection ada di model?
  - Generally not
  - Some views need independent selections (e.g. two windows on the same document)
- View yang lain perlu di sinkronisasi selection-nya
  - e.g. table view & chart view

Problem: Direct Manipulation

- Direct manipulation: user menunjuk objek yang ditampilkan dan bisa memanipulasinya secara langsung
- View harus menyediakan **affordance** untuk controller
  - e.g. scrollbar thumb, selection handles
- View juga harus menyediakan feedback tentang status controller
  - e.g., button is depressed

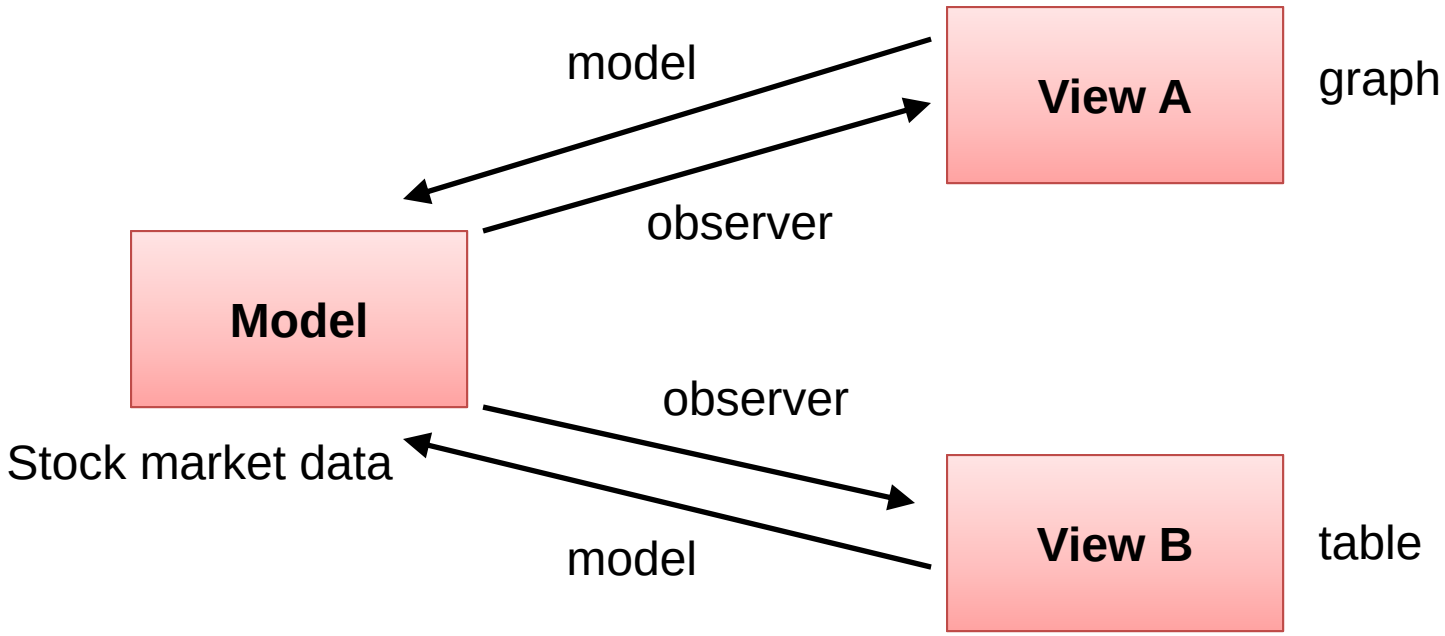
Reality: Tightly Coupled View & Controller

- MVC sebagian besar "*tergantikan*" oleh MV (Model-View)
- View yang dapat dipergunakan ulang untuk input dan output
  - Disebut juga widget atau component
- Controller yang dipergunakan ulang biasanya jarang
  - Actions in Java Swing: objects that sit behind menu item, toolbar button, or keyboard shortcut
  - E.g. cut, copy, paste, delete

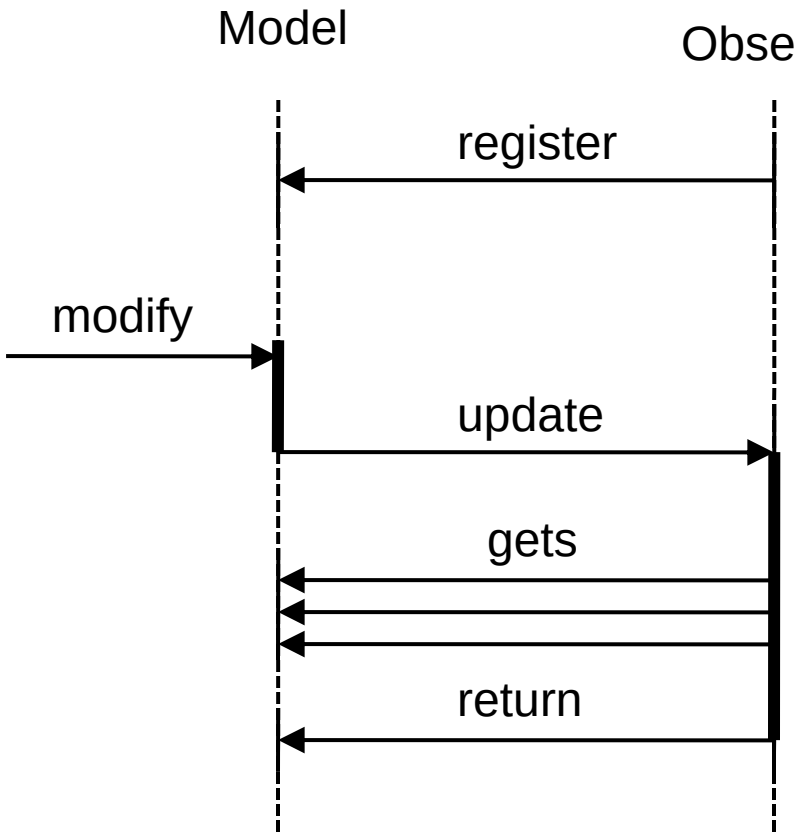
Observer Pattern

- Observer pattern (publish/subscribe pattern) digunakan untuk memisahkan model dari tampilan
- Observer adalah pola desain software di mana suatu objek **(Publisher/Subject)** mempunyai daftar objek yang mempunyai ketergantungan **(Subscriber)/Observer object)**
- Ketika ada perubahan status di Subject maka akan diberitahukan secara otomatis ke Observer (biasanya dengan memanggil suatu method) sehingga Subscriber akan menerima perubahan data seketika
- Hal ini utaman digunakan untuk mengimplementasikan sistem event handling terdistribusi.

Observer Pattern



Basic Interaction




```
Interface Model{
   void register(Observer)
   void unregister(Observer)
   Object get()
   void modify()
}

Interface Observer{
   void update(Event)
}
```

Contoh

```
/* File Name : EventSource.java */
//Alex Pantaleev
package obs;

import java.util.Observable;          //Observable is here
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class EventSource extends Observable implements Runnable {
    public void run() {
        try {
            final InputStreamReader isr = new InputStreamReader( System.in );
            final BufferedReader br = new BufferedReader( isr );
            while( true ) {
                String response = br.readLine();
                setChanged();
                notifyObservers( response );
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
/* File Name: ResponseHandler.java */

package obs;

import java.util.Observable;
import java.util.Observer;  /* this is Event Handler */

public class ResponseHandler implements Observer {
    private String resp;
    public void update (Observable obj, Object arg) {
        if (arg instanceof String) {
            resp = (String) arg;
            System.out.println("\nReceived Response: "+ resp );
        }
    }
}
```

```
/* Filename : MyApp.java */
/* This is the main program */

package obs;

public class MyApp {
    public static void main(String args[]) {
        System.out.println("Enter Text >");

        // create an event source - reads from stdin
        final EventSource evSrc = new EventSource();

        // create an observer
        final ResponseHandler respHandler = new ResponseHandler();

        // subscribe the observer to the event source
        evSrc.addObserver( respHandler );

        // starts the event thread
        Thread thread = new Thread(evSrc);
        thread.start();
    }
}
```

Design Pattern

Design Pattern

- **Creational**
  - Abstract factory
  - Builder
  - Factory method
  - Prototype
  - Singleton

Design Pattern

- **Structural**
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy

Design Pattern

- **Behavioral**
  - Chain of responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - **Observer**
  - State
  - Strategy
  - Template method
  - Visitor