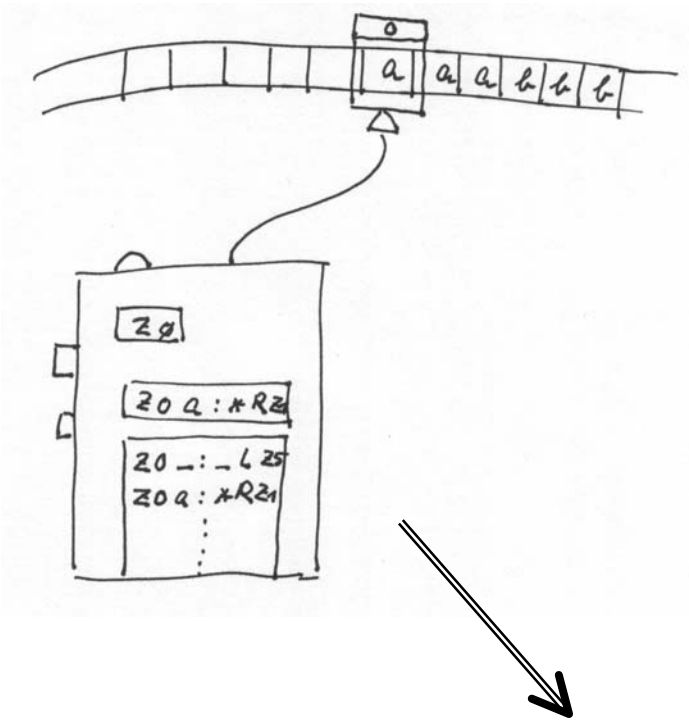
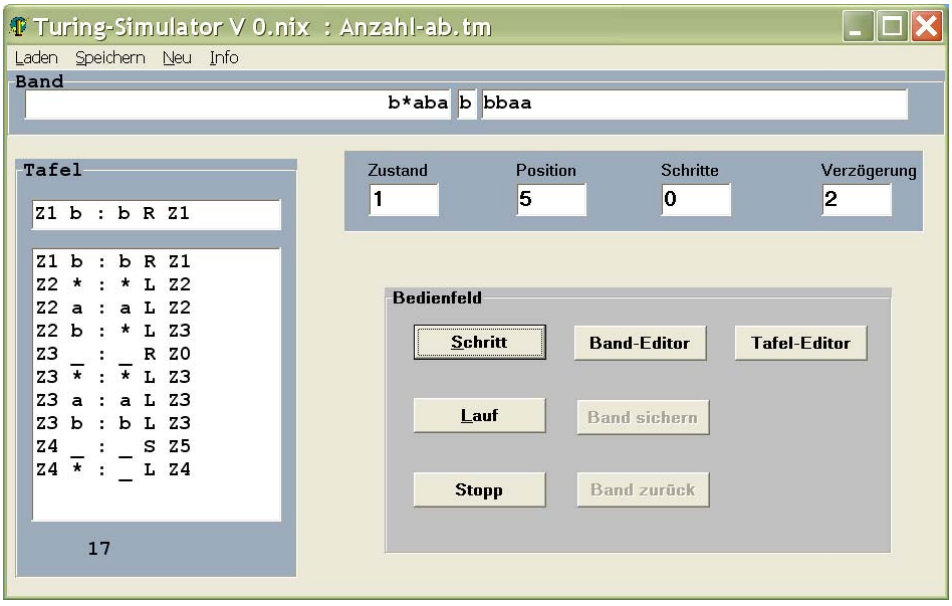
IFB-Veranstaltung : OOP mit Delphi II

U.Mayr : Turing-Simulator

Ein Softwareprojekt mit Delphi





Inhaltsverzeichnis

1. Ein Softwareprojekt : Die Simulation einer Turingmaschine

Aufgabenstellung :

Die Anwendung soll Turing-Maschinen mit einem unbegrenzten Band simulieren.

Folgende Fähigkeiten sollen dabei realisiert werden.

Schritt eine bereits geladene Maschine soll einen Befehl ausführen

Lauf eine bereits geladene Maschine soll solange Befehle ausführen bis sie gestoppt wird durch einen Stopp-Befehl, durch den Nutzer oder weil kein Befehl passt.

Eingabe aller Angaben soll möglich sein, die die Maschine für ihre Arbeit braucht.

Speichern

Eine Maschine soll gespeichert werden.

Laden Eine Maschine soll geladen werden.

Neu Eine neue Maschine mit leerem Band und leerer Tafel soll bereit gestellt werden.

Tafel-Editor

Zu einer Maschine sollen Befehle hinzugefügt und gelöscht werden können.

Band-Editor

Die Bandbelegung soll verändert werden können.

Vorgehensweise

Entwicklung der Anwendung nach dem MVC-Konzept.

Model (Modell)

das Fachkonzept bzw. der algorithmische Kern der Anwendung wird objektorientiert durch Klassen modelliert, die miteinander in Beziehungen stehen

View (Ansicht)

Konstruktion von GUI – Objekten zur Darstellung des Modells für den Nutzer

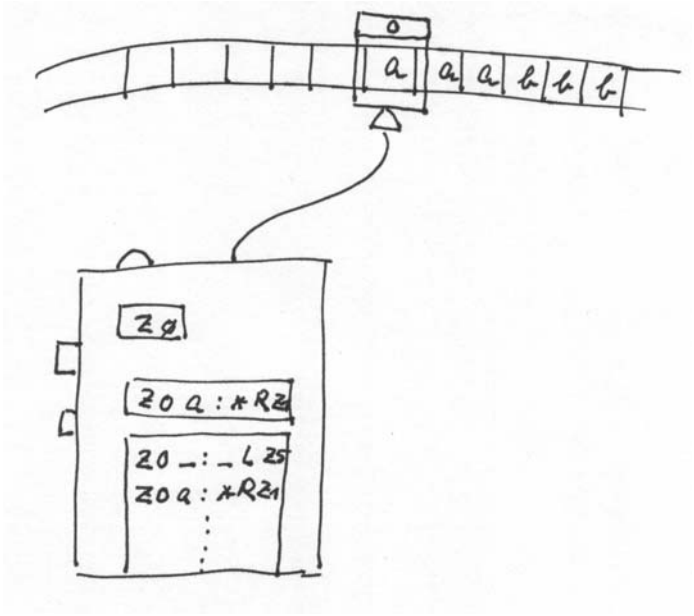
Controller (Steuerung)

Die Einheiten der Anwendung, die den Ablauf der Anwendung steuern : Kommunikation mit dem Nutzer, veranlassen der Änderungen im Modell und der Anpassung der Ansichten.

Die verschiedenen Klassen des Modells werden in Delphi rein textlich entworfen und in eigenen Units abgelegt. Beim Entwurf können UML-Diagramme helfen.

Die GUI-Objekte zur Realisierung von View/Controller können dagegen graphisch entworfen werden. In Delphi werden die verschiedenen Ansichten in entsprechenden Formularen realisiert.

2. Modellierung einer Turingmaschine



Die Turing-Maschine soll als eigenständig-agierendes Objekt aufgefasst und als Klasse TMaschine formuliert werden.

Was gehört zu einer Turingmaschine ?

Attribute von TMaschine

- Band mit Schreib-Lesekopf (Objekt vom Typ TBand)
- Tafel mit den Befehlen (Objekt vom Typ TTafel)
- Zustand
- Schrittzähler
- ---

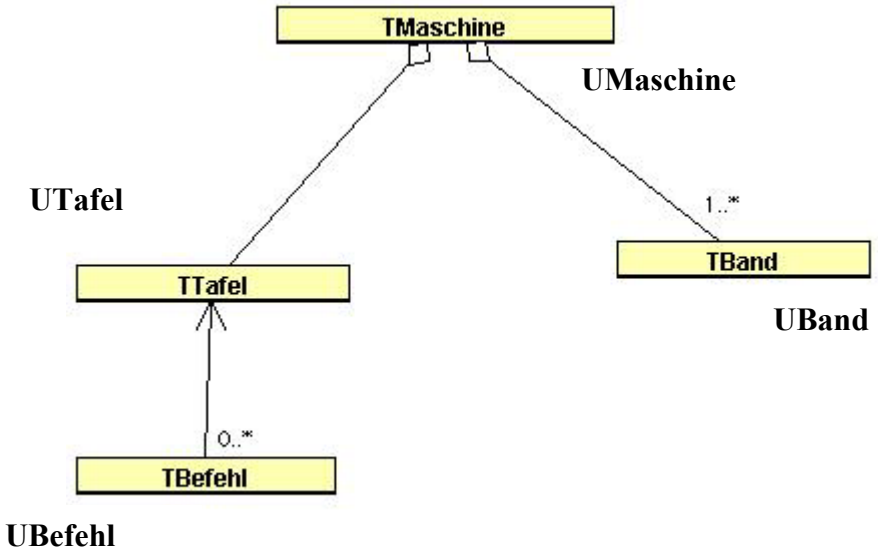
Welche Aufgaben soll eine Turing-Maschine erledigen können ?

Methoden von TMaschine

- Einzelschritt ausführen
- Lauf bis zum Stopp
- Speichern der aktuellen Maschine
- Laden einer Maschine
- Aktuellen Befehl aus der Tafel holen
- Rücksetzen nach einem Stopp durch einen Stoppbefehl
- ---

Das UML Klassendiagramm für TMaschine

mit Angabe der Unit-Namen



Wir stellen die Formulierung von TMaschine n Delphi zurück und formulieren zuerst die Teilklassen.

3. Die Klasse TBand

Formulierung in Delphi (Unit UBand)

```
interface
uses classes;

type

  TMenge = set of char;

  TBand = class(TObject)
    private
      FLWort    : string;
      FZeichen  : char;
      FRWort    : string;
      FBewegung : TMenge;

      FPosition : integer;

      procedure rechts;
      procedure links;
      procedure FAendere_Position( d : integer );
    public
      Property  LWort    : string  read FLWort    write FLWort;
      Property  RWort    : string  read FRWort    write FRWort;
      Property  Zeichen  : char    read FZeichen  write FZeichen;
      Property  Bewegung : TMenge  read FBewegung;

      Property  Position : integer read FPosition
                                   write FAendere_Position;

      procedure initialisiere(lw:string; z:char; rw:string);
      procedure ausfuehren( move, Zeichen :char );
      procedure speichern( var f : textfile);
      procedure laden( var f : textfile );
      constructor create;
    end;
```

Eigenschaften (Properties) wie Position dienen in Pascal dazu das Geheimnisprinzip für Attribute zu wahren ohne auf die übliche Schreibweise mit Wertzweisungen zu verzichten. Ein Sachverhalt der das lästige benutzen von Zugriffsmethoden vereinfacht, der aber von OO-Puristen äußerst kritisch gesehen wird.

Zur Eigenschaft **Position** gehört der Speicherplatz **FPosition**. Die Read- und Write-Angaben geben an, ob direkt auf **FPosition** zugegriffen werden kann oder ob besondere Methoden dafür vorgesehen sind. Fehlt die Write-Angabe, so kann nur lesend zugegriffen werden.

Die Anweisung :

```
Band.Position := Band.Position + 1;
```

bedeutet damit eine Abkürzung für :

```
Band.FAendere_Position(Band.FPosition+1);
```

Die Methode FAendere_Position bewirkt, dass mit der Änderung der Bandposition gleichzeitig die Bandkonfiguration angepasst wird.

```
procedure TBand.FAendere_Position( d : integer );
var i : integer;
begin
  d:= d - FPosition;
  for i:=  1   to   d do rechts;
  for i:= -1 downto d do links;
end;
```

Dabei ist die Methode rechts folgendermaßen definiert :

```
procedure TBand.rechts;
begin
  FLWort := FLWort+ FZeichen;
  if length(FRwort)>0 then begin
    FZeichen:= FRWort[1];
    delete(FRWort,1,1); end
  else
    FZeichen := ' ';
  FPosition := FPosition + 1;
end;
```

Bemerkung :

Bei den Eigenschaften **LWort, RWort, Zeichen, Bewegung** wird direkt auf die privaten Speicherplätze zugegriffen. Man hätte hier die Speicherplätze mit gleichem Effekt als public deklarieren können.

Gehen wir noch auf die Methoden ausführen, create und laden ein.

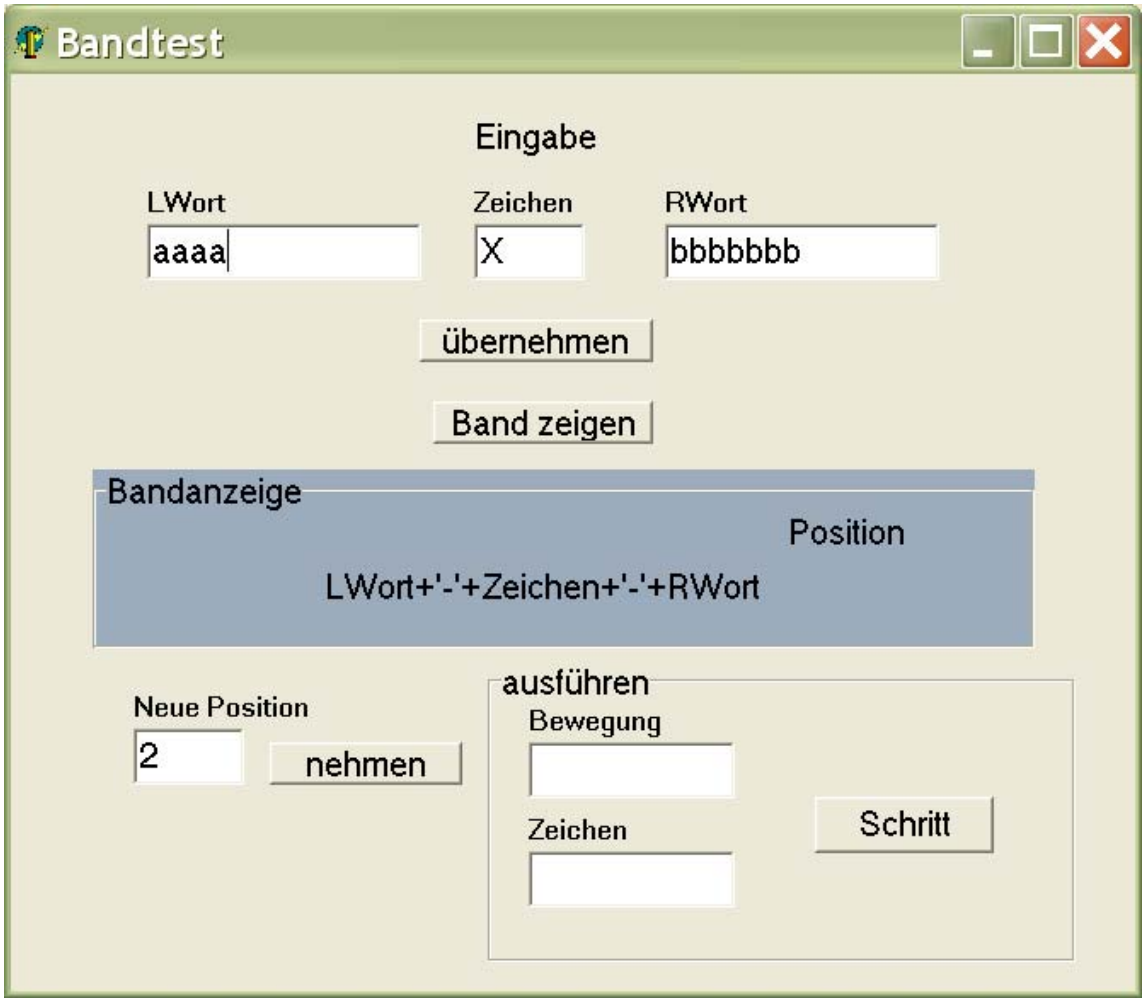
```
constructor TBand.create;
begin
  inherited create;
  initialisiere('',' ','');
  FBewegung := ['R','L','N','S'];
  FPosition := 0;
end;

procedure TBand.ausfuehren( move, Zeichen :char );
begin
  if move in Bewegung then begin
    FZeichen := Zeichen;
    case move of
      'R' :  rechts;
      'L' :  links;
    end;
  end;
end;

procedure TBand.laden( var f : textfile );
begin
  readln(f,FLWort);
  readln(f,FZeichen);
  readln(f,FRWort);
  readln(f,FPosition);
end;
```

Übungsphase : Test der Unit TBand / Teilprojekt Bandeditor

Sie sollen jetzt die vorliegende Unit UBand in einem kleinen Projekt Bandtest.dpr testen.

Vorgehensweise :

1. Öffnen Sie das Projekt PBandtest.dpr.
2. Fügen Sie dem Projekt die Unit UBand hinzu.
3. Vereinbaren Sie in UBandtest eine Variable Band : TBand.
   Vergessen Sie die uses-Anweisung für UBand nicht.
4. Ergänzen Sie die Leerprozduren zum Testen der Unit UBand.
5. *Für ganz Schnelle : Speichern und Laden ausprobieren*



4. Die Klassen TTafel und TBefehl

Da die Turing-Tafel Befehle enthält wird zuerst die Klasse TBefehl definiert.

```
TBefehl = class
  private
    FAZustand : integer;
    FEZustand : integer;
    FAZeichen : char;
    FEZeichen : char;
    FBewegung : char;
    FMarke    : boolean;
  public
    property AZustand : integer read   FAZustand
                                write  FAZustand;
    - - - - - - - - - - - - - -
    property Marke    : boolean read   FMarke
                                write  FMarke;
    constructor create; // setzt Zustände auf -1
    procedure initialisiere(za:integer;ca,ce,b:char;
                            ze : integer; m:boolean);
    procedure speichere( var f : textfile);
    procedure lade( var f : textfile);
    function  als_text : string;
end;
```

Die Klasse TTafel

Die Turing-Tafel besteht aus einer Liste von Befehlen. Zur Verwaltung der Liste gehören Einfügen, Löschen und Suchen von Befehlen.
Darüber hinaus muss die Turing-Tafel gespeichert und geladen werden können. Da in Delphi bereits eine universelle lineare Liste (Klasse TList) existiert, muss diese nur für unsere Zwecke erweitert werden.
TList ist keine GUI-Klasse und kann daher als Container-Klasse in vielen Modellen verwendet werden.

Informationen über TList aus der Delphi-Hilfe

TList verwaltet eine Liste von Zeigern auf Objekte.

Unit classes

Beschreibung : Mit einem TList-Objekt wird eine Liste von Objekten gespeichert und verwaltet. TList führt verschiedene Eigenschaften und Methoden ein, die folgenden Zwecken dienen:

- Objekte zur Liste hinzufügen oder daraus entfernen.
- Objekte in der Liste neu anordnen.
- Objekte in der Liste finden und darauf zugreifen.
- Objekte in der Liste sortieren.

Einige Attribute/Eigenschaften :

`Count` gibt an wie viele Elemente in der Liste sind

`Items` **`( property Items[Index: Integer]: Pointer; )`**
Array der Zeiger auf die Objekte der Liste

Die Eigenschaft **`Items`** erlaubt es , dass man wie bei einem Array über einen Index auf die Elemente der Liste zugreifen kann.

`L.Items[0]` … **`L.Items[L.count-1]`** sind die Elemente einer Liste L.

In eine Liste kann alles eingefügt werden, was über Zeiger angesprochen werden kann, da nur die Zeiger gespeichert werden.

L.Items[i] ist für Delphi nur ein Zeiger.

Will man mit einem eingefügten Objekt der Liste arbeiten, so muss zuerst eine **Typumwandlung** stattfinden.

Zum Beispiel `b := TBefehl( L.Items[2]);`

Ordne b das Element aus L mit dem Index 2 zu, das ein Befehl ist.

Allein der Programmierer ist dafür verantwortlich, was in L eigentlich gespeichert ist.

Einige Methoden von TList :

```
function Add(Item: Pointer): Integer
procedure Clear
procedure Delete(Index: Integer)
function IndexOf(Item: Pointer): Integer
procedure Insert(Index: Integer; Item: Pointer)
function Remove(Item: Pointer): Integer;
```

Beim Ableiten von TTafel aus TList soll dafür gesorgt werden, dass die Liste angepasst wird für die Aufnahme von Befehlen.

```
TTafel = class(TList)
  private
    function  bewerte( i: integer; c: char ): integer;
    function  Wert( b:TBefehl ):integer;
  public
    function  gib_Befehl( i : integer) : TBefehl;
    procedure einfuegen( b : TBefehl);
    function  suche_Befehl( Zustand: integer; ch : char)
                                                    :
TBefehl;
    function  suche_Index( Zustand: integer; ch: char)
                                                    :
integer;
    procedure loesche_Befehl( i: integer);
    procedure entleeren;
    procedure laden(var f: textfile);
    procedure speichern(var f: textfile);
end;
```

Einige Implementationen :

```
  function TTafel.gib_Befehl( i : integer) : TBefehl;
  begin
    result := nil;
    if (0<=i)and(i<count) then begin
      result := TBefehl(items[i]);
    end;
  end;
```

```
function  TTafel.suche_Befehl( Zustand : integer;
                               ch : char) : TBefehl;
var i : integer;
begin
  result:= nil;
  for i:= 0 to count-1 do
    if  Wert(gib_Befehl(i)) = bewerte(Zustand,ch) then
      result:= gib_Befehl(i);
end;
```

```
procedure TTafel.einfuegen( b : TBefehl);
var Position, i : integer;
begin
  if count=0 then add(b)
  else begin

loesche_Befehl(suche_index(b.AZustand,b.AZeichen));
    Position := count;
    for i:= count-1 downto 0 do
      if Wert(gib_Befehl(i)) > Wert(b) then
        Position:= Position-1;
    if Position < count then insert(Position,b)
    else add(b);
  end;
end;
```

```
procedure TTafel.speichern(var f : textfile);
var i : integer;
begin
```

```
    for i:= 0 to count-1 do
TBefehl(items[i]).speichere(f);
  end;
```

5. Die Klasse TMaschine

Die Klasse TMaschine kann nun formuliert werden.

```
uses UBefehl, UBand, UTafel, classes;
type

  TMaschine = class
    private
      FOnExecute : TNotifyEvent;
    public
      Tafel : TTafel;
      Band  : TBand;
      Z_aktuell   : integer;
      Schrittzahl : integer;
      Verzoegerung: integer;
      beendet     : boolean; // durch Stoppbefehl
      angehalten  : boolean; // durch Nutzer, Marke
      constructor  create;
      destructor   free;
      procedure Schritt;
      procedure Lauf;
      procedure ruecksetzen;
      procedure laden( var f :textfile);
      procedure speichern( var f : textfile);
      function  Befehl_aktuell : TBefehl;
      property  OnExecute :TNotifyEvent read  FOnExecute
                                        write FOnExecute;
  end;
```

```
  constructor  TMaschine.create;
  begin
    Tafel:= TTafel.create;
    Band := TBand.create;
    Band.initialisiere('',' ','');
    Z_aktuell    := 0;
    Schrittzahl  := 0;
    Verzoegerung := 10;
    beendet      := false;
    angehalten   := false;
  end;
```

```
destructor   TMaschine.free;
begin
  Tafel.entleeren;
  inherited free;
end;
```

```
procedure TMaschine.Schritt;
var b: TBefehl;
begin
  b:= Befehl_aktuell;
  if (b<>nil)and not(beendet) then begin
    Band.ausfuehren(b.Bewegung,b.EZeichen);
    Z_aktuell  := b.EZustand;
    Schrittzahl:= Schrittzahl + 1;
    if b.Bewegung = 'S'    then  beendet := true;
    if assigned(Onexecute) then  FOnexecute(self);
  end;
end;

procedure TMaschine.Lauf;
var b : TBefehl;
begin
  angehalten := false;
  repeat
    Schritt;
    b:= Befehl_aktuell;
    if b<>nil then angehalten:= angehalten or b.Marke;
  until (beendet)or(angehalten)or(b=nil);
end;
```

Bemerkung :
Das Ereignis OnExecute besteht aus einer Zeigervariablen, die auf eine Methode des Typs :

```
TNotifyEvent = procedure (Sender: TObject) of object;
```

zeigt.

Der Nutzer des Typs TMaschine kann nachträglich durch eine Zuweisung eine Methode eigener Wahl festlegen.

```
if assigned(Onexecute) then  FOnexecute(self);
```

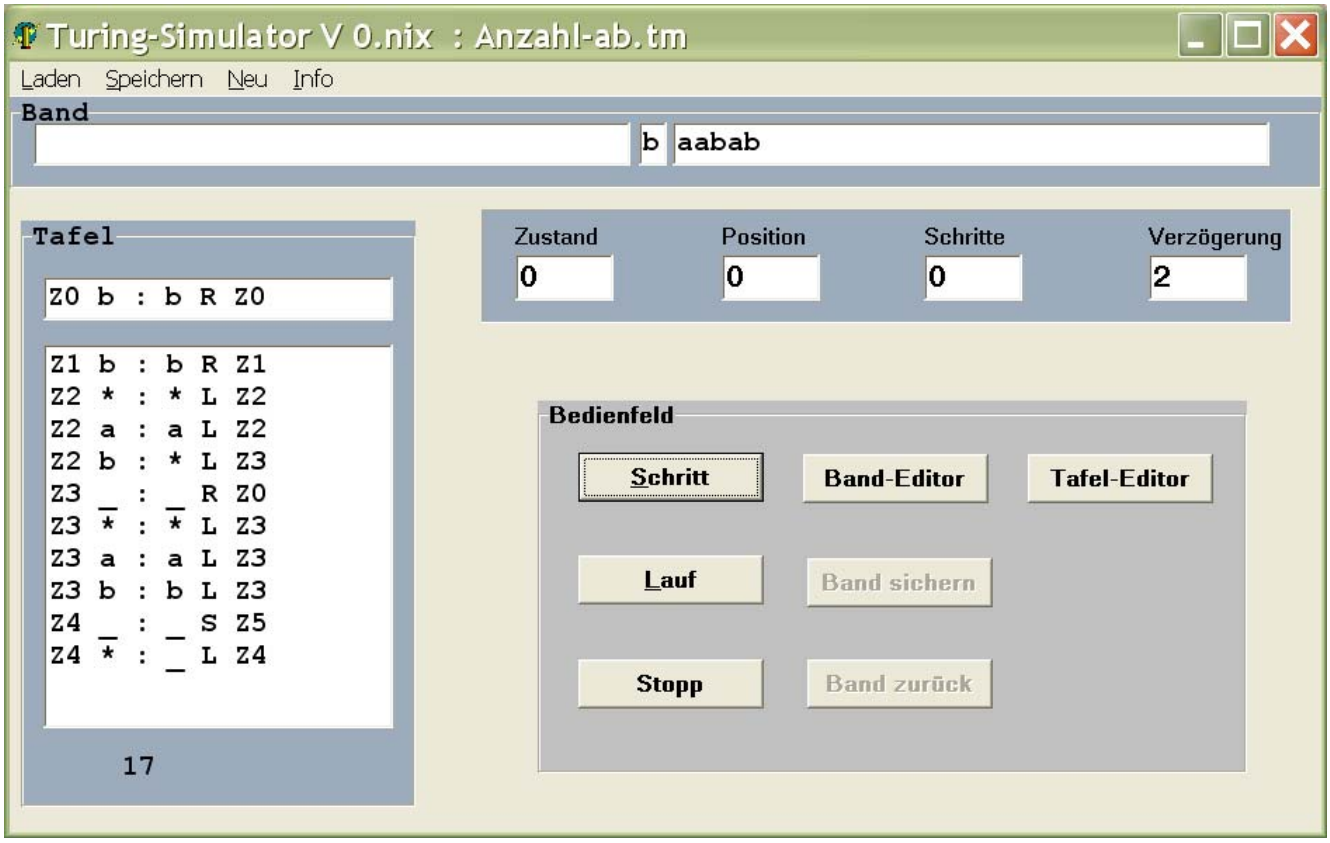
Wenn eine Methode bei Onexecute festgelegt wurde, so soll diese ausgeführt warden.

```
  procedure TMaschine.laden(var f :textfile);
  begin
    Tafel.entleeren;
    readln(f,Z_aktuell);
    readln(f,Schrittzahl);
    readln(f,Verzoegerung);
    Band.laden(f);
    Tafel.laden(f);
  end;

  procedure TMaschine.ruecksetzen;
  begin
    Z_aktuell    := 0;
    Schrittzahl  := 0;
    beendet      := false;
    angehalten   := false;
  end;

  procedure TMaschine.speichern(var f : textfile);
  begin
    writeln(f,Z_aktuell);
    writeln(f,Schrittzahl);
    writeln(f,Verzoegerung);
    Band.speichern(f);
    Tafel.speichern(f);
  end;

  function  TMaschine.Befehl_aktuell : TBefehl;
  begin
    result :=
Tafel.suche_Befehl(Z_aktuell,Band.Zeichen);
  end;

end.
```

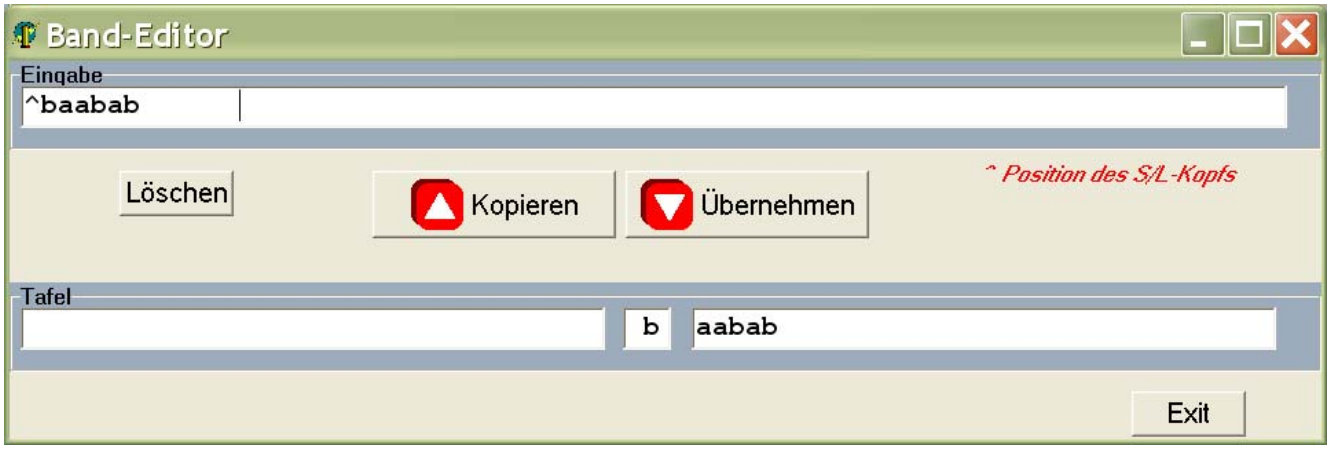
6. Realisierung von View / Controller

Das Hauptformular und die Integration der Teilformulare

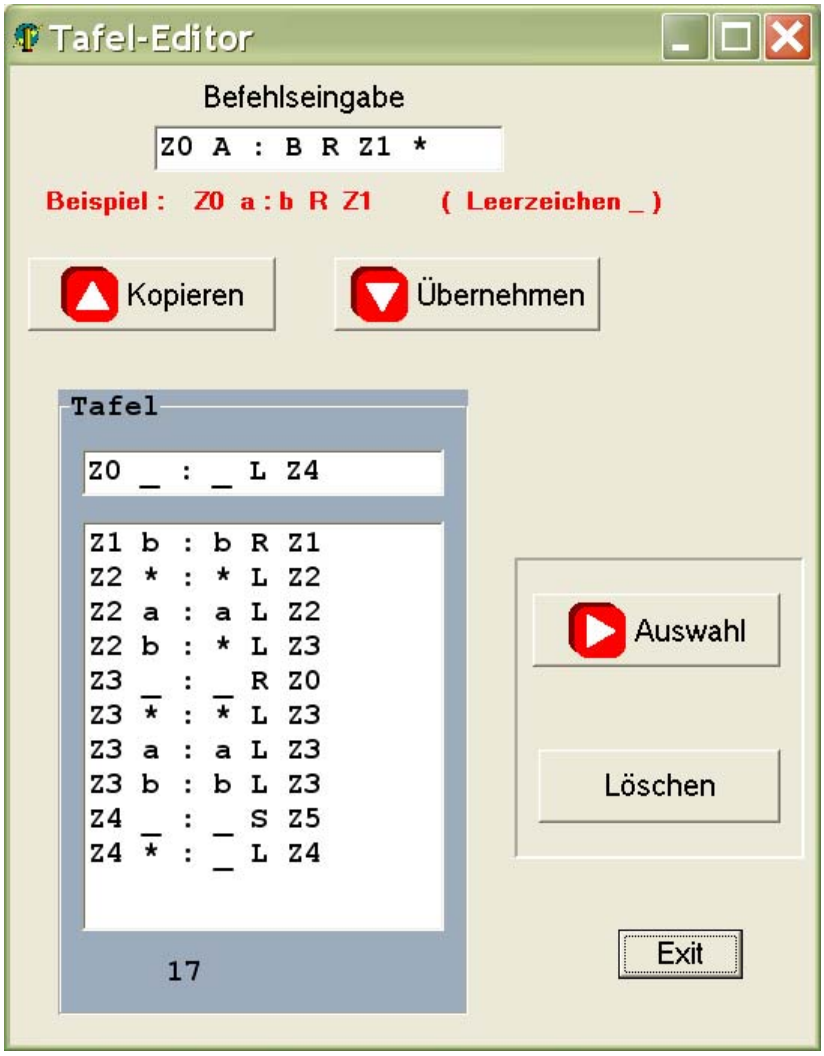
Die Anwendung Turing_Simulator besitzt drei Ansichten / Views. Das Hauptformular (haupt.pas) für die vollständige Turingmaschine,



das Formular für den Band-Editor (UViewBand.pas)



und das Formular für den Tafel-Editor(UViewTafe.pas)



Zur Realisierung von haupt.pas

```
uses
  Windows, Messages, SysUtils, ------------------ Menus,
  UBand, UViewBand, UTafelView, UMaschine, UTafel, UBefehl;

type
  TForm1 = class(TForm)
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    //-------------------
    GBox_Tafel   : TGroupBox;
    Edit_Befehl  : TEdit;
    Memo_Tafel   : TMemo;
    Label_Anzahl: TLabel;
    //-------------------
    GBox_Band     : TGroupBox;
    Edit_LWort    : TEdit;
    Edit_Zeichen  : TEdit;
    Edit_RWort    : TEdit;
```

```
    GBox_Bedienfeld   : TGroupBox;
    Button_Schritt    : TButton;
    Button_Lauf       : TButton;
    Button_Stopp      : TButton;
    Button_Band       : TButton;
    Button_Tafel      : TButton;
    //-------------------
    Panel_Status    : TPanel;
    LEdit_Zustand   : TLabeledEdit;
    LEdit_Position  : TLabeledEdit;
    LEdit_Warten    : TLabeledEdit;
    LEdit_Schritte  : TLabeledEdit;
    //-------------------
    Label_Stop  : TLabel;
    Button_Reset: TButton;
    //-------------------
    MainMenu1  : TMainMenu;
    Laden1     : TMenuItem;
    Speichern1 : TMenuItem;
    Neu1       : TMenuItem;
    Info1      : TMenuItem;
    //-------------------
    procedure FormCreate(Sender: TObject);
    //-------------------
    procedure Button_BandClick(Sender: TObject);
    procedure Button_TafelClick(Sender: TObject);
    procedure Button_SchrittClick(Sender: TObject);
    procedure Button_LaufClick(Sender: TObject);
    procedure Button_ResetClick(Sender: TObject);
    procedure Button_StoppClick(Sender: TObject);
    //-------------------
    procedure LEdit_ZustandExit(Sender: TObject);
    procedure LEdit_WartenExit(Sender: TObject);
    procedure LEdit_PositionExit(Sender: TObject);
    //-------------------
    procedure Laden1Click(Sender: TObject);
    procedure Speichern1Click(Sender: TObject);
    procedure Neu1Click(Sender: TObject);
    procedure Info1Click(Sender: TObject);
  private
    function  Zahl_einlesen(var a : integer; E :TLabeledEdit):
                                                          boolean;
    procedure Band_zeigen;
    procedure Tafel_zeigen;
    procedure Maschine_zeigen;
    procedure verzoegert_anpassen( s :  TObject);
    procedure anpassen( s : TObject);
  end;

var
  Form1    : TForm1;
  Maschine : TMaschine;
```

Die Implementation einiger Methoden

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Maschine:= TMaschine.create;
  Maschine.OnExecute := verzoegert_anpassen;
  Maschine_zeigen;
end;

procedure TForm1.verzoegert_anpassen( s :  TObject);
begin
  wait(Maschine.Verzoegerung);
  Form1.Maschine_zeigen;
end;
```

```
procedure TForm1.Button_BandClick(Sender: TObject);
begin
  Form2.OnBandchange := anpassen;
  form2.show;
end;

procedure TForm1.anpassen( s :  TObject);
begin
  Form1.Maschine_zeigen;
end;
```

```
procedure TForm1.Button_SchrittClick(Sender: TObject);
begin
  Maschine.Schritt;
  Maschine_zeigen;
end;

procedure TForm1.Button_LaufClick(Sender: TObject);
begin
  Maschine.Lauf;
end;
```

Einbinden der Teilformulare

Die Formulare für die Editoren von Band und Tafel entsprechen in etwa den Formularen die beim Testen von UBand und UTafel verwendet wurden.
Sie werden angezeigt und aktiviert durch das Drücken der Knöpfe Band-Editor und Tafel-Editor im Hauptformular.

Welche Schritte sind nötig, um ein Formular aus einer eigenständigen Anwendung in das Projekt einzugliedern ?

1. Dem Projekt muss die Unit des zu integrierenden Formulars hinzugefügt werden. (Projekt -> hinzufügen , Umschalt F11).
   Der Erfolg ist in der Projektdatei ersichtlich.
2. Im Hauptformular, von dem das Teilformular aufgerufen wird, muss eine entsprechende uses – Anweisung erscheinen.
3. Verzweigung zum Teilformular durch Aufruf von form?.show aus dem Hauptformular heraus.
4. Soll auch das Teilformular auf die Variablen des Hauptformulars zugreifen, so muss in der Unit des Teilformulars ein uses-Anweisung auf die Unit des Hauptformulars erfolgen. Dies muss im Implementionsteil stehen.
5. Identifizierung von Objekten des Hauptformulars mit denen des Teilformulars.
   Wurden im Teilformular eigenständige Variablennamen benutzt, so

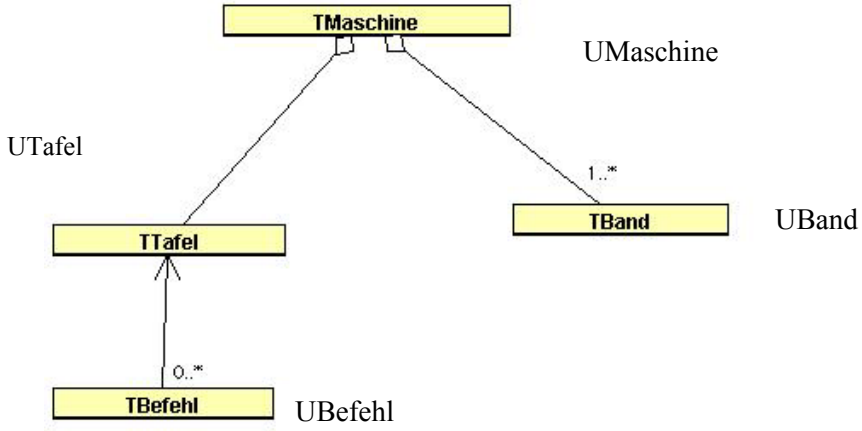
können diese mit Variablen im Hauptformular durch Wertzuweisungen identifiziert werden.

Beispiel :

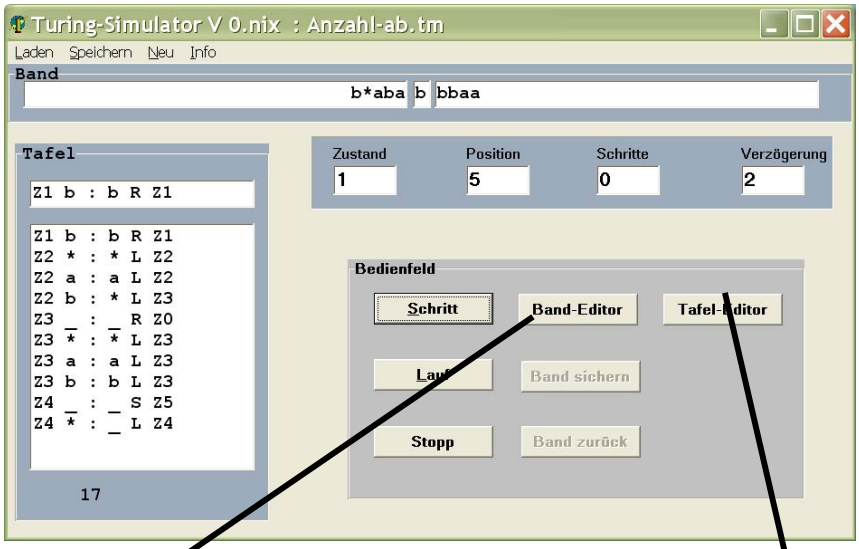
Die Variable Band aus UViewBand.pas wird mit Maschine.Band aus haupt.pas identifiziert.

```
procedure TForm2.FormCreate(Sender: TObject);
begin
  Band := Maschine.Band;
  ausgeben;
  BitBtn_kopierenClick(Sender);
end;
```

Zusammenfassung des Projekts



haupt.pas



UViewBand.pas

ViewTafel.pas

Projekterstellung

Die Projekterstellung könnte in folgenden Phasen ablaufen.

1. Problemformulierung

   Ziel : Spezifikation des Projektes

2. Entwicklung der Grobstruktur der Anwendung

   Zerlegung in die Teilaufgaben (mehrere Formulare)

3. Gemeinsame Formulierung des Modells

   Spezielle Methoden können den jeweiligen Gruppen überlassen werden

4. Aufteilung des Projekts für die Arbeitsgruppen

   Tafel-Editor, Band-Editor, Hauptformular

   Während die Gruppen (Tafel-Editor / Bandeditor) zuerst Primitiv-Versionen erstellen, um schnell zu Testdaten zu kommen, kann die Gruppe mit dem Hauptformular sich um eine ausführliche Ausgestaltung des Formulars kümmern

      Grundlage ist die Modell-Unit.

   Zur Erleichterung der späteren Integration kann jede Gruppe die restlichen Formulare durch Dummy-Formulare ersetzen.

5. Erstellung und Test der Unterformulare durch die Arbeitsgruppen

6. Integration der Formulare in das Hauptprogramm /

   Test der Anwendung

7. Möglichkeiten zur Verbesserung und Erweiterung

Rückgängigmachen der Schritte

Speicherung der durchgeführten Befehle auf einem Stack

Ausdruck der Tafel und eines Verarbeitungsablaufs (Trace)

Maschinen mit mehreren Bändern

Verschiedene Maschinenmodelle

(Kellerautomaten / endliche Automaten/ halbes Turingband)

Aufbau einer Hierarchie von Maschinenmodellen (Vererbung)

Zusammenfassung von Befehlsgruppen durch Jokerzeichen

Darstellung der Befehlsverarbeitung in einem Übergangsgraphen

Modell bleibt / View wechselt