



Programska arhitektura uporabniških vmesnikov

Vsebina poglavja

- vrste uporabniških vmesnikov
- strukturna razlika uporabniških vmesnikov
- vrste vhodnih dogodkov
- okenski sistem X
- Java, C
- hierarhija orodij za načrtovanje uporabniških vmesnikov
- orodja za načrtovanje uporabniških vmesnikov
- obravnava dogodkov, vrsta dogodkov, zanka dogodkov
- arhitektura model – pogled - nadzornik

Vrste uporabniških vmesnikov (UV)


- konzolno usmerjeni uporabniški vmesniki
Akcija→Objekt
 - navadni tekstovni vmesniki
 - konzolni interaktivni vmesniki
- grafični uporabniški vmesniki
Objekt→Akcija
 - preprosti dogodkovni interaktivni vmesniki (X11)
 - dogodkovni interaktivni vmesniki z grafičnim uporabniškim vmesnikom → pisanje tiste kode, ki se izvaja kot odziv na dogodek
 - GTK+, Qt, Motif: odzivne funkcije
 - Java: poslušalci

Strukturna razlika UV

- konzolni vmesniki: programer določi, kdaj bo prišlo do vnosa podatkov, izpisa rezultatov, ...
postopkovno programiranje
- grafični uporabniški vmesniki: uporabnik izvede akcijo, kadar želi, vnos podatkov, izpis rezultatov kot odziv na akcijo
dogodkovno usmerjeno programiranje

Navadni tekstovni vmesniki

- ne omogočajo neposredne interakcije
- zaporedno izvajanje kode od začetka do konca
- potek izvajanja je določen vnaprej
- možno je vplivanje preko vhodnih parametrov



```
int main(int argc,
        char *argv[]) {
    stavek_1;
    stavek_2;
    stavek_3;
    ...
    ...
    ...
    ...
    ...
    stavek_n;
    return 0;
}
```


Konzolni interaktivni vmesniki

- omogočajo neposredno interakcijo
- nelinearno izvajanje kode
- vmesnik čaka na vnos uporabnika, po vnosu nadaljuje izvajanje
- potek izvajanja je odvisen od uporabniškega vnosa
- veliko časa neizvajanja

```
int main(int argc,
          char *argv[]) {
    deklaracija;
    inicializacija;
    zanka{
        vnos uporabnika
        switch (vnos){
            case vnos1:
                ukazi_1;
                break;
            ...
            ...
        }
    }
    return 0;
}
```

Preprosti dogodkovni interaktivni vmesniki

- omogočajo neposredno interakcijo
- nelinearno izvajanje kode
- potek izvajanja je odvisen od uporabniškega vnosa
- veliko časa neizvajanja
- uporabnik lahko v vsakem trenutku dela različne stvari: pritisk na gumb, tipkanje, ...
- tem dejanjem rečemo **dogodki**
- na dogodke čaka aplikacija v zanki dogodkov, za katero poskrbi sama



```
int main(int argc,
        char *argv[]) {
    deklaracija;
    inicializacija;
    zanka{
        dogodek uporabnika
        switch (dogodek){
            case vnos1:
                ukazi_1;
                break;
            ...
            ...
        }
    }
    return 0;
}
```

Vrste vhodnih dogodkov

- uporabniške dogodke lahko ločimo v dve skupini:
 - preprosti vhodni dogodki
 - premik miške
 - pritisk ali spust gumba miške
 - pritisk ali spust tipke na tipkovnici
 - prevedeni vhodni dogodki (visokonivojski dogodki)
 - klik ali dvojni klik (dvoklik) miške
 - vstop/izstop miške v/iz komponente
 - pridobitev/izguba fokusa
 - odtipkan znak

Ozenski sistem X (X11)

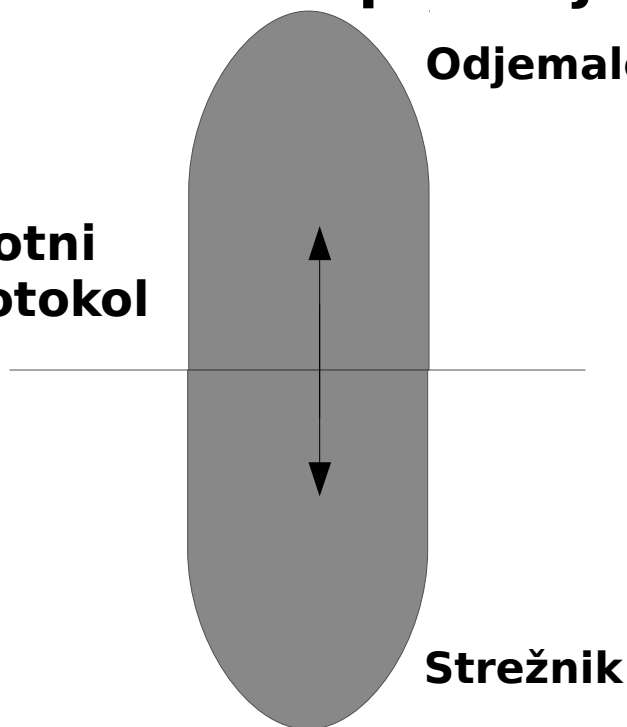
- zahteve:
 - povezati množico **grafičnih** delovnih postaj in jih uporabiti kot učne pripomočke
 - poganjati grafične aplikacije na poljubnem sistemu
 - poganjati grafične aplikacije na oddaljenih sistemih
- rezultat:
 - ozenski sistem X, prvo strojno in pozicijsko neodvisno okolje
- lastnosti:
 - neodvisen od operacijskega sistema
 - grafično okolje veliko sistemov (HPUX, AIX, Solaris, ...)
 - sistem okna
 - koncept odjemalec/strežnik
 - odjemalec in strežnik lahko na različnih računalnikih

Okenski sistem X (X11)

Grafična aplikacija

Odjemalec

**Enotni
protokol**

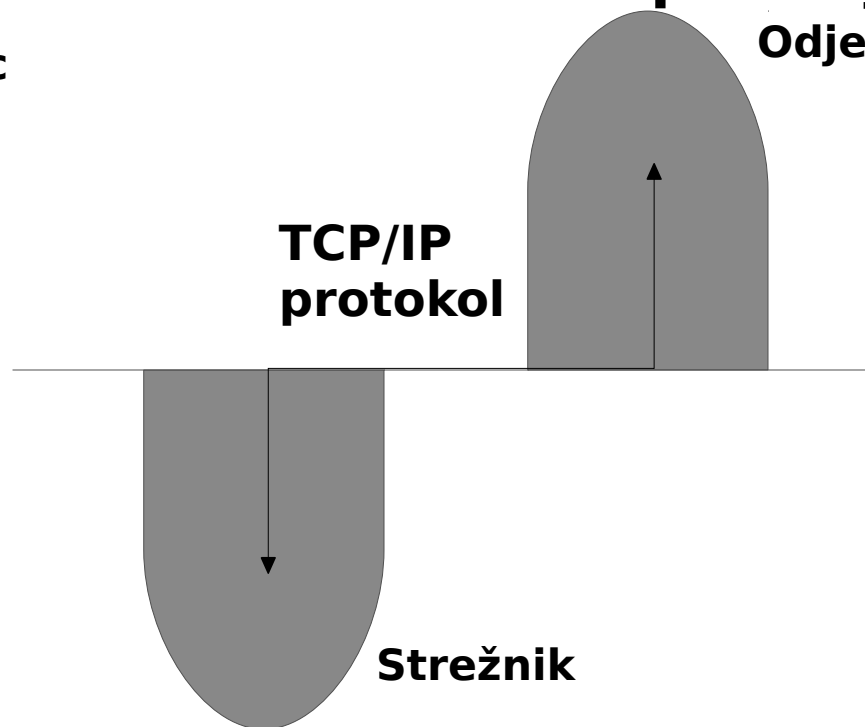


**Strojna
neodvisnost**

Grafična aplikacija

Odjemalec

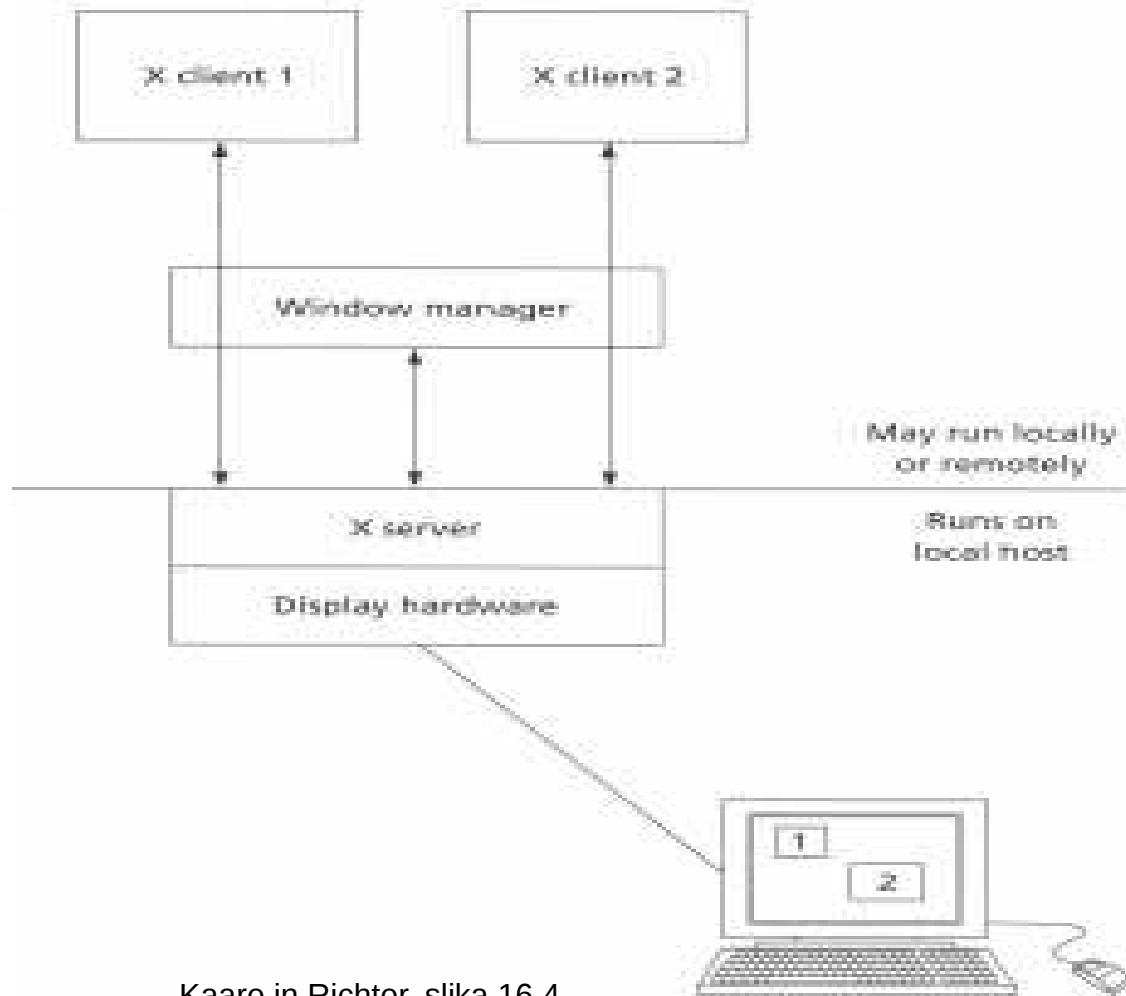
**TCP/IP
protokol**



**Pozicijska
neodvisnost**

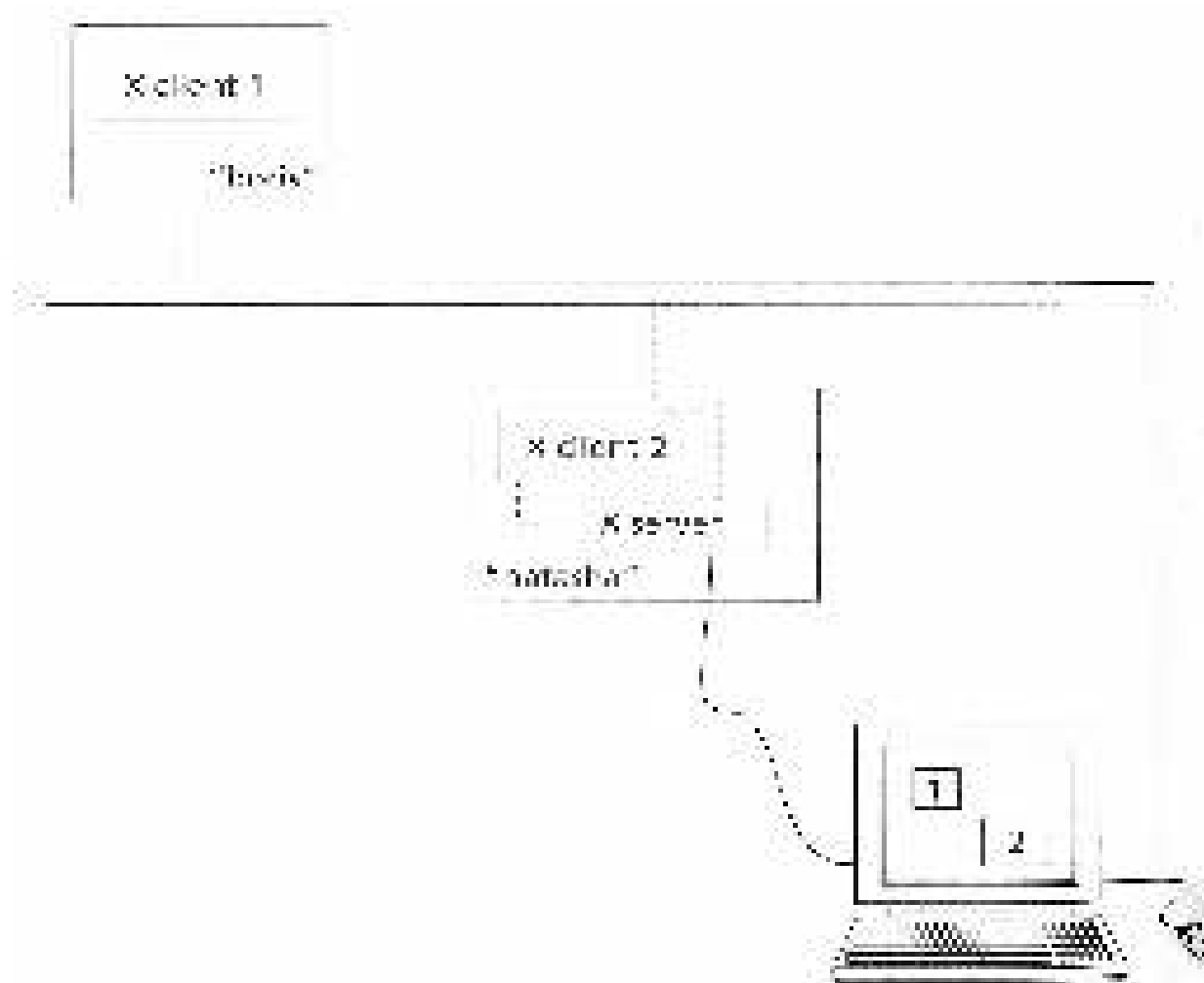


Ozenski sistem X (X11)



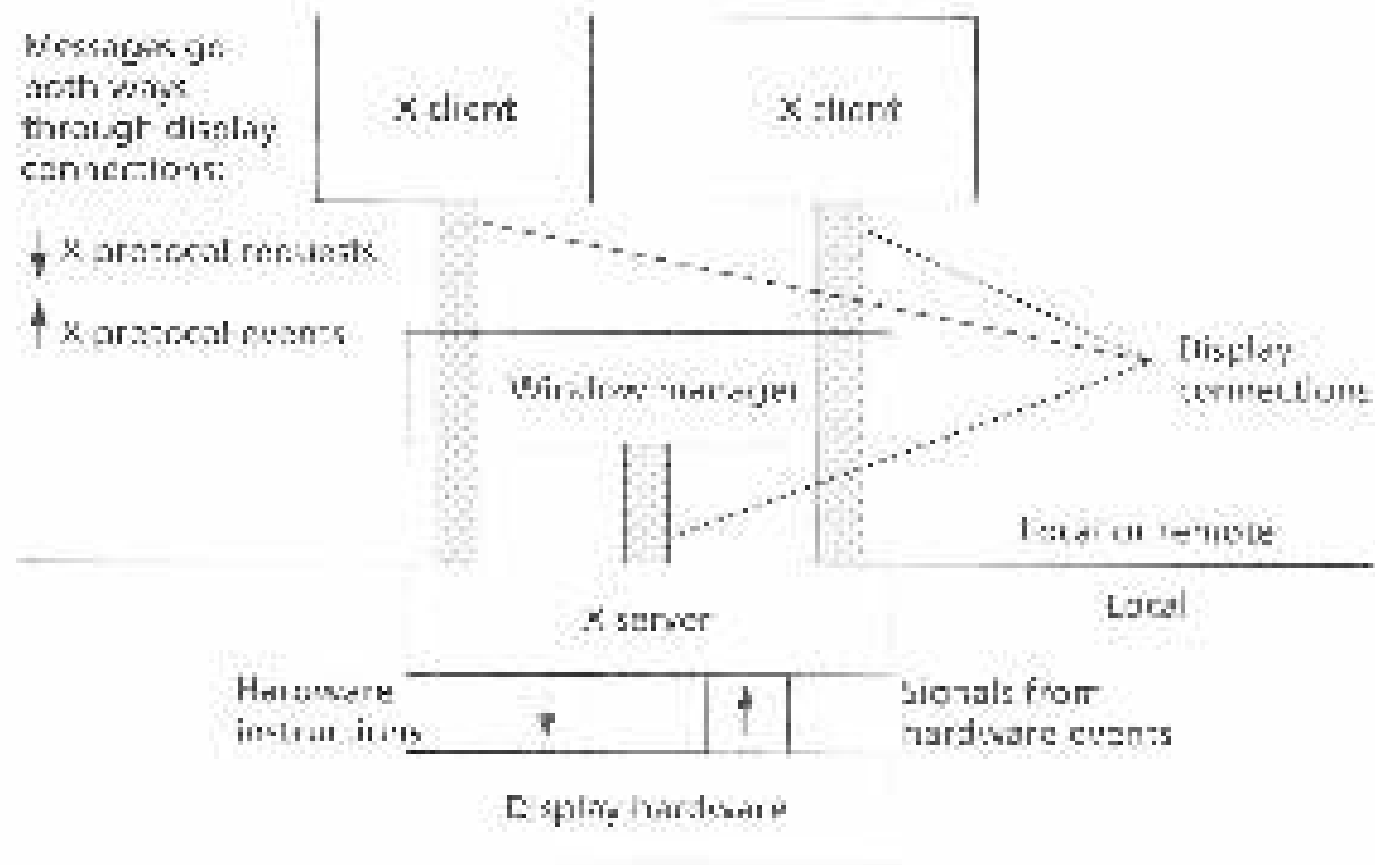


Oskenski sistem X (X11)





Okenski sistem X (X11)



Okenški sistem X (X11)

- Xt - bistvena orodja X (X toolkit Intrinsics) – knjižnica ki implementira aplikacijski programski vmesnik (API) za pomoč pri razvoju grafičnih programov za X
- uwm – upravnik oken (ultrix window manager) – standardni upravnik oken za okenški sistem X
- Xlib – grafična knjižnica – enaka za vsak računalnik, vsebuje funkcije in rutine za interakcijo s strežnikom X: risanje, barve, besedila, interakcija z upravniki oken, delo z dogodki, delo z grafičnim kontekstom
- X mrežni protokol
 - definira podatkovne strukture za prenos podatkov med strežniki in odjemalci
 - zagotavlja strojno in pozicijsko neodvisnost
 - omogoča uporabo več postaj naenkrat
 - uporabniki oken so obravnavani kot aplikacije

Okenski sistem X (X11)

- prikazovalnik (DISPLAY) – abstrakcija, ki predstavlja vhodno/izhodne naprave (miška, tipkovnica, zaslon) uporabnika
- strežnik nadzoruje prikazovalnik
 - nadzoruje dostop do prikazovalnika za odjemalce
 - pošilja sporočila po mreži
 - sledi zahtevam odjemalcev in posodablja okna
 - dvodimenzionalno risanje
 - sprejema vhodne dogodke in jih posreduje odjemalcem

Okenski sistem X (X11)

- odjemalci imajo dostop do prikazovalnika
 - izvajajo se simultano
 - več oken istočasno
 - poganjanje aplikacij na več računalnikih istočasno
 - aplikacije so neodvisne
 - uporabljajo mrežni protokol preko vmesnika Xlib
 - med seboj komunicirajo preko strežnika
 - vsak odjemalec posreduje strežniku zahtevo za gradnjo vsaj enega okna, ki je naslednik osnovnega okna



Javanski glavni program

This is the definition of the class
OurFirstProgram. The class
definition always contains the
method main().

```
public class OurFirstProgram  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Zdravo! DAS li Java??");  
    }  
}
```

This is the definition of the method main().
The keyword `public` indicates that globally accessible.
The keyword `static` ensures it is accessible even
though no objects of the class exist.
The keyword `void` indicates it does not return a value.



Primerjava Java in C

Program v Javi:

```
import java.io.*;

public class Hellow
{
    public static void main(String args[])
    {
        System.out.println("Hello world!");
    }
}
```

args.length
args[0], args[1], args[2], ...

Program v C:

```
#include <stdio.h>

void main(int argc, char *argv[])
{
    printf("Hello world!\n");
}
```

argc
argv[1], argv[2], argv[3], ...

Ozenski sistem X (primer)

- gradnja in izvajanje aplikacije v ozenskem sistemu X
 - vzpostavitev povezave s strežnikom
 - nastavljanje lastnosti okna
 - napotki upravniku oken
 - gradnja grafičnega konteksta
 - prikaz okna
 - obravnavanje dogodkov v zanki dogodkov
 - razkroj okna
 - prekinitev povezave s strežnikom



Ozenski sistem X (primer)

```
#include <stdio.h>
#include <string.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

char hello[] = "Hello, World.";
char title[] = "X-Window";

int main(int argc, char *argv[])
{
    Display *mydisplay; /* Struktura sadrži informaciju o prikazivaču */
    Window mywindow; /* Osnovni ID aplikacije u X-Window */
    GC gc; /* Graphics Context */
    XEvent myevent; /* Struktura sadrži informaciju o događajima */
    XScreen myscreen; /* Informacija o ekranu po "logičkom" */
    XColor mycolor; /* Informacija o boji */
    XSetWindowAttributes myattrs; /* Atributi prozora */
    unsigned long mybackground, myforeground; /* Boje pozadine i prednje */

    int i;
    char text[100];
    int len;

    /* Inicijalizacija */
    mydisplay = XOpenDisplay(""); /* Inicijalizacija povezivanja s X-Window */
    /* aplikacija se pokreće u okruženju X-Window */
    /* opis povezivanja, uzme ime prikazivača */

    /* Ekran zaslon je u uporabi */
    myscreen = DefaultScreen(mydisplay);
    mybackground = WhitePixel(mydisplay, myscreen);
    myforeground = BlackPixel(mydisplay, myscreen);
    /* Boje za pozadinu i prednju */
    /* tekst, tekst i boje */
}
```

```
typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window window;
} XAnyEvent;
```

```
Pixmap background_pixmap;
unsigned long background_pixel;
Pixmap border_pixmap;
unsigned long border_pixel;
int bit_gravity;
int win_gravity;
int backing_store;
unsigned long backing_planes;
unsigned long backing_pixel;
Bool save_under;
long event_mask;
long do_not_propagate_mask;
Bool override_redirect;
Colormap colormap;
Cursor cursor;
```

```
mydisplay = XOpenDisplay("");
```

```
myscreen = DefaultScreen(mydisplay);
mybackground = WhitePixel(mydisplay, myscreen);
myforeground = BlackPixel(mydisplay, myscreen);
```



Ozenski sistem X (primer)

```
/* Sam Z novo zgradi */  
myhint.x = 200; myhint.y = 200;  
/* sirina, dolzina */  
myhint.width = 350; myhint.height = 200;  
  
/* Katera komponenta u strukturi XSizeHints */  
/* se upotrebljava. Program izlaza pokazuje */  
/* da velicina okna */  
myhint.flags = PPosition | PSize;  
  
/* zahteva postaji, da zgradi okno aplikacije */  
/* okno je zgrajeno u osnovnom oknu */  
/* Ali okno je širok pet tock in cm */  
mywindow = XCreateSimpleWindow(mydisplay,  
    DefaultRootWindow(mydisplay), /* predhodnik */  
    myhint.x, myhint.y, myhint.width, myhint.height,  
    5, myforeground, mybackground);  
  
/* Informacija o na novo zgrajenom oknu za upravljanje oknom */  
XSetStandardProperties(mydisplay, mywindow, hello, hello,  
    None, argv, argc, &myhint);  
  
/* Zgradi graficki kontekst */  
mygc = XCreateGC(mydisplay, mywindow, 0, 0);  
XSetBackground(mydisplay, mygc, mybackground);  
XSetForeground(mydisplay, mygc, myforeground);  
  
/* Program zeli biti obavezan u svakom dogadjku i misec */  
/* tipkovnic, ali ot otkazivanje zaslona */  
XSelectInput(mydisplay, mywindow,  
    ButtonPressMask | KeyPressMask | ExposureMask);  
  
/* Prikaz okna na zaslonu */  
XMapRaised(mydisplay, mywindow);
```

myhint.flags = PPosition | PSize;

mywindow = XCreateSimpleWindow(mydisplay,
 DefaultRootWindow(mydisplay),
 myhint.x, myhint.y,
 myhint.width, myhint.height,
 5, myforeground, mybackground);

mygc = XCreateGC(mydisplay, mywindow, 0, 0);

XSelectInput(mydisplay, mywindow,
 ButtonPressMask | KeyPressMask | ExposureMask);

XMapRaised(mydisplay, mywindow);



Okenjski sistem X (primer)

```
done = 0;
while (done == 0) { /* zanka dogodkov */
    XNextEvent(mydisplay, &myevent); /* naslednji dogodek */
    switch (myevent.type) {
        case KeyPress: /* dogodek s tipkanjem */
            i = XLookupString(&myevent.xkey, text, 10, &mykey, NULL);
            if (i == 1 && text[0] == 'q') done = 1;
            break;
        case Expose: /* dogodek ob zagonu programa */
            if (myevent.xexpose.count == 0)
                XDrawImageString(
                    myevent.xexpose.display,
                    myevent.xexpose.window,
                    mygc, 50, 50, /* kam v okno zapise besedilo */
                    hello, strlen(hello));
            break;
        case ButtonPress: /* dogodek s klikom */
            XGetWindowAttributes(
                myevent.xbutton.display,
                myevent.xbutton.window,
                myw, &myevent.xbutton.w, &myevent.xbutton.h,
                &myevent.xbutton.x, &myevent.xbutton.y);
            break;
    } /* switch */
} /* while */

/* Razkrojni grafični kontekst */
XFreeGC(mydisplay, mygc);
/* Razkrojni okno */
XDestroyWindow(mydisplay, mywindow);
/* Prekine povezavo s skrinom X */
XCloseDisplay(mydisplay);

return 0;
} /* main */
```

while (done == 0) { ... }

```
case KeyPress: {
    i=XLookupString(&myevent.xkey,
        text, 10, &mykey, NULL);
    ...
}
```

case Expose: { ... }

case ButtonPress: { ... }

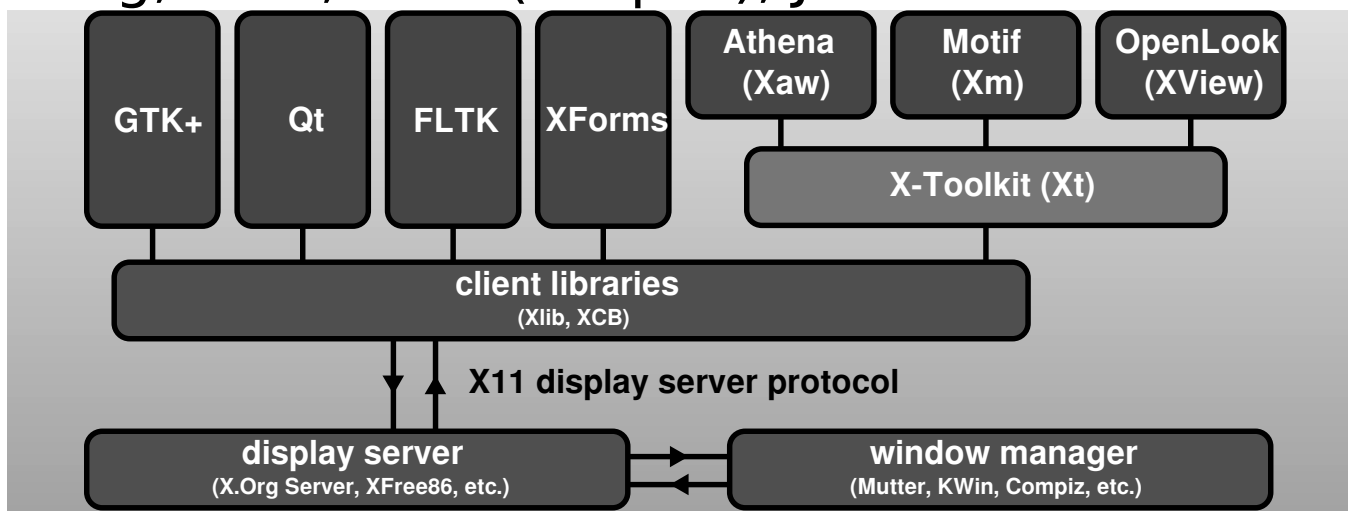
XFreeGC (mydisplay, mygc);

XDestroyWindow(mydisplay, mywindow);

XCloseDisplay(mydisplay);

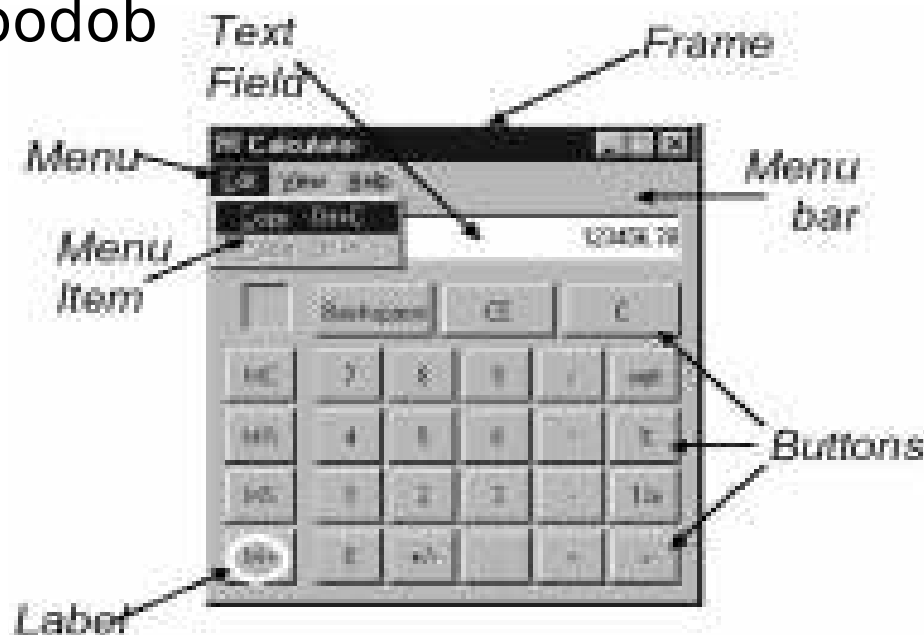
Orodja (toolkit) za načrtovanje UV in hierarhija

- nekatera programska okolja oziroma jeziki nudijo možnost razvoja dogodkovnih interaktivnih vmesnikov z grafičnim uporabniškim vmesnikom (GUV):
 - GTK+, Qt, wxWidgets
 - MFC, Cocoa, Motif
 - Swing, AWT, SWT (eclipse), JavaFX



Orodja za načrtovanje UV

- na voljo so knjižnice razredov, ki predstavljajo predloge za ustrezne grafične elemente
- objektom teh razredov (podobe) lahko nastavljamo različne lastnosti (barva, besedilo, velikost položaj, ...)
- nekatere podobe (vsebovalniki) lahko vsebujejo tudi eno ali več podob



Dogodkovni interaktivni vmesnik z GUV

- omogočajo neposredno interakcijo
- nelinearno izvajanje kode
- potek izvajanja je odvisen od uporabniškega vnosa
- veliko časa neizvajanja
- uporabnik lahko v vsakem trenutku dela različne stvari: pritisk na gumb, tipkanje, ...
- tem dejanjem rečemo **dogodki**
- interna zanka dogodkov
- potrebna je registracija rokovalnikov dogodkov

```
int main(int argc,
        char *argv[]) {
    deklaracija;
    inicialiacija;
    ...
    gradnja objektov GUV;
    registracija odzivnih
        funkcij;
    ...
    glavna zanka dogodkov;
}

callback1() {
    koda1;
}

callback2() {
    koda2;
}
```

- na dogodke čakajo rokovalniki dogodkov - odzivne funkcije



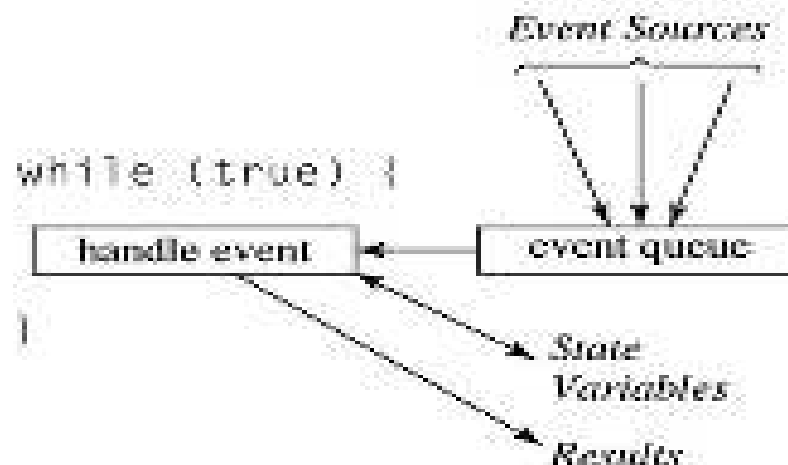
Dogodkovni interaktivni vmesnik z GUV

- Java – Swing/AWT
 - zanka dogodkov se izvaja avtomatično v ločeni niti
 - za zaznavo dogodkov moramo implementirati poslušalce:
 - ActionListener: *actionPerformed*
 - WindowListener: *windowClosed*
 - MouseListener: *mouseClicked*
 - MouseMotionListener: *mouseMoved*
 -

```
public Class{  
    int main(int argc,  
              char *argv[]) {  
        deklaracija;  
        inicializacija;  
        ...  
        gradnja objektov GUV;  
        registracija odzivnih  
            funkcij;  
        ...  
    }  
    listener1() {  
        koda1;  
    }  
    listener2() {  
        koda2;  
    }  
}
```

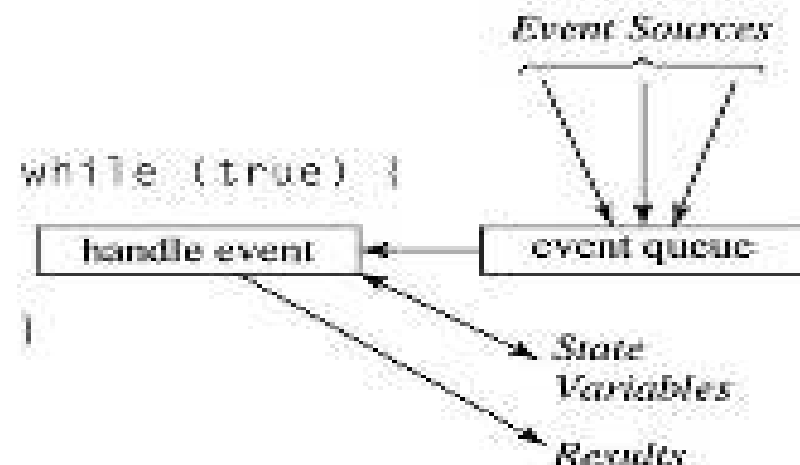
Obravnava dogodkov

- dogodki so objekti, ki so kreirani kot odziv na neko akcijo ali spremembo
- do dogodkov tipično pride ob interakciji uporabnika z neko podobo v GUV
- zanka dogodkov zazna dogodek, pogleda če je registriran ustrezen rokovalnik dogodkov, in sproži ustrezno akcijo oziroma rokovalnik dogodkov



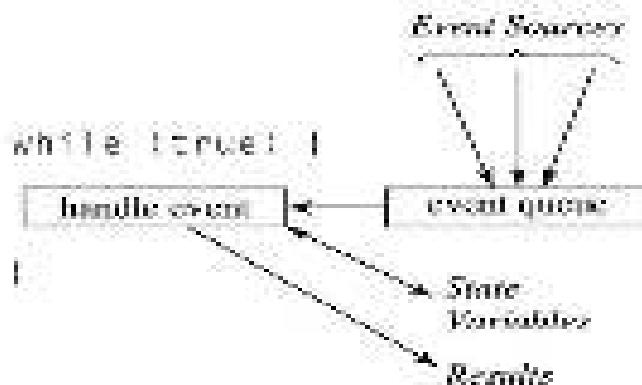
Vrsta dogodkov

- dogodki se hranijo v vrsti (queue)
- rokovalnik dogodkov lahko porabi več časa za procesiranje dogodka, kot pa je čas med pojavitvijo dveh dogodkov
- pojavitev novega dogodka pred koncem obravnave prejšnjega lahko privede do težav (dostop do podatkov)
- vrsta prepreči pojavitev težav – dogodek se ne procesira, dokler prejšnji ni procesiran
- premiki miške se pretvorijo v en sam dogodek



Zanka dogodkov

- čakanje na dogodek, dokler ni pripravljen
 - ko je pripravljen se vzame iz vrste
 - včasih se preprosti dogodki (pritisk/spust gumba miške,) pretvorijo v preveden dogodek (klik ali dvojni klik gumba miške, fokus, znaki,)
- zagotavljanje zanke dogodkov
 - napredna orodja za gradnjo GUI naredijo to interno
 - pri preprostih orodjih mora aplikacija to narediti sama



Model-pogled-nadzornik

- model (»Model« → M)
 - podatki povezani z aplikacijo
 - metode za dostop in spreminjanje podatkov in stanja
 - ni povezan z vmesnikom oziroma predstavitvijo
 - pogosto shranjen na ločeni lokaciji
- pogled (»View« → V)
 - prikaže vsebino modela uporabniku v ustreznem vmesniku
 - omogoči uporabniku upravljanje s podatki
 - ob spremembi modela se mora pogled prilagoditi
- nadzornik (»Controller« → C)
 - posrednik med modelom in pogledom
 - prevede uporabnikove akcije v operacije na modelu
 - posodobi model ko uporabnik upravlja s pogledom
 - primeri uporabnikov akcij: pritisk na gumb, izbira v meniju, ...

Motivacija za pristop MVC

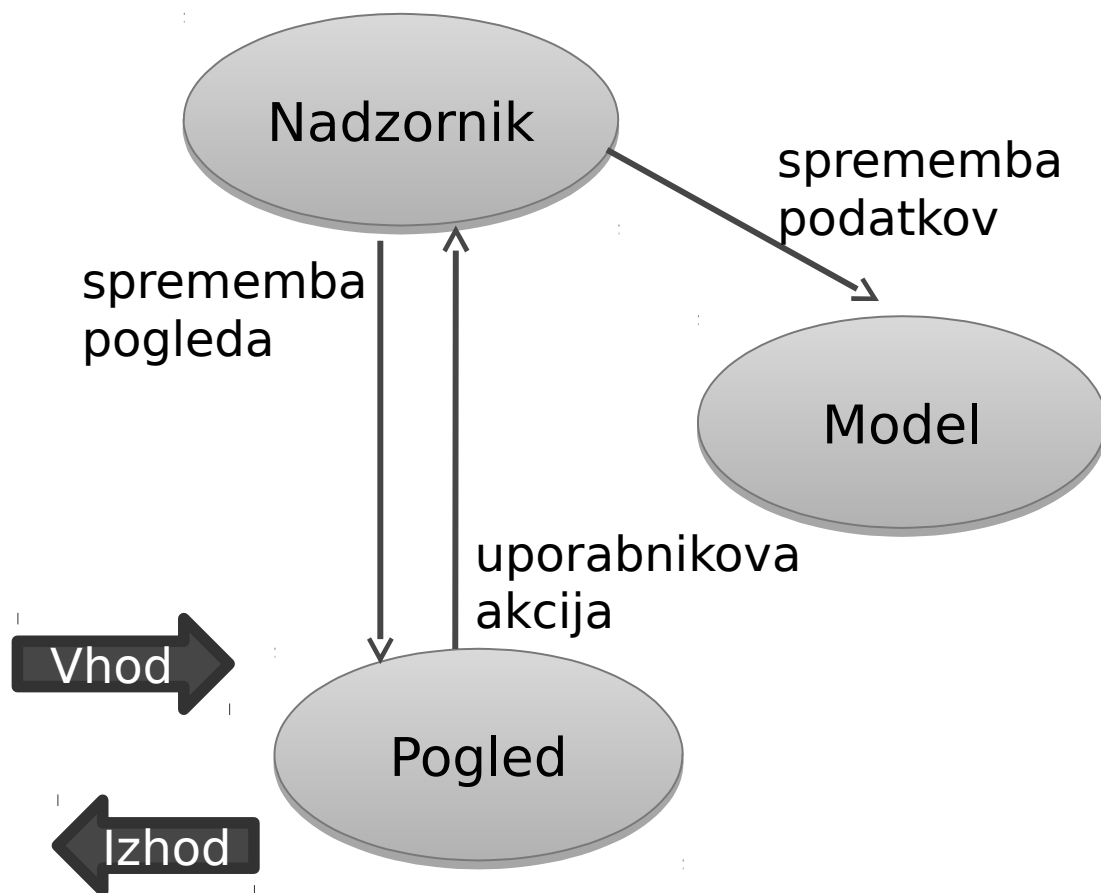
- osnovne komponente vsake interaktivne aplikacije
 - podatki s katerimi se rokuje
 - uporabniški vmesnik, ki služi za rokovanje s podatki
- podatki so logično neodvisni od načina predstavitve uporabniku
 - prikaz naj bi bilo mogoče načrtovati ločeno
- primer takega pristopa: porazdelitev ocen pri predmetu
 - mogoča je predstavitev s stolpičnim in/ali tortnim diagramom

Zakaj uporaba MVC

- dobra ločitev predstavitev, podatkov in poslovne logike
 - omogoča lažje vzdrževanje aplikacije
 - omogoča izogibanje enemu razredu/objektu/..., ki bo storil vse
- ločevanje delov aplikacije omogoča ponovno uporabo
 - z zmanjševanjem odvisnosti komponent se lažje vzame komponenta in uporabi v drugi aplikaciji
- omogoča zmanjševanje kompleksnosti delov aplikacije
 - kompleksnost je porazdeljena med različne dele
 - vsak del zase je manj kompleksen
- poveča fleksibilnost
 - vsak del aplikacije se lažje prilagodi kakršnim koli spremembam
- omogoča ločen razvoj, testiranje in vzdrževanje vsakega izmed delov aplikacije



Osnovna interakcija v MVC



Osnovna interakcija v MVC

- uporabnik izvaja interakcijo z uporabniškim vmesnikom (pogled)
- nadzornik prejme vhod od uporabniškega vmesnika
- nadzornik spremeni model glede na uporabnikovo akcijo
- model procesira posodobitve, ki jih zahteva nadzornik, pogosto je to pisanje/branje v podatkovno bazo
- nadzornik posodobi ali spremeni pogled s posodobitvami modela
- posodobljen uporabniški vmesnik se prikaže in ponovno čaka na uporabnikovo interakcijo