C4 Ereignisgesteuerte Programmierung

Bei der programmgesteuerten Eingabebehandlung hat man folgendes Schema:

```
finished := false;
while not finished do
  cmd := readLine();
  cmdType, args := parseCommand(cmd);
  case cmdType of
    t1 : interpret_t1_command(args);
    ...
    t2 : interpret_t2_command(args);
    ...
    tn : interpret_tn_command(args);
    ...
    quit : finished := true
  end case
end while
```

Oder man hat ein Automatenmodell:
In jedem Zustand wird eine Eingabe gelesen etwa: `b:=readkey`;
In Abhängigkeit von b wird zu einem neuen Zustand gegangen.

Beispiel für ein solches System ist etwa ein CAD-System, das durch Eingabe von Kommandos gesteuert wird. Immer wenn eine Eingabe benötigt wird, wird eine Routine z.B. `nextch` aufgerufen.
Trotzdem können die Kommandos auch durch einen Mausklick, etwa in einer Kommandoleiste eingegeben werden. Durch den Mausklick wird die zugehörige Eingabe in einem Eingabepuffer geschrieben, der dann durch die Aufrufe von `nextch` abgearbeitet wird.

Man hat hier folgendes Aufrufschema:
Programm → Eingabe(nextch)

Bei der Ereignisgesteuerten Programmierung ist jedem Ereignis (Tasttatureingabe, Mausklick) eine Prozedur zugeordnet, die dann aufgerufen wird. Die Prozedur wurde vorher für dieses Ereignis registriert.

Man hat hier folgendes Aufrufschema:
Eingabeereignis →Prozedur des Programms

Dieses Programmiermodell wird insbesondere angewandt, wenn man eine graphische Benutzeroberfläche mit verschiedenen Fenstern, die evtl. noch zu verschiedenen Programmen gehören, hat.
Ein solches Konzept ist realisiert bei

- MS-Windows,
- X-Windows,
- Delphi(VCL),
- JAVA(AWT, Swing).

Das Schema sieht dabei folgendermaßen aus:

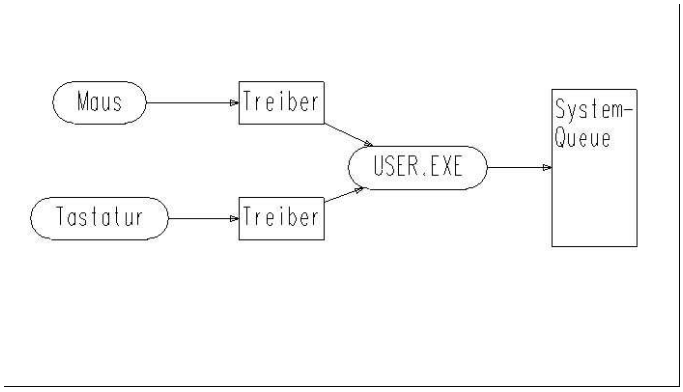
```
PROGRAM main.
USES ...;                 { Nutzung von Komponenten aus ... }
VAR event: TEvent       { Struktur, die Ereignis beschreibt }
BEGIN
  Init;                     { Initialisierungen }
  REPEAT                       { Wiederhole }
    GetEvent(event);              { Ereignis ermitteln }
    HandleEvent(event);           { auf Ereignis reagieren }
  UNTIL quit(event);        { bis Ende-Anforderung }
  Done;                     { Abschlußhandlungen }
END.
```

Da zu einem Zeitpunkt mehrere Fenster offen sein können, muss das Ereignis Tastatureingabe oder Mausklick jeweils dem richtigen Programm zugeordnet werden. Dazu wird eine Reihe von Ereignis-Queues unterhalten. Für ein Anwendungsprogramm kann die Ereignisschleife auch in einem Objekt verborgen werden. Als Beispiel folgendes Java Programm:

```
import javax.swing.*;

class MyFrame extends JFrame {
    public MyFrame () {
        super ("Hello, Swing!"); // Fenstertitel
        setSize (300, 200);      // Groesse setzen und
        setVisible (true);       // anzeigen
    }
}

public class hello {
    public static void main (String [] args) {
        new MyFrame ();
    }
}
```

Die Klasse JFrame registriert sich beim Betriebssystem. Wenn ein Mausklick in einem Bereich stattfindet, in dem der Frame dargestellt wird, wird dieses Ereignis zur Ereignis -Schlange des Frames geschickt. Ebenso erhält er die Tastaturereignisse, wenn er z.Z. den Fokus hat (Kopfzeile blau).

Windows Ereignisbehandlung



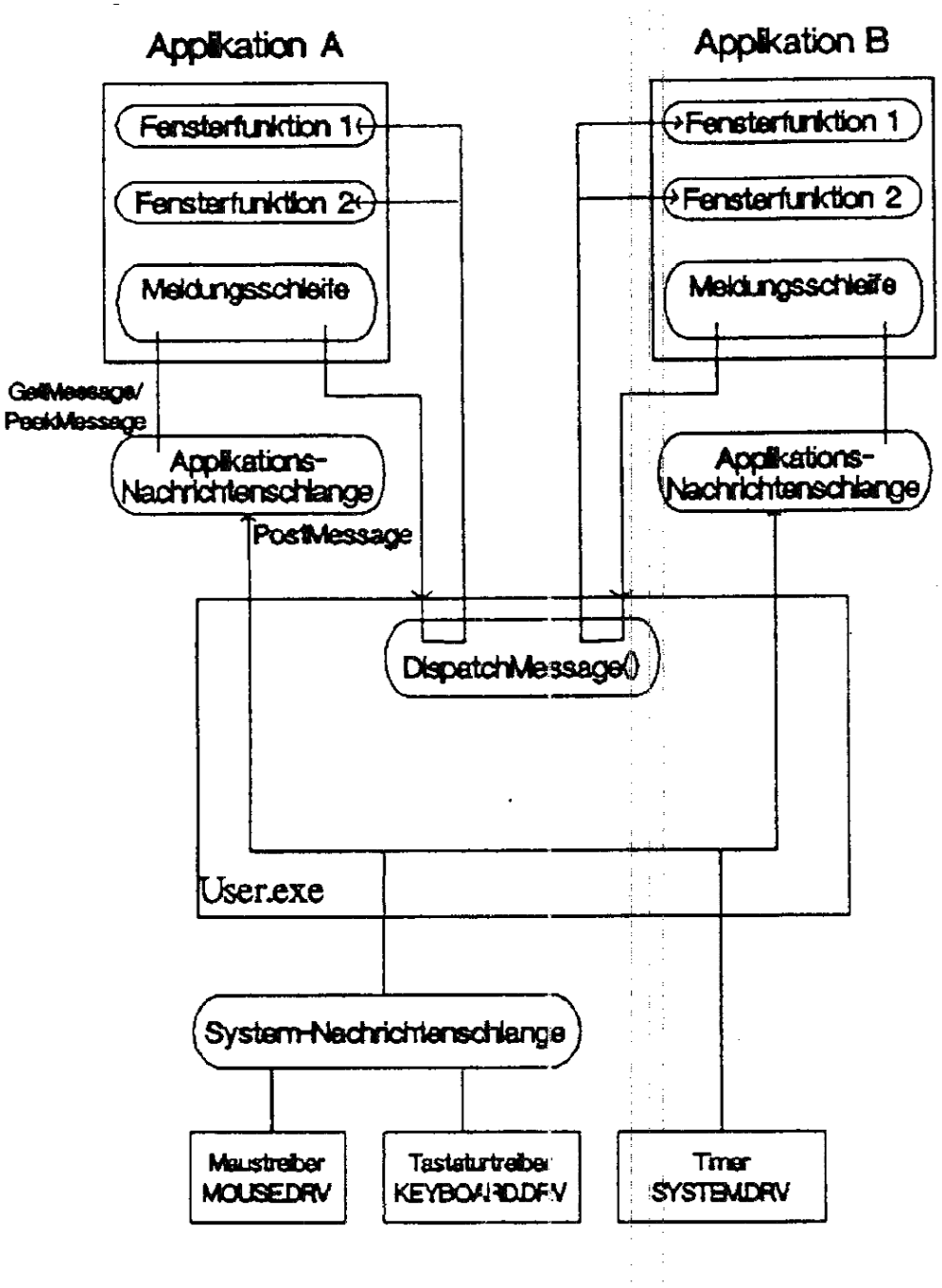
Ein Mausereignis liefert über seinen Treiber die Aktion (Click, Verschieben) und die Koordinaten. Die verschiedenen Ereignisse werden in der System-Queue gesammelt und durch den Windows-Kernel abgearbeitet.
Bei einem Mausereignis wird anhand der Koordinaten ermittelt, in welchem Fenster das Ereignis stattgefunden hat und der dem Fenster zugeordnete Task ermittelt.

Mauskoordinaten ----> Fenster -----> Task.

Bei einem Tastaturereignis wird durch den Treiber die gedrückte bzw. gelöste Taste gemeldet. Der Kernel prüft, ob die Taste Windows selbst zugeordnet ist. Wenn ja wird sie durch die Windows-Oberfläche bearbeitet. Sonst wird ermittelt, welches Fenster z.Z. den Eingabe-Fokus hat und dem zugehörendem Task wird das Ereignis übermittelt.

Tasttaur -----> z.Z. aktives Fenster ----> Task.

Für jeden task,der Ereignisse verarbeiten kann, gibt es eine Task-Queue.
Mit `GetMessage()` holt der Task das Ereignis bzw. die Message aus der Task-Queue.
Mit `DispatchMessage()` veranlasst der Task das System die entsprechende Fensterprozedur aufzurufen. Diese wurde zuvor von dem Task beim System für das entsprechende Fenster registriert.
Will der Task etwa bei der Tastatureingabe nicht die elementaren Ereignisse (`WM_KEYDOWN, WM_KEYUP`)verarbeiten sondern den Zeichenwert erhalten (`WM_CHAR`), muss er zuvor die Prozedur `TranslateMessage()` aufrufen.
Hat der Task mehrere Fenster, wird trotzdem nur eine Message-Queue benötigt, da das zugehörende Fenster in der Message enthalten ist. Den verschiedenen Fenstern sind jedoch i.A. verschiedene Fensterprozeduren zugeordnet, die durch `DispatchMessage` aufgerufen werden.




```
type TMessage =packedrecord
    Msg: Cardinal;
    Case Integer of
      0: (
        WParam: Longint;
        LParam: Longint;
        Result: Longint);
      1: (
        WParamLo: Word;
        WParamHi: Word;
        LParamLo: Word;
        LParamHi: Word;
        ResultLo: Word;
        ResultHi: Word);
  end;
```

Beschreibung
Der Typ TMessage repräsentiert in der Methode WndProc und in anderen Methoden eine Windows-Botschaft.
Das Feld Msg enthält die ID der Windows-Botschaft.
Das Feld WParam enthält den Parameter WParam der Botschaft. Für den Zugriff auf die nieder- und höherwertigen Words dieses Feldes müssen die Felder WParamLo und WParamHi verwendet werden.
Das Feld LParam enthält den Parameter LParam der Botschaft. Für den Zugriff auf die nieder- und höherwertigen Words dieses Feldes müssen die Felder LParamLo und LParamHi verwendet werden.
Das Feld Result enthält den Rückgabewert. Für den Zugriff auf die nieder- und höherwertigen Words dieses Feldes müssen die Felder ResultLo und ResultHi verwendet werden.

Aufbau eines Graphical User Interface (GUI)

Das erste GUI wurde entwickelt in den siebziger Jahren am Xerox Palo Alto Research Center (PARC) mit einem System für eine Smalltalk Umgebung. Davon abgeleitet wurden in der Folge eine Reihe weiterer Systeme entwickelt:

Apple LISA, McIntosh 1984
GEM (Atari)
Intuition (Commodore)
Presentation Manager (IBM, OS/2)
MS- Windows zunächst auf DOS-Basis, dann ab Windows NT eigenständig.
X-Windows (UNIX, Linux)
Openlook (SUN, AT&T)
Nextstep

Um überhaupt eine graphische Ausgabe machen zu können benötigen die GUIs eine Grafikschicht
Die Grafikschicht stellt dem Programmierer Funktionen zur Ausgabe von grafischen Elementen und Text zur Verfügung. Sie basiert auf den grafischen Treibern. Die höheren Schichten verwenden sie, um die von ihnen verwalteten Objekte darzustellen. Ihre besondere Bedeutung liegt in der Standardisierung von Grafikausgaben. Gegenüber dem Programm werden alle Hardwareabhängigkeiten verdeckt.
Die Grafikschicht bietet in erster Linie Funktionen zur Darstellung von Text, Linien, Rechtecken, Kreisen und Polygonen. Die geschlossenen Figuren sind gefüllt oder als Rahmenlinie verfügbar. Insgesamt sind es hauptsächlich Funktionen, die man von anderen Grafikbibliotheken her kennt. Durch die Orientierung an der Vektorgrafik ist eine Vergrößerung oder Verkleinerung leicht möglich. Ein weiterer Vorteil liegt in der Möglichkeit, höhere Auflösungen optimal unterstützen zu können. Daneben werden auch Pixelgrafiken unterstützt. Diese wird zur Darstellung von Piktogrammen benötigt.
Eine weitere wichtige Funktion ist das Verschieben und Kopieren von Rechtecken, die bei Fenstermanipulationen gebraucht werden. Dazu bieten die Grafikschichten an, einen gewöhnlichen Speicherbereich als Bildschirmbereich interpretieren zu können.

Neben den Zeichenfunktionen stellt die grafische Schicht Funktionen zur Einstellung der Darstellungsart zur Verfügung. Diese betreffen Linienstärke, Muster und Farbe bei grafischen Objekten oder Schriftart, Schriftstil und Größe bei Texten. Ein wichtiger Bereich sind die Anfragefunktionen. Damit sind Parameter der Grafikumgebung zu ermitteln. Hier finden sich wichtige Informationen wie die der Bildschirmdimension (bzw. Druckerauflösung), die Anzahl der verfügbaren Farben und das Größenverhältnis von X- und Y-Punkten, damit ein Quadrat immer wie ein Quadrat aussieht.

System	Name der vergleichbaren Schicht	
Macintosh	QuickDraw	
GEM	Virtual Device Interface (VDI)	
MS-Windows	Graphic Device Interface (GDI)	
OS/2	Graphics Programming Interface	
X-Windows	Xlib	

Eigenschaften eines GUI:
Bildschirm wird belegt durch ein- oder mehrere Fenster, die sich überlappen können und auch zu verschiedenen Programmen gehören können.

Eigenschaften der Fenster:
Die Fenster können Verkleinert, Vergrößert, Verschoben oder Iconisiert werden. Die erfordert ein Neuzeichnen von Fenstern, die vorher verdeckt waren.

Daher benötigt jedes Fenster eine Methode,die auf die Nachricht `WM_PAINT` reagiert.

```
procedure WMPaint(var Message: TWMPaint); message WM_PAINT;
```

Fenster können Scrollbalken haben d.h. das virtuelle Fenster ist größer als das am Bildschirm dargestellte.

In den Fenstern befinden sich verschiedene Steuerelemente (Controls). Steuerelemente sind visuelle Komponenten, die der Benutzer zur Laufzeit sehen und mit denen er interagieren kann. Alle Steuerelemente besitzen gemeinsame Eigenschaften, Methoden und Ereignisse, die sich speziell auf visuelle Aspekte beziehen. Dazu gehören beispielsweise die Position des Steuerelements, der mit dem Steuerelement verbundene Cursor oder Kurzhinweis, Methoden zum Zeichnen oder Verschieben des Steuerelements und Ereignisse, die auf Benutzeraktionen reagieren.

Einige Steuerelemente:

Menüs:

Die Menüs können hierarchisch aufgebaut sein. Jedem Menüpunkt ist entweder ein Untermenü oder eine Behandlungsroutine zugeordnet.

Aktionsschalter (Schaltfläche, Button):
Das Clicken auf den Button bewirkt die vorgegebene Aktion.

Radiobutton(Optionsfeld):
Mit Hilfe von Optionsfeldern können dem Benutzer eine Reihe von Optionen angeboten werden, die sich gegenseitig ausschließen und von denen immer nur eine zur selben Zeit aktiviert werden kann. Wenn der Benutzer auf ein Optionsfeld klickt, wird das Optionsfeld, das bisher aktiviert war, deaktiviert. Optionsfelder werden gewöhnlich in einem Gruppenfeld (TRadioGroup) zusammengefasst.

Checkbox:
CheckBox repräsentiert ein Kontrollfeld, das aktiviert (markiert) oder deaktiviert (nicht markiert) sein kann.

Messagebox:
Mit einer Messagebox kann eine Meldung auf dem Bildschirm ausgegeben werden. Diese Box muss dann vom Benutzer geschlossen werden. Damit hat man einen Dialog.
Man unterscheidet:
Modaler Dialog: Der Benutzer kann in der zugehörigen Anwendung nur in diesem Fenster ein Ereignis auslösen.
Nichtmodaler Dialog: Das Fenster verhält sich normal.

Dialogbox: Verschiedene Controls, die Ausgeben machen und den Benutzer zu einer Auswahl oder Eingabe auffordern.
TColorDialog erzeugt ein Dialogfeld zur Auswahl von Farben.
TCommonDialog ist der Vorfahr aller Komponenten, welche die in Windows üblichen Dialogfelder repräsentieren.
TFindDialog zeigt ein Suchdialogfeld an, mit dem der Benutzer Text in einer Datei suchen kann.
TFontDialog zeigt ein Dialogfeld zur Auswahl einer Schrift an.
TOpenDialog zeigt ein Dialogfeld an, in dem der Benutzer Dateien auswählen kann.
TPrintDlg ist die Standardaktion zum Anzeigen eines Druckerdialogs.
Die Komponente TPrinterSetupDialog zeigt ein Dialogfeld an, mit dem Drucker konfiguriert werden können.
TReplaceDialog zeigt ein Dialogfeld zum Suchen und Ersetzen von Text an.
TSaveDialog zeigt ein Dialogfeld zum Speichern von Dateien an.

Aufbau eines minimalen Programm in einer GUI (MS-Windows):

```
/*------------------------------------------------------
   HELLOWIN.C -- Displays "Hello, Windows" in client area
                 (c) Charles Petzold, 1992
  ------------------------------------------------------*/

#include

long FAR PASCAL _export WndProc (HWND, UINT, UINT, LONG) ;

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
     {
     static char szAppName[] = "HelloWin" ;
     HWND        hwnd ;
     MSG         msg ;
     WNDCLASS    wndclass ;

     if (!hPrevInstance)
          {
          wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc   = WndProc ;
          wndclass.cbClsExtra    = 0 ;
          wndclass.cbWndExtra    = 0 ;
          wndclass.hInstance     = hInstance ;
          wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
          wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
          wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass.lpszMenuName  = NULL ;
          wndclass.lpszClassName = szAppName ;

          RegisterClass (&wndclass) ;
          }

     hwnd = CreateWindow (szAppName,          // window class name
                    "The Hello Program",     // window caption
                     WS_OVERLAPPEDWINDOW,     // window style
                     CW_USEDEFAULT,           // initial x position
                     CW_USEDEFAULT,           // initial y position
                     CW_USEDEFAULT,           // initial x size
                     CW_USEDEFAULT,           // initial y size
                     NULL,                    // parent window handle
                     NULL,                    // window menu handle
                     hInstance,              // program instance handle
                    NULL) ;                  // creation parameters

     ShowWindow (hwnd, nCmdShow) ;
     UpdateWindow (hwnd) ;

     while (GetMessage (&msg, NULL, 0, 0))
          {
          TranslateMessage (&msg) ;
          DispatchMessage (&msg) ;
          }
     return msg.wParam ;
     }

long FAR PASCAL _export WndProc(HWND hwnd, UINT message,
                                UINT wParam, LONG lParam)
     {
     HDC         hdc ;
     PAINTSTRUCT ps ;
     RECT        rect ;
```

```
     switch (message)
          {
          case WM_PAINT:
               hdc = BeginPaint (hwnd, &ps) ;
               GetClientRect (hwnd, &rect) ;
               DrawText (hdc, "Hello, Windows!", -1, &rect,
                         DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
               EndPaint (hwnd, &ps) ;
               return 0 ;
          case WM_DESTROY:
               PostQuitMessage (0) ;
               return 0 ;
          }

     return DefWindowProc (hwnd, message, wParam, lParam) ;
     }
}
```

WndProc ist eine Call-Back Prozedur. D.h. die Prozedur wird dem System bekannt gegeben (registriert) und dann später vom System aufgerufen. In Windows ist die Aufrufkonvention für CallBackFunktionen Pascal, da die Funktionsparameter von links nach rechts in den Laufzeitstack gekellert werden, was zur Laufzeit wesentlich effektiver ist. C kellert die Parameter von rechts nach links wegen der Funktionen mit variablen Parameterlisten.

DefWindowProc: Standardeventhandler, der immer dann aufgerufen wird, wenn eine Windowfunction ein Event nicht auswertet. Der Standardeventhandler behandelt dann diesen Event und löst ggf. andere Events aus.

```
procedure TApplication.Run;
begin
  FRunning := True;
  try
    AddExitProc(DoneApplication);
    if FMainForm <> nil then
    begin
      case CmdShow of
       SW_SHOWMINNOACTIVE:FMainForm.FWindowState:=wsMinimized;
       SW_SHOWMAXIMIZED: MainForm.WindowState := wsMaximized;
      end;
      if FShowMainForm then
        if FMainForm.FWindowState = wsMinimized then
          Minimize else
          FMainForm.Visible := True;
      repeat
        try
          HandleMessage;
        except
          HandleException(Self);
        end;
      until Terminated;
    end;
  finally
    FRunning := False;
  end;
end;

function TApplication.ProcessMessage(var Msg: TMsg): Boolean;
var
  Handled: Boolean;
```

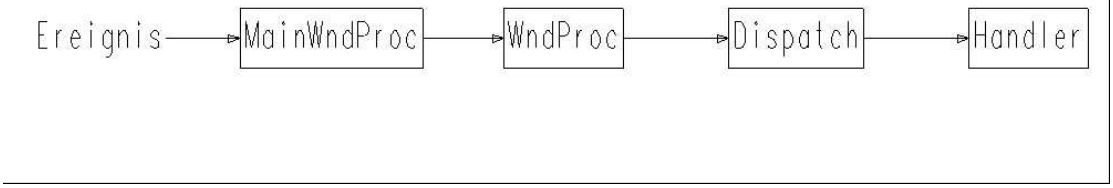
```
begin
  Result := False;
  if PeekMessage(Msg, 0, 0, 0, PM_REMOVE) then
  begin
    Result := True;
    if Msg.Message <> WM_QUIT then
    begin
      Handled := False;
      if Assigned(FOnMessage) then FOnMessage(Msg, Handled);
      if not IsHintMsg(Msg) and not Handled
      and not IsMDIMsg(Msg) and not IsKeyMsg(Msg)
      and not IsDlgMsg(Msg) then
      begin
        TranslateMessage(Msg);
        DispatchMessage(Msg);
      end;
    end
    else
      FTerminate := True;
  end;
end;

procedure TApplication.ProcessMessages;
var
  Msg: TMsg;
begin
  while ProcessMessage(Msg) do {loop};
end;

procedure TApplication.HandleMessage;
var
  Msg: TMsg;
begin
  if not ProcessMessage(Msg) then Idle(Msg);
end;
```

Was passiert bei DispatchMessage?

Normalerweise wird die Prozedur aufgerufen, die bei Windows für das entsprechende Fenster definiert wurde.
Möchte man dies objektorientiert machen geht man (in Delphi) folgendermaßen vor:
Delphi registriert für jeden Komponententyp eine Methode mit dem Namen `MainWndProc` als Fensterprozedur.
In `MainWndProc` wird die Botschaftsstruktur an eine virtuelle Methode `WndProc` übergeben. Tritt eine Exception auf, erfolgt der Aufruf der Methode `HandleException` der Klasse.
`MainWndProc` ist eine nicht-virtuelle Methode, Anpassungen erfolgen in der virtuellen Methode `WndProc,` die von jedem Komponententyp überschrieben werden kann. Dabei können z.B. gerade unerwünschte Botschaften abgefangen werden.
Beispiel: Komponenten, die gerade mit der Maus verschoben werden, ignorieren Tastaturereignisse. Am Ende ruft `WndProc` die Methode `Dispatch` auf.
`Dispatch` ist eine nicht virtuelle Methode der Komponente, die prüft, ob für das Ereignis eine methode definiert ist, z.B. in der Form:

```
procedure WMPaint(var Message: TWMPaint); message WM_PAINT;
```

Wenn ja, wird die entsprechende Methode aufgerufen, wenn nein wird die Methode `DefaultHandler` aufgerufen.



Für jedes Ereignis ist damit definiert, wie bei einer Komponente damit verfahren werden soll, evtl. wird es einfach ignoriert.
Für die Steuerelemente sind einige Standardereignisse vordefiniert:
OnClick, OnDblClick
OnDragDrop, OnDragOver, OnEndDrag
OnMouseDown, OnMouseMove, OnMouseUp

Fensterorientierte Steuerelemente haben zusätzlich:
OnEnter, OnExit
OnKeyDown, OnKeyUp, OnKeyPress