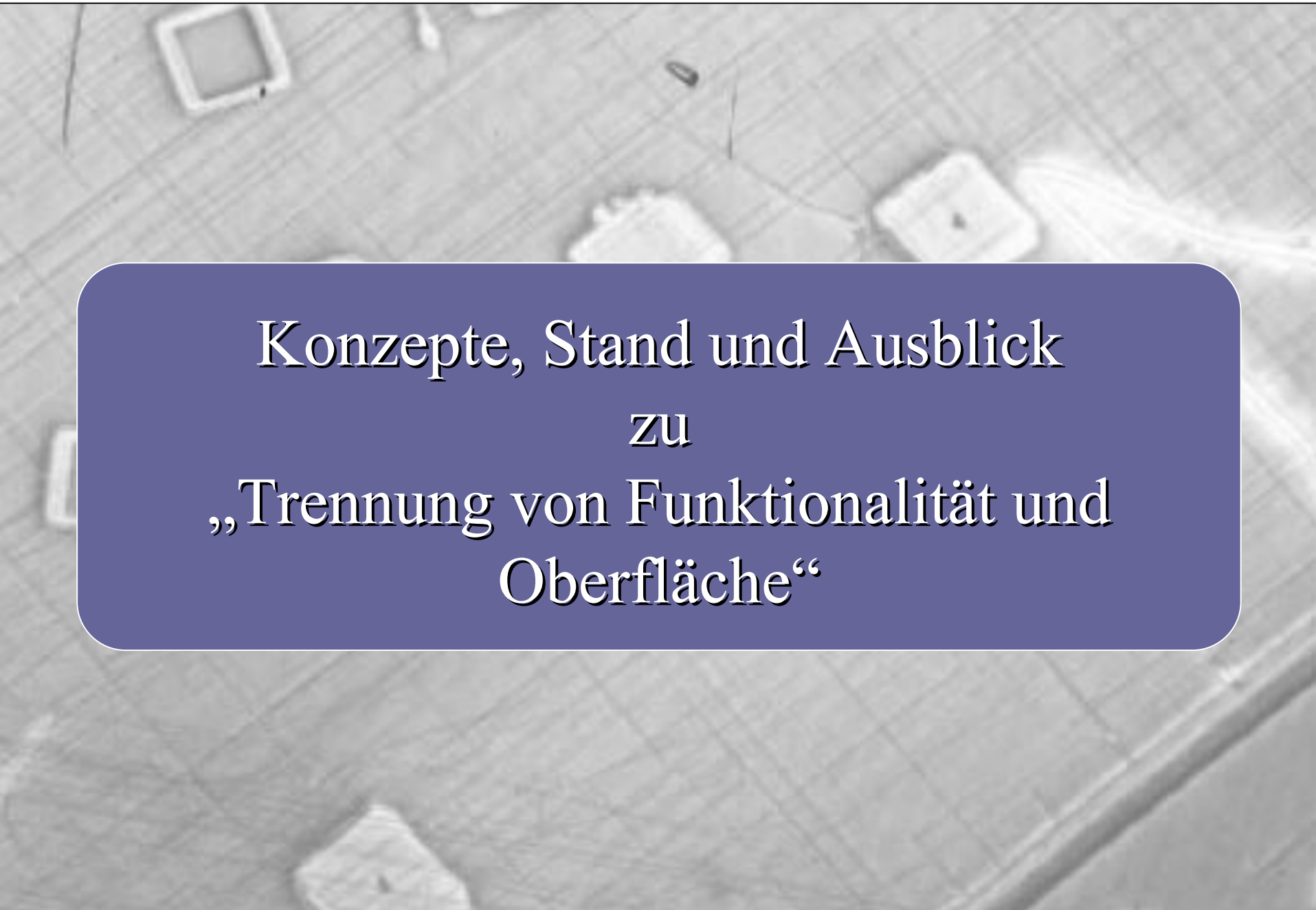


Konzepte, Stand und Ausblick zu „Trennung von Funktionalität und Oberfläche“

1. Kommunikation Funktionalität - Oberfläche

- 1.1 Polling - Verfahren
- 1.2 Pushing – Verfahren
- 1.3 Vergleich Polling vs. Pushing (π-Berechnung)

2. Neuentwurf „Manuelle Justage“

- 2.1 Analyse-Definition
- 2.2 Reverse-Engineering
- 2.3 Design / Implementation
- 2.4 Test

3. Re-Engineering: Subsystem Motorsteuerung

- 3.1 Schwerpunkte
- 3.2 entstandene Dokumente

4. Re-Engineering: Subsystem Ablaufsteuerung

- 4.1 Schwerpunkte
- 4.2 entstandene Dokumente

5. Re-Engineering: Subsystem Oberfläche

- 5.1 neue Klassen
- 5.2 entstandene Dokumente
- 5.3 neue Vererbungshierarchie

6. Neues Subsystem Utilities

- 6.1 U_TIMER
- 6.2 U_FILES
- 6.3 U_VALUES

7. aktuelle Arbeiten & Ausblick

1.1 Polling - Verfahren

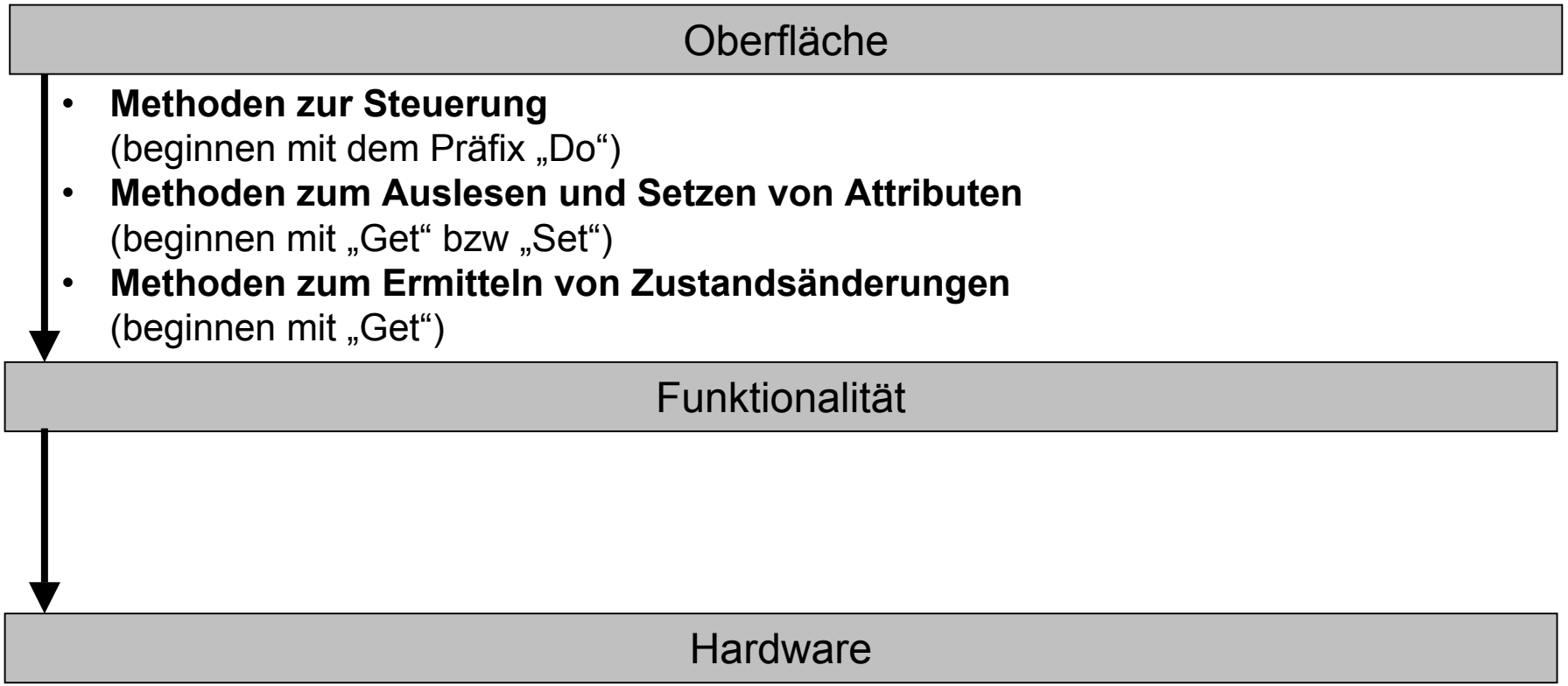


Abb.1 „Schichten-Kommunikation bei *polling*“ (Quelle: nach Balzert)

1.2 Pushing - Verfahren

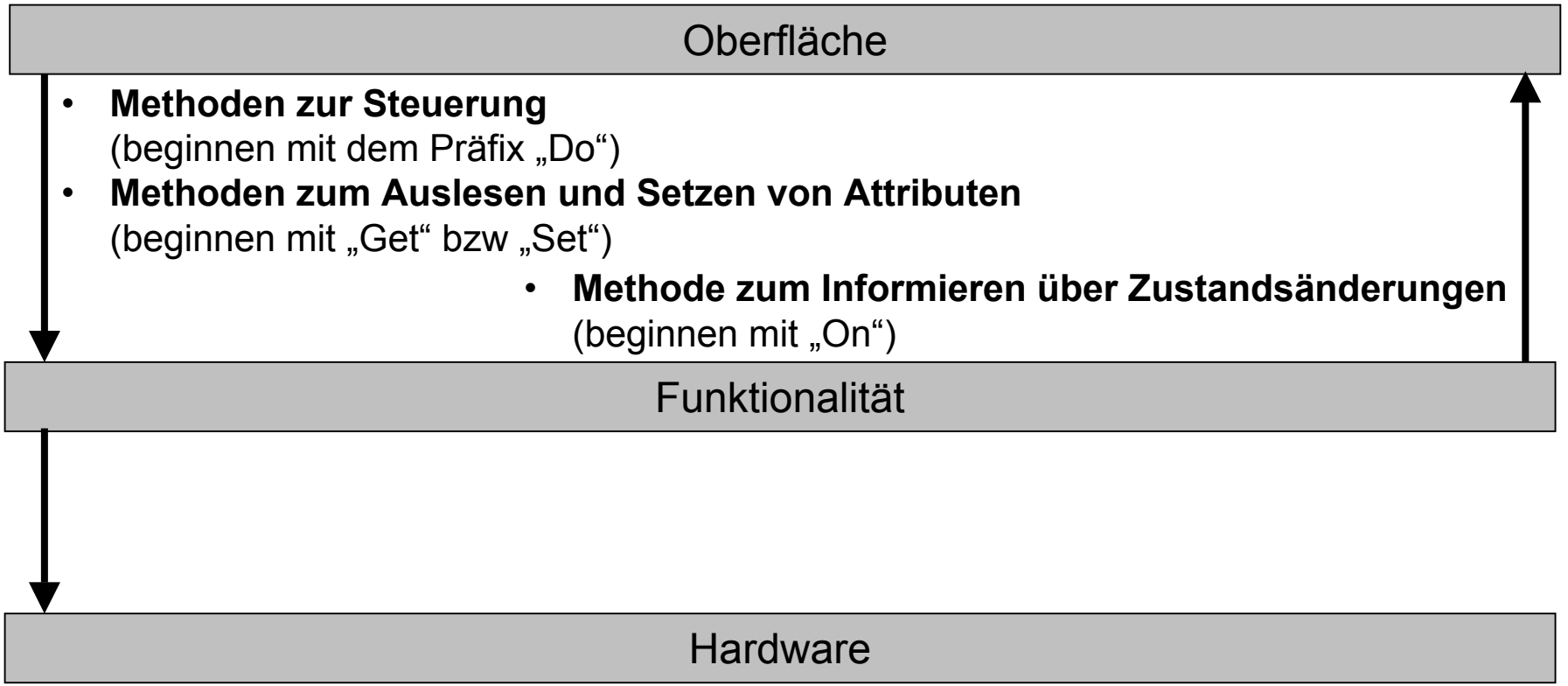


Abb.2 „Schichten-Kommunikation bei *pushing*“ (Quelle: selbst)

1.3 Vergleich Polling vs. Pushing (π -Berechnung)

1996 hat David H. Bailey,
zusammen mit Peter Borwein und Simon Plouffe,
eine neue Formel für π entdeckt:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left[\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right]$$

Abb.3 „Formel zur Berechnung von π“ (Quelle: http://www.nersc.gov/~dhbailey/)

1.3 Polling vs. Pushing

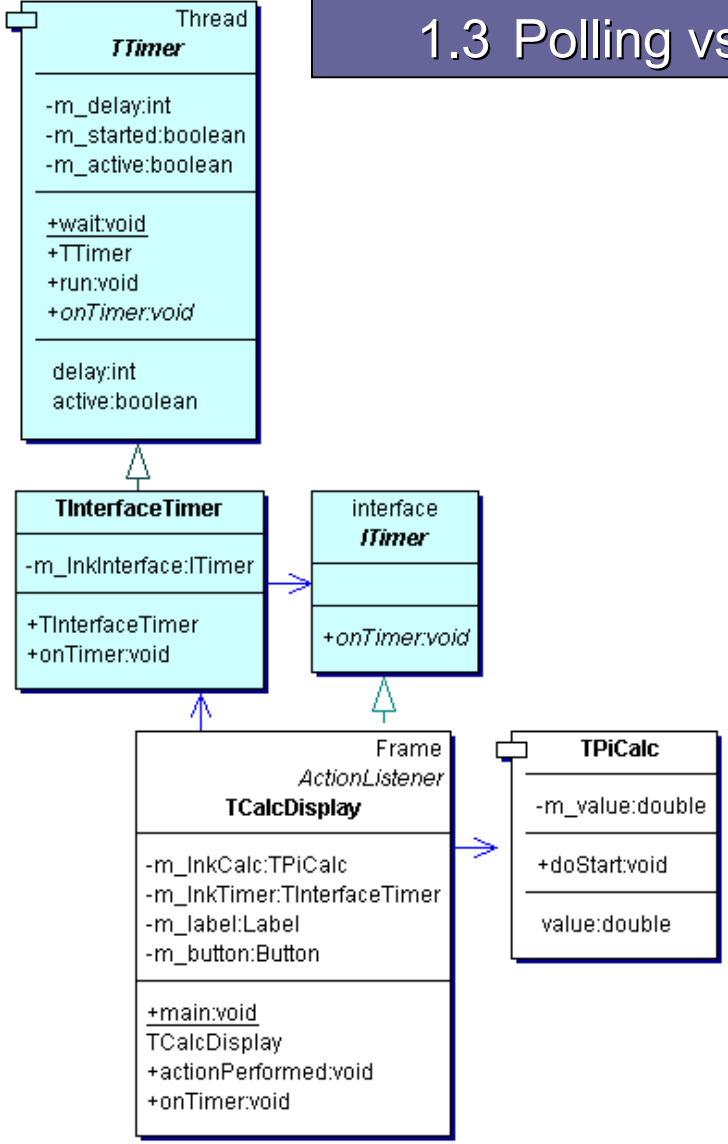


Abb.4 „UML-Klassendiagramm *polling*“ (Quelle: Together®, Version 6.0)

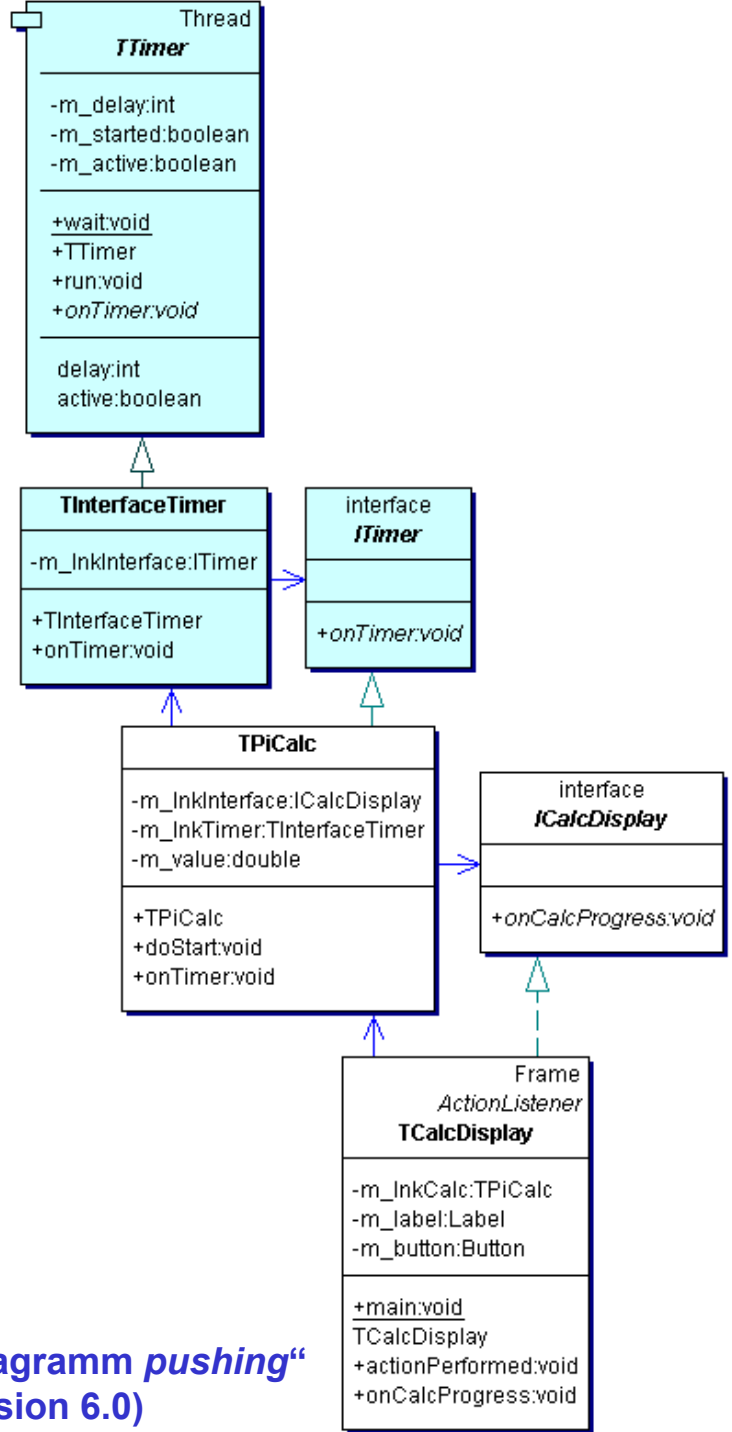


Abb.5 „UML-Klassendiagramm *pushing*“ (Quelle: Together®, Version 6.0)

Abb.5 „*polling* vs. *pushing* – Funktionalität“ (Quelle: selbst)

polling2

```
//#####################################################################
//                                                IPiCalc
//#####################################################################
class TPiCalc {
  private double m_value= 0;



  public double getValue() {
    return m_value;
  }



////////////////////////////////////// Berechnung ////////////////

  public void doStart() {
    m_value= 0;
    double lastValue= -1;

    for (int lastChange= 0, k= 0;
         (lastChange<5) && (k<Integer.MAX_VALUE); k++)
    {
      m_value+= ( 1/ Math.pow( 16, k) ) *
                (
                 ( 4/(double)(8*k+1) ) - ( 2/(double)(8*k+4) ) -
                 ( 1/(double)(8*k+5) ) - ( 1/(double)(8*k+6) )
                );
      if ( m_value != lastValue ) {
        lastValue= m_value;
        lastChange= 0;
      } else lastChange++;
      TTimer.wait( 500 ); //Berechnung abbremsen
    }


  }




} //class TPiCalc; LOC: 29; LOCR: 22
```

pushing2

```
//#####################################################################
//                                                ICalcDisplay
//#####################################################################
interface ICalcDisplay {
  void onCalcProgress(double aNewValue);
} //ICalcDisplay; LOC: 7; LOCR: 3

//#####################################################################
//                                                IPiCalc
//#####################################################################
class TPiCalc implements ITimer {
  private double m_value= 0;
  private ICalcDisplay m_lnkInterface; //Ereignisse an diesen Obj.
  private TInterfaceTimer
    m_lnkTimer= new TInterfaceTimer( 500, this );



  public TPiCalc( ICalcDisplay aInterface ) {
    m_lnkInterface= aInterface;
  }

////////////////////////////////////// Berechnung ////////////////

  public void doStart() {
    m_value= 0;
    double lastValue= -1;
    m_lnkTimer.setActive( true );
    for (int lastChange= 0, k= 0;
         (lastChange<5) && (k<Integer.MAX_VALUE); k++)
    {
      m_value+= ( 1/ Math.pow( 16, k) ) *
                (
                 ( 4/(double)(8*k+1) ) - ( 2/(double)(8*k+4) ) -
                 ( 1/(double)(8*k+5) ) - ( 1/(double)(8*k+6) )
                );
      if ( m_value != lastValue ) {
        lastValue= m_value;
        lastChange= 0;
      } else lastChange++;
      TTimer.wait( 500 ); //Berechnung abbremsen
    }
    m_lnkTimer.setActive( false );
    onTimer(); //letzte Bildschirmaktualisierung steht noch aus
  }

  public void onTimer() {
    m_lnkInterface.onCalcProgress( m_value );
  }

} //class TPiCalc; LOC: 39; LOCR: 30
```

```
//#################################################################
//                                                  TCalcDisplay
//#################################################################
public class TCalcDisplay extends Frame implements ActionListener,
                                                    ITimer
{
  //Objekt (zum Delegieren der Funktionalität)
  private TPiCalc m_lnkCalc= new TPiCalc();
  private TInterfaceTimer
    m_lnkTimer= new TInterfaceTimer( 500, this );
  private Label m_label= new Label( "not yet calculated" );
  private Button m_button= new Button( "start" );

  public static void main( String args[] ) {
    new TCalcDisplay().show();
  }

  TCalcDisplay() {
    super("Pi calculator");
    setLayout( new BorderLayout() );
    add( BorderLayout.CENTER, m_label );
    add( BorderLayout.SOUTH, m_button );
    m_button.addActionListener( this );
    setSize( 200, 85 );
  }

////////////////////////////////////// auf Ereignisse reagieren //

  public void actionPerformed( ActionEvent e ) {
    if ( e.getSource()==m_button ) {
      m_button.setEnabled( false );
      m_lnkTimer.setActive( true );
      m_lnkCalc.doStart();
      m_lnkTimer.setActive( false );
      m_button.setEnabled( true );
      onTimer(); //letzte Bildschirmaktualisierung steht noch aus
    }
  }

  public void onTimer() {
    m_label.setText( Double.toString( m_lnkCalc.getValue() ) );
  }
} //TCalcDisplay; LOC: 49; LOCR: 30
```

polling2; LOC: 79; LOCR: 52

```
//#################################################################
//                                                  TCalcDisplay
//#################################################################
public class TCalcDisplay extends Frame implements ActionListener,
                                                    ICalcDisplay
{
  //Objekt (zum Delegieren der Funktionalität); Ereignisse an uns
  private TPiCalc m_lnkCalc= new TPiCalc( this );


  private Label m_label= new Label( "not yet calculated" );
  private Button m_button= new Button( "start" );

  public static void main( String args[] ) {
    new TCalcDisplay().show();
  }

  TCalcDisplay() {
    super("Pi calculator");
    setLayout( new BorderLayout() );
    add( BorderLayout.CENTER, m_label );
    add( BorderLayout.SOUTH, m_button );
    m_button.addActionListener( this );
    setSize( 200, 85 );
  }

////////////////////////////////////// auf Ereignisse reagieren //

  public void actionPerformed( ActionEvent e ) {
    if ( e.getSource()==m_button ) {
      m_button.setEnabled( false );

      m_lnkCalc.doStart();

      m_button.setEnabled( true );

    }
  }

  public void onCalcProgress( double aNewValue ) {
    m_label.setText( Double.toString( aNewValue ) );
  }
} //TCalcDisplay; LOC: 44; LOCR: 26
```

pushing2; LOC: 92; LOCR: 59

LOC – LinesOfCode
LOCR – LinesOfCodeReal (ohne Leer- und Kommentarzeilen)

Abb.6 „*polling* vs. *pushing* – Oberfläche“ (Quelle: selbst)

2.1 Analyse-Definition

- ???/Layoutkonventionen
- Manual Adjustment/Errors/Bewertung-v1.7
- Manual Adjustment/Errors/ursprüngliche Manuelle Justage-v1.0
- Manual Adjustment/Analysis and Definition/Gen.Descr./Pflichtenheft-v2.1
- Manual Adjustment/Test/Test Cases/neue Manuelle Justage-v1.4

2.2 Reverse-Engineering

- User Interface/Design/Gen.Description/RevE GUI-Baisklassen-v1.2
- Manual Adjustment/Design/Gen.Description/RevE Manuelle Justage-v1.6
- Steering Motors/Design/Gen.Description/RevE Motorsteuerung-v1.5
- Flow Control/Design/Gen.Description/RevE Ablaufsteuerung-v1.9
- Windows-Ressourcen/Design/Gen.Description

2.3 Design / Implementation

Subsystem: Manuelle Justage

- neues Verzeichnis: ManJust
- getrennte Implementation (derzeit Polling-Variante):

Funktionalität:	MJ_Funk.cpp	MJ_Funk.h
Oberfläche:	MJ_GUI.cpp	MJ_GUI.h

Funktionalität:

`Get`/ `Set`-Methoden	→	lesen/ schreiben von Parametern
`Do`-Methoden	→	Aktionen durchführen (z.B. Antriebsbewegung durchführen)
`Is`/ `Has`-Methoden	→	Statusinformationen
`CanDo`-Methoden	→	Prüfung, ob `Set`/ `Do`-Methoden durchführbar (z.B. Prüfung ob Bewegung durchführbar)

- bei `Set`/ `Do`-Methoden Statusinformation als Rückgabewert
  bei `Is`/ `Has`-Methoden per Referenzparameter
- beim Setzen, sofortige Sicherung der Antriebsparameter in den ini-Dateien
- Benutzung von `TMList` und `TMotor` zur Motorsteuerung (ohne C-Interface)

2.3 Design / Implementation

Oberfläche:

- Problem der Verwaltung der drei Teilbereiche
  ➔nicht jedes Steuerelement separat ansprechen, sonst LOC-Explosion
- `enum`-Auflistung für die Steuerelemente in einem Teilbereich
- Steuerelemente können durch diesen `enum` und die Nummer des Teilbereichs eineindeutig angesprochen werden (Auflösung in Ress.-Id)
- Bildung von Klassen von Steuerelementen (z.B. Positionsangaben) in einem Teilbereich
- jeweils zwei Methoden** (zum Füllen und Freigeben bzw. Sperren) der enthaltenden Steuerelemente
- Problem mit der Ausführungsgeschwindigkeit bei der Vielzahl an Steuerelementen
  ➔ Startzeit von 5 Sekunden bei Celeron 1GHz
- **-Methoden werden indirekt per Windowsbotschaft aufgerufen
  ➔ damit Verarbeitung im Hintergrund
  ➔ Startzeit von ½ Sekunde, danach füllen und freigeben/ sperren der StE

TInterfaceTimer →

← TModalDlg

TManJustage →

← TManJustageDlg

← TMotorOffsetDlg

← TPsdOffsetDlg

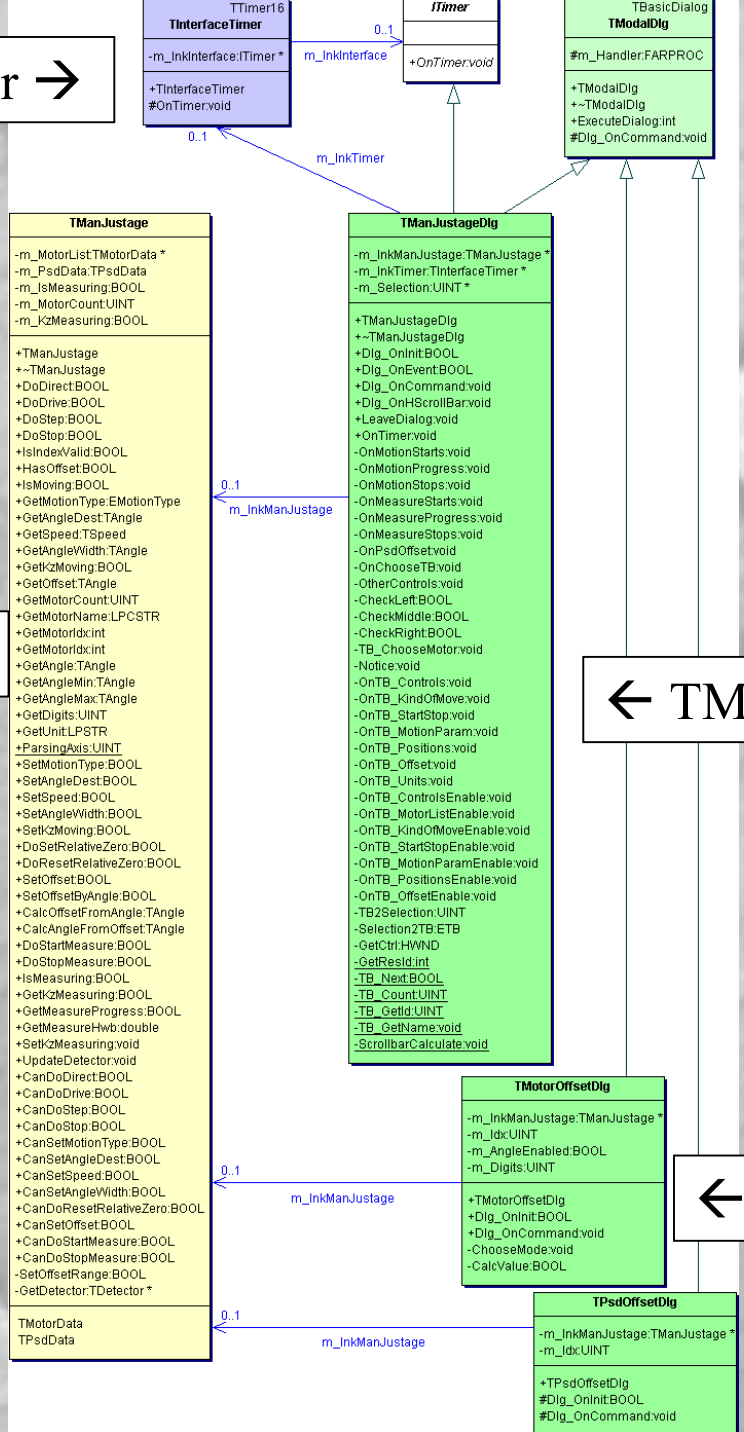


Abb.7 „UML-Klassendiagramm der neuen Manuellen Justage“ (Quelle: Together®, Version 6.0)

2.4 Test

Idee: separates Testen der Funkt. und Oberfl.

Funktionalität:

- CTE-Diagramm (CTE-XL) für vollständigen Test erstellt
- aufgrund der hohen Komplexität der MJ_GUI wurde Test-GUI implementiert
- Vorteile: einfacher Code, Vorab-Bewertung des Funkt.-Interface möglich
- Testverlauf:
  - Grundfunktionen Get/ Set
  - Funktionen nur mit Abh. zu Grundfunktionen
  - komplexe Funktionen (Antriebsbewegungen)
- Designidee: jeden Memberzugriff auch intern über Get/ Set-Methoden

3.1 Schwerpunkte Re-Engineering Motorsteuerung

- stärkere Datenkapselung (Attribute public → protected)
- alle FRIEND Deklarationen entfernt
- TMotor exportiert
- Instanziierungsmöglichkeiten von TMList eingeschränkt
- kommentiert und strukturiert
- Wertebereich von Parametern und Rückgabewerten korrigiert
- Toten Code mit Datum auskommentiert

3.2 entstandene Dokumente

- Steering Motors/Design/Gen.Description/RevE Motorsteuerung-v1.5
- Steering Motors/Design/Gen.Description/RE Motorsteuerung-v1.0
- Steering Motors/Design/Gen.Description/RE Motorsteuerung-v1.1

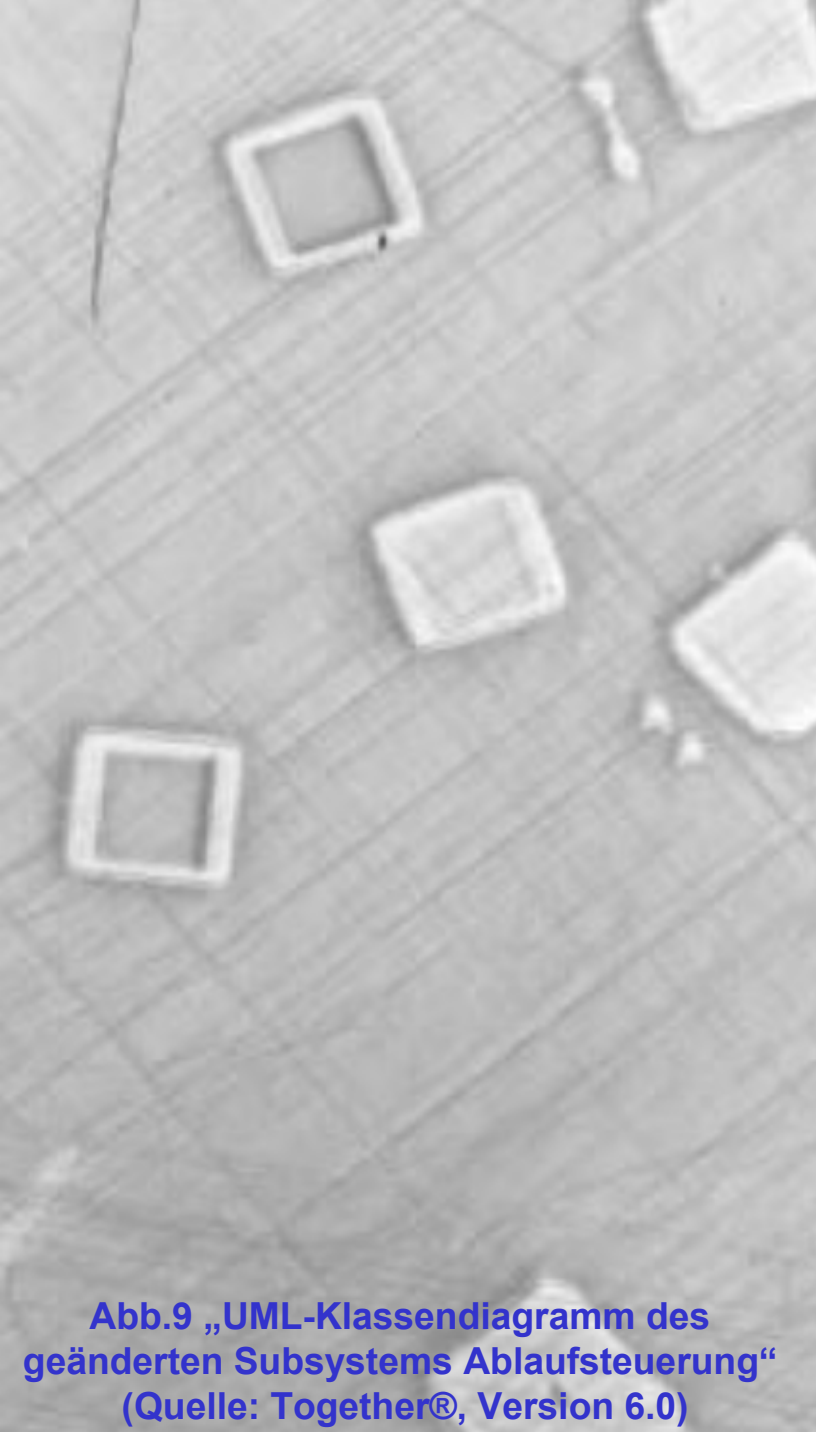
Abb.8 „UML-Klassendiagramm des geänderten Subsystems Motorsteuerung“ (Quelle: Together®, Version 6.0)

4.1 Schwerpunkte Re-Engineering Ablaufsteuerung

- Member umsortiert und “Sinneinheiten“ zusammengefasst
- einheitlich strukturiert und formatiert
- Wertebereich von Parametern und Rückgabewerten korrigiert
- Toten Code mit Datum auskommentiert
- TSteering
  - stärkere Datenkapselung (public → protected): neue Accessor-/ Mutatormethoden
  - vormals globale Symbole in TSteering aufgenommen (meist static)
  - alle FRIEND Deklarationen entfernt
  - neue POLLING-Methode zum Auslesen der Status-/ Fehlerinformationen
  - Absturzursache und sonstige Fehler beseitigt
- TCmd und abgeleitete Klassen
  - neue Methode zum Ermitteln des Namens des Kommandos

4.2 entstandene Dokumente

- Flow Control/Design/Gen.Description/RevE Ablaufsteuerung-v1.9
- Flow Control/Design/Gen.Description/RE Ablaufsteuerung-v1.0
- Flow Control/Design/Gen.Description/RE Ablaufsteuerung-v1.1



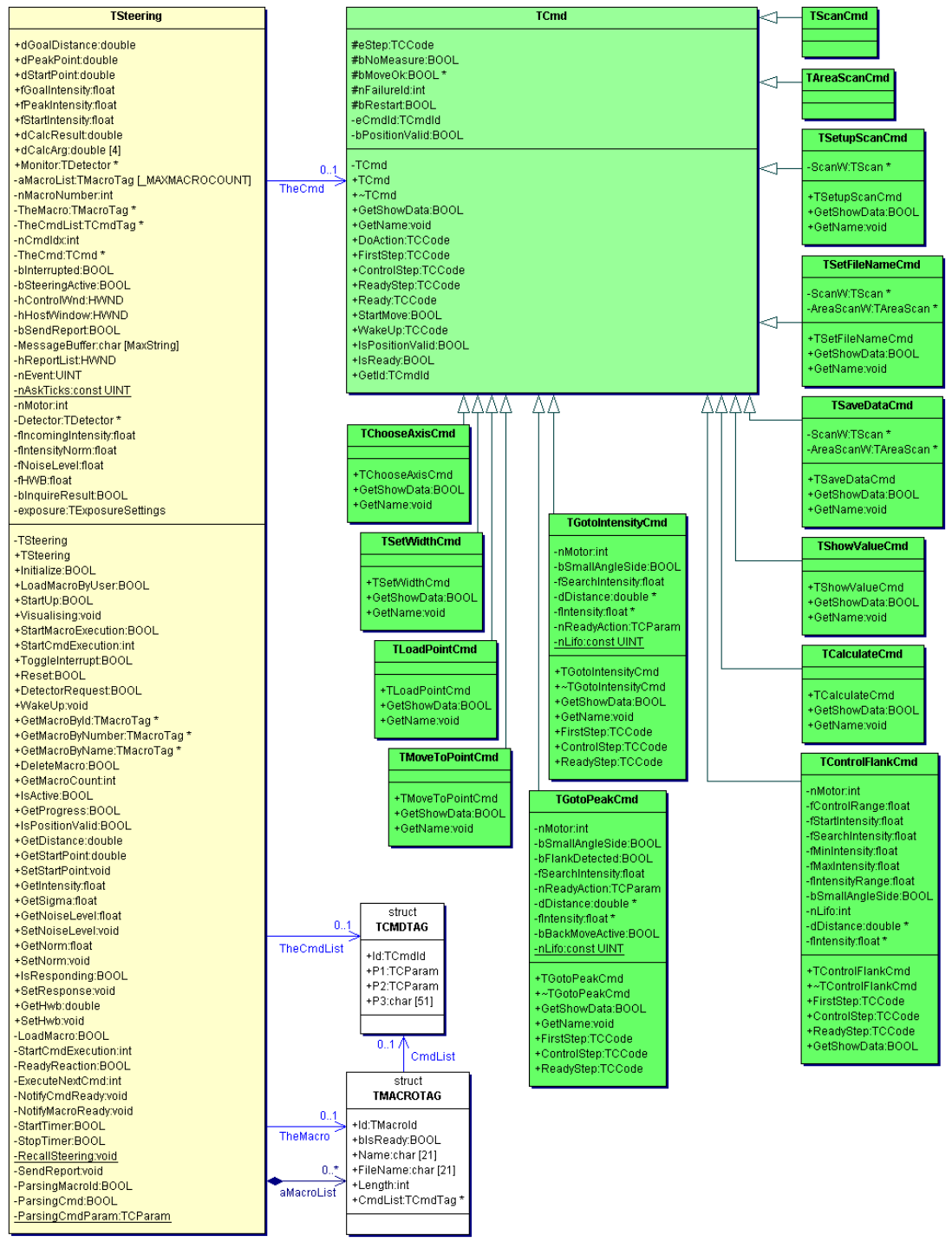


Abb.9 „UML-Klassendiagramm des geänderten Subsystems Ablaufsteuerung“ (Quelle: Together®, Version 6.0)

5.1 Schwerpunkte Re-Engineering Oberfläche

- Problem: nur das letzte erzeugte Fenster bekommt alle Nachrichten
- TBasicWindow
  - neue Basisklasse für alle Fenster
  - Zuordnung zwischen Fensterhandle und -objekt
  - Veraltung von Tastenkombinationen (nur ab Win32)
- THotKey
  - neue Klasse für Tastenkombinationen, aber erst ab Win32 verfügbar
- TBasicDialog (abgeleitet von TBasicWindow)
  - neue Basisklasse für alle Dialogfenster
  - beinhaltet Funktionalität zum lesen/ setzen von Steuerelementen
  - davon abgeleitet TModalDlg, TModelessDlg

5.2 entstandene Dokumente

- User Interface/Design/Gen.Description/RevE GUI-Baisklassen-v1.2
- User Interface/Design/Gen.Description/RE GUI-Baisklassen-v1.0

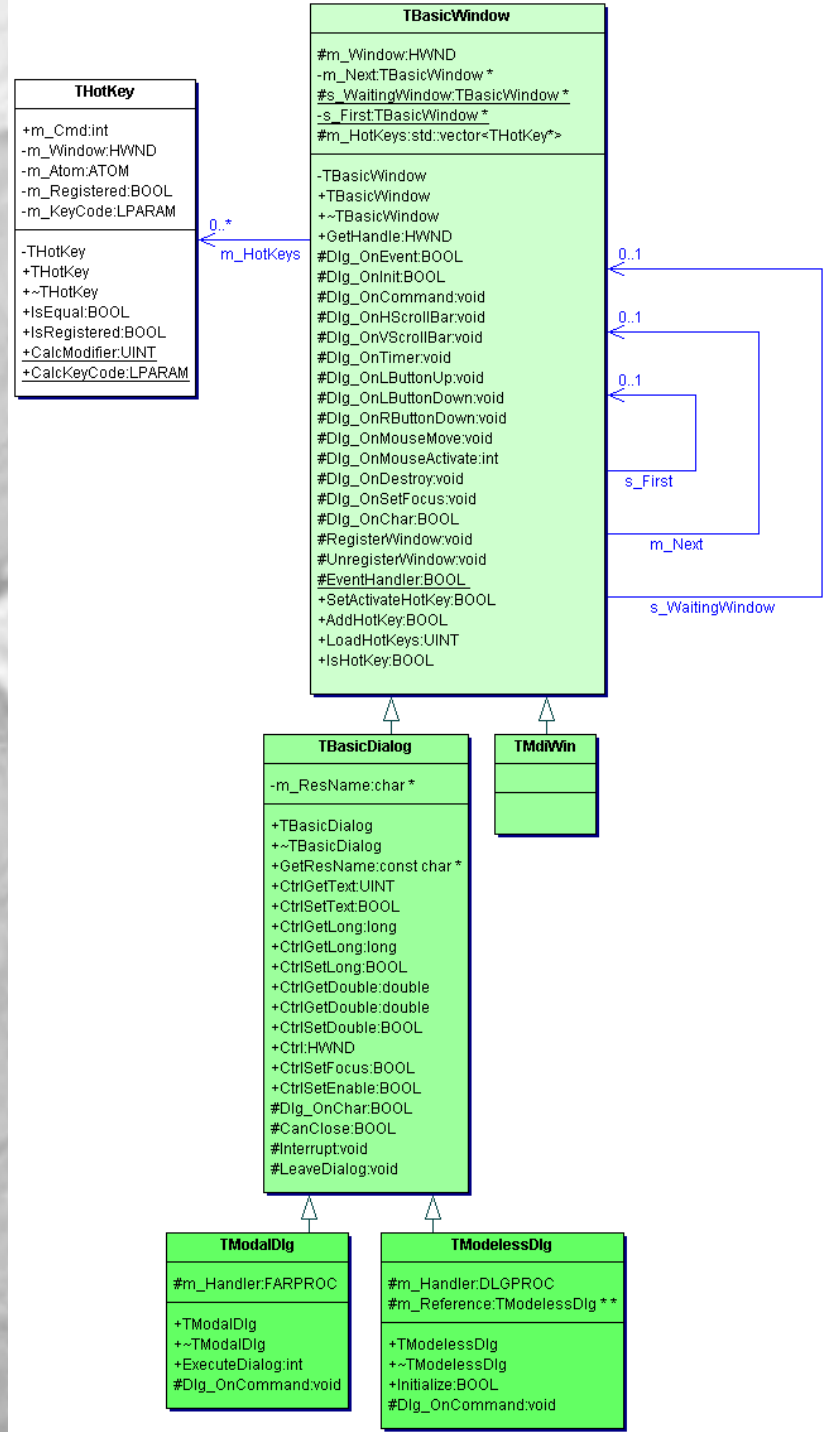


Abb.10 „UML-Klassendiagramm des neuen Subsystems Oberfläche“ (Quelle: Together®, Version 6.0)

6.1 U_TIMER

- Timerklassen für alle Betriebssysteme 16bit, NT
- völlig oberflächenunabhängig und mit identischem Interface

6.2 U_FILES

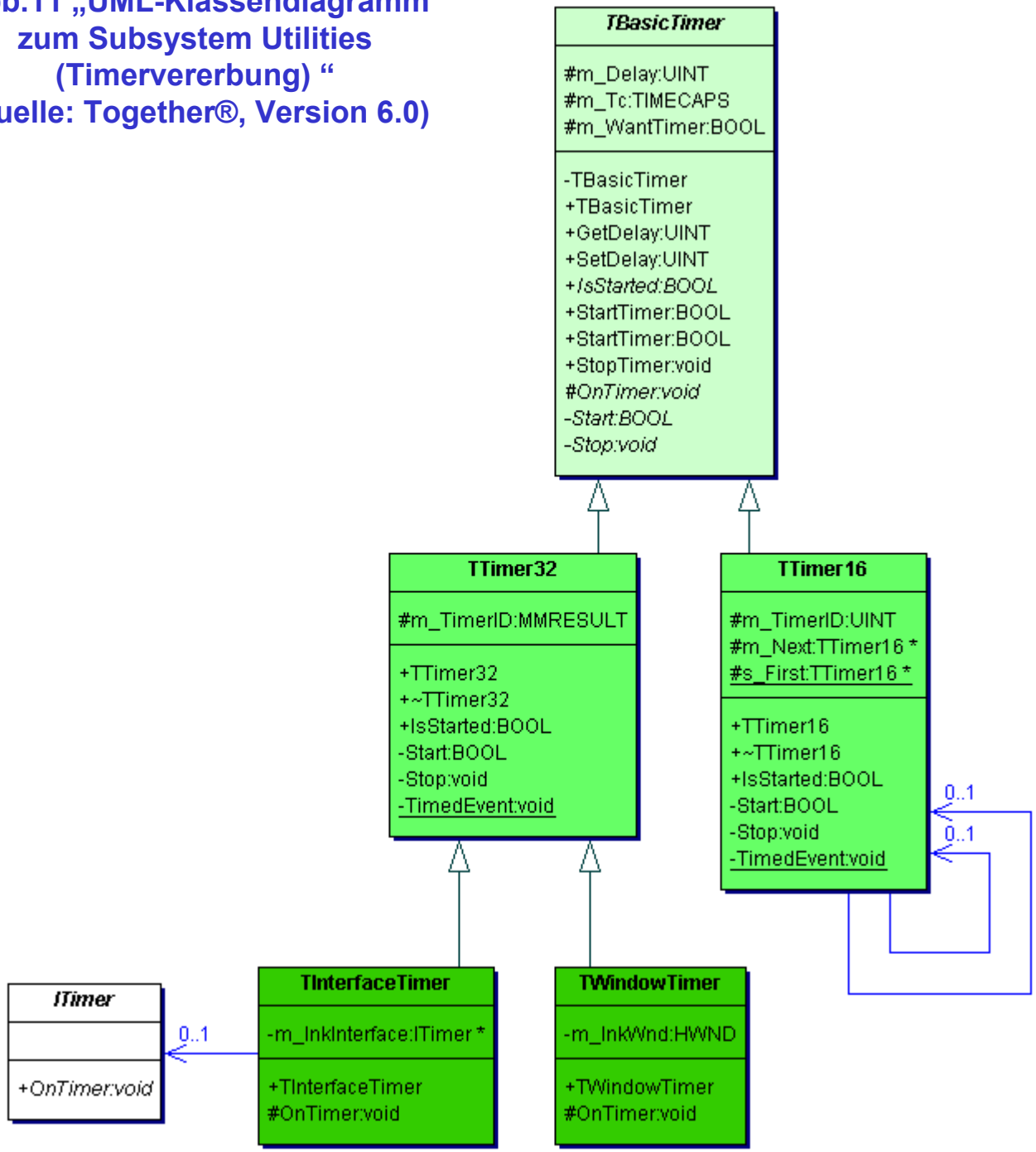
- globale Methoden zum lesen/ schreiben von String/ Double/ Long-Werten aus/ in ini-Dateien
- Klasse TTxtRead zum Lesen von Textdateien (Unix/ PC-Zeilenumbrüche) eingesetzt zum Lesen der mak-Dateien (Ablaufsteuerung)

6.3 U_VALUES

globale Methoden:

- zum Runden von Double-Werten
- zur Zahlenkonvertierung zwischen String und Long / String und Double

Abb.11 „UML-Klassendiagramm zum Subsystem Utilities (Timervererbung) “ (Quelle: Together®, Version 6.0)



7. aktuelle Arbeiten

- Sammlung und Auswahl von Kommunikations-Konzepten
- Testfälle und Test der Manuelle Justage GUI

7. Ausblick

- Vorstellung des Neuentwurfs bei der Physik
- Re-Engineering (schnelle Trennung): Alte Manuelle Justage und Topographie
- Untersuchung ob Regelwerk zur schnellen Trennung ableitbar