

Technische Universität Ilmenau



## Studienarbeit

Vergleich der Programme „Vee“ und „TestPoint“ beim Zugriff auf externe  
Messgeräte

Norman Krebs

Januar 2003

# Inhaltsverzeichnis

<b>1</b>	<b>Bemerkungen zur Aufgabenstellung</b>	<b>2</b>
<b>2</b>	<b>Grundlagen</b>	<b>3</b>
2.1	Datenaufnahme . . . . .	3
2.2	Datenverarbeitung . . . . .	3
2.3	Der prinzipielle Aufbau . . . . .	4
2.4	Die Arbeitsweise im Rechner . . . . .	4
2.5	GPIB . . . . .	6
2.6	VISA . . . . .	6
<b>3</b>	<b>Der Versuchsaufbau</b>	<b>8</b>
<b>4</b>	<b>Vee Pro</b>	<b>9</b>
4.1	Die Installation . . . . .	9
4.2	Hardware-Zugriff . . . . .	9
4.3	Die Vee-IDE . . . . .	9
4.4	Die Sprache Vee . . . . .	10
4.4.1	Multitasking . . . . .	11
4.4.2	Rekursion und Rückkopplung . . . . .	11
4.4.3	Programmbeispiele . . . . .	12
4.5	Das Testprogramm . . . . .	18
<b>5</b>	<b>TestPoint</b>	<b>20</b>
5.1	Die Installation . . . . .	20
5.2	Hardware-Zugriff . . . . .	20
5.3	Die Testpoint-IDE . . . . .	20
5.4	Die Sprache TestPoint . . . . .	20
5.4.1	Variablen . . . . .	22
5.4.2	Multitasking . . . . .	22
5.4.3	Programmbeispiele . . . . .	22
5.5	Das Testprogramm . . . . .	26
<b>6</b>	<b>Fazit</b>	<b>27</b>
<b>7</b>	<b>Anhang</b>	<b>28</b>

## 1 Bemerkungen zur Aufgabenstellung

Ziel der Studienarbeit war ein Vergleich zwischen den Programmen „Vee Pro“ und „TestPoint“ hinsichtlich ihrer Eignung für einen exemplarischen Versuchsaufbau. Dabei sollte ein externes Messgerät an einen PC angeschlossen werden, um dann mit den beiden Programmen eine Verbindung zu dem Messgerät herzustellen.

Folgende Aufgaben sollte das jeweilige Programm erfüllen:

- Ansteuern des Messgerätes
- Auslesen von Messdaten
- Darstellen der Messdaten auf dem PC-Bildschirm
- Abspeichern der Messdaten

Da beide Programme über eigene Makrosprachen verfügen, war es Bestandteil der Aufgabenstellung, sich in die Programmierung der Sprachen einzuarbeiten.

## 2 Grundlagen

Um den aktuellen Zustand eines beliebigen Systems beschreiben zu können, wird sein Verhalten zunächst auf möglichst wenige Zustandsgrößen reduziert, die dann für die Betrachtung ausreichen sollten und auch prinzipiell messbar sind. Aber trotz der Reduktion des Systems fällt bei der Messung seiner Betriebsparameter eine sehr grosse Datenmenge an, die vor der Einführung der modernen Datenverarbeitungstechnik oft gar nicht aufnehm- und verarbeitbar war.

### 2.1 Datenaufnahme

Die Datenaufnahme geschieht mit dem eigentlichen Messmittel (Messfühler, Sensor u.a.) sowie einem Messgerät, welches die Daten wandelt und so aus der reinen Messgrösse ein Messsignal bereitet, das darstellbar und ohne Verlust logisch übertragbar wird. Manche Geräte verfügen auch über eine eigene Datenspeicherung und -auswertung (Pufferung, Filterung). Oft werden die Messwerte mit hohen Frequenzen abgetastet. Die entstehende Datenmenge ist dann nicht übertragbar und wird durch eine Mittelwertbildung reduziert, wobei auch Störeffekte ausgefiltert werden. Da die Messgeräte aber nicht über ausreichenden Platz zur Speicherung aller anfallenden Daten verfügen, wird oft das Messgerät mit einem PC/Workstation-Rechner verbunden, der seinerseits die grob aufbereiteten Daten aufnimmt und zunächst nur speichert. Dies bietet sich an, weil heutige PCs über eine sehr vielfältige Programmierbarkeit und einen flexiblen Hardwareaufbau verfügen. Verschiedene Hardwareschnittstellen können eingebaut und angesprochen werden. Weiterhin ist es auf diese Weise möglich, die Daten in digitaler Form aufzunehmen, was die Nachbearbeitbarkeit und auch die fehler- und verlustfreie Kopierbarkeit ermöglicht. Da der PC als Datenaufnahmegesetz räumlich von dem(dem) Messgerät(en) getrennt ist, kann der PC in einem anderen Raum oder sogar in einem anderen Gebäude aufgestellt werden, z.B. um den Bediener von schädlichen Einflüssen in der unmittelbaren Umgebung des Messplatzes abzuschirmen. Wenn ein PC von mehreren, räumlich auseinanderliegenden Messgeräten Daten empfangen soll, ist es sogar zwangsläufig so, dass der PC von den jeweiligen Messgeräten getrennt ist. Um nun die Messungen effektiv durchführen zu können, muss der PC neben der Datenaufnahme noch die Funktion einer externen Bedieneinheit für die Messgeräte ausführen. Die Bedienung kann so sehr bequem und schnell erfolgen und die Bedienschnittstelle des jeweiligen Messgerätes oft noch erweitern.

### 2.2 Datenverarbeitung

Der zweite wesentliche Punkt nach der Datenaufnahme und -speicherung ist die Datenverarbeitung. Oft liegen die Messdaten in einer „rohen“ Form vor und müssen statistisch ausgewertet und archiviert werden. Da sich die Art und der Umfang der Auswertung stark nach dem Ziel der Messung richten und sehr verschieden sein können, eignen sich auch hierfür PCs recht gut, sofern sich der Rechenaufwand in kleinem bis mittlerem Umfang bewegt. Es gibt aber auch heute noch sehr viele Messdaten, die aufgrund ihrer Kompliziertheit und Menge nicht adäquat ausgewertet werden können, wie z.B. das `seti@home`-projekt zeigt. Hier wird von einem weltweiten Netz freiwilliger Teilnehmer Rechenzeit zur Verfügung gestellt, um ausserirdische Radiosignale auf die Existenz von Nachrichten hin zu überprüfen. Neben den eigentlichen, systembeschreibenden Daten fallen auch oft noch grosse Datenmengen anderer, benachbarter Systeme an, da sich diese in der Praxis nicht komplett trennen lassen. Bei Kenntnis des Verhaltens der störenden Systeme ist es möglich, die erwünschten von den unerwünschten Daten rechnerisch zu trennen.

## 2.3 Der prinzipielle Aufbau

Abbildung 1 zeigt den Informationsfluss vom Messaufnehmer zum Anwender sowie die jeweiligen Aktionen, die mit den Messdaten ausgeführt werden. Die vielfältige Verwendbarkeit des PCs beruht nicht zuletzt auf seiner freien Programmierbarkeit. Von den verfügbaren Programmen hängt also in grossem Maße die Nutzbarkeit des Rechners für den bestimmten Zweck ab.

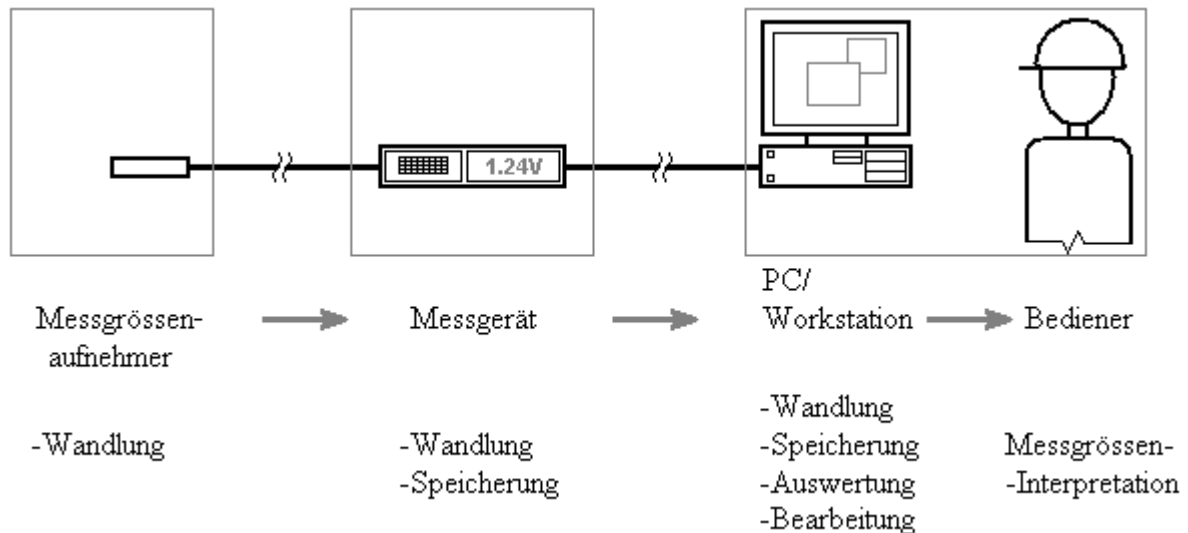


Abbildung 1: Messaufbau und Informationsfluss

## 2.4 Die Arbeitsweise im Rechner

Der PC verfügt im allgemeinen über ein sog. Betriebssystem, ein Programm, das die Verwaltung der im Rechner befindlichen Ressourcen und die Handhabung der bearbeiteten Daten übernimmt.

In Abbildung 2 wird der Informationsfluss im Innern des Messgerätes sowie des Rechners dargestellt. Natürlich kann dies in diesem Rahmen nur grob vereinfacht geschehen. Im allgemeinen bildet das Messgerät mitsamt seiner internen Firmware und seinen Elementen eine nach aussen hin geschlossene Einheit, die über eine Schnittstellenkarte die Daten liefert bzw. empfängt. Hier sind daher auch keine nennenswerten Konfiguration bzw. grössere Umbauarbeiten nötig. Da der PC freier konfigurierbar ist, fällt ihm die Aufgabe zu, sich auf die Gegebenheiten des Messgerätes in bezug auf die Hardware- und Softwareschnittstellen einzustellen. Praktisch heisst das, daß eine entsprechende Schnittstellenkarte in den PC eingesetzt werden muss. PC-Hardware wird nicht vom Anwendungsprogramm direkt angesprochen, sondern nur vom Betriebssystem. Dieses verfügt in Richtung der Hardware über eine standardisierte Softwareschnittstelle für den Hardwarezugriff: die Treiber. Diese sind nur in Richtung der Hardware spezifisch gestaltet und nur über sie kann das Betriebssystem auf die Hardware zugreifen. Treiber und Betriebssystem verfügen also über eine gemeinsame Softwareschnittstelle.

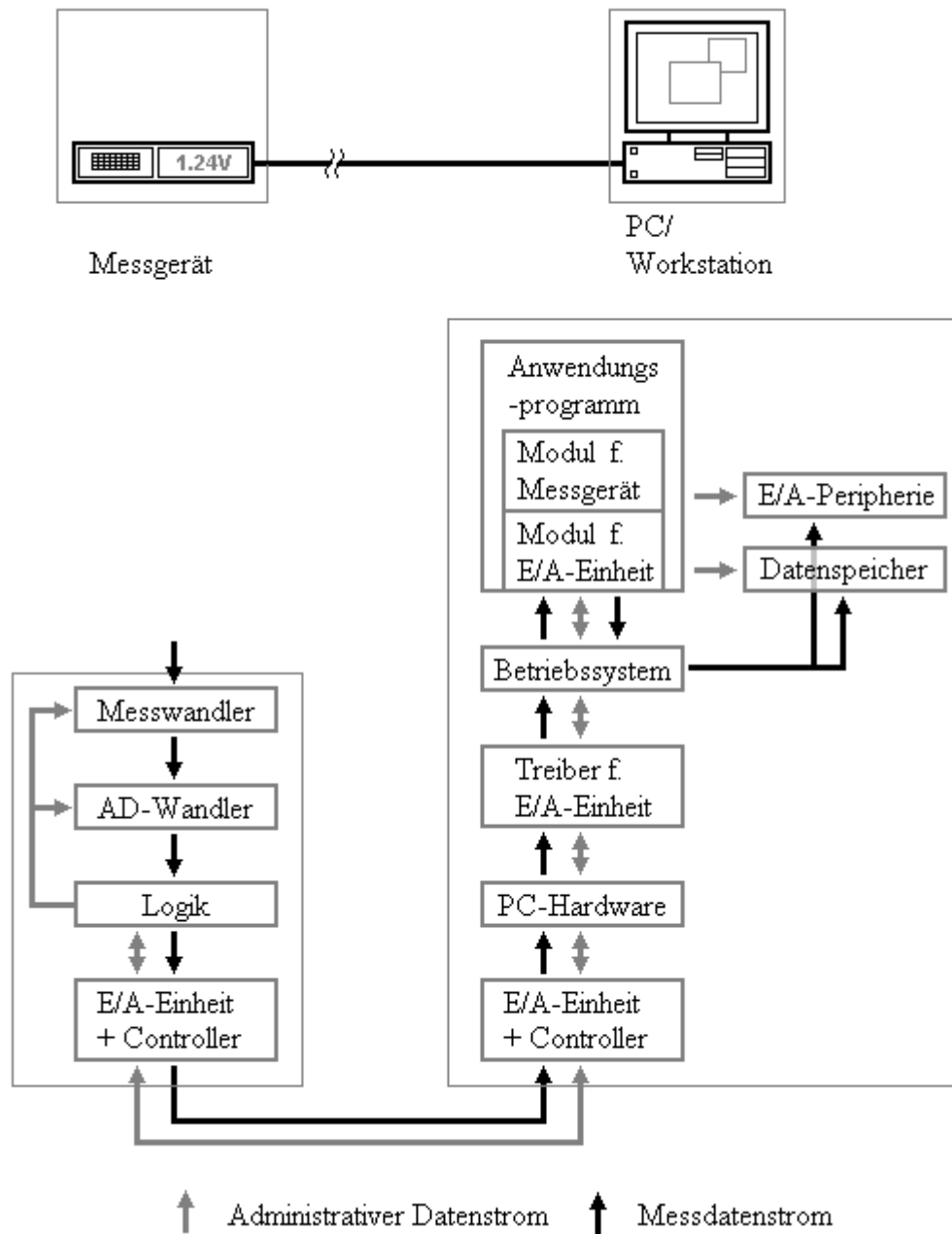


Abbildung 2: Interner Informationsfluss

Im Idealfall läuft also ein Hardwarezugriff eines Anwendungsprogramms wie folgt ab:

1. Anwendungsprogramm ruft Betriebssystemfunktion auf
2. Betriebssystem ruft Treiberfunktion auf
3. Treiber ruft Funktion der Hardware auf

Diese Vorgehensweise ist korrekt und lässt dem Betriebssystem die volle Kontrolle über die Hardware. Leider ist sie aber oft recht langsam, da

- die Daten, die eventuell zwischen Hardware und Anwendungsprogramm transferiert werden sollen, mehrfach von einer Schicht an die nächste übergeben werden müssen und
- die Betriebssystemfunktionen oft nicht das gesamte Spektrum der vorhandenen Treiberfunktionen abdecken.

Deshalb gibt es verschiedene Softwareschnittstellen, die es dem Anwendungsprogramm erlauben, Treiberfunktionen aufzurufen, wobei dem Betriebssystem die betreffende Funktion u.U. unbekannt ist. Es reicht den Aufruf einfach an den Treiber weiter. Funktionsbibliotheken, die diese Möglichkeit nutzen, sind OpenGL und DirectX. Leider geht damit die Hardwareunabhängigkeit des Anwendungsprogrammes z.T. verloren; bzw. sie wird gegen eine Abhängigkeit von einer Funktionsbibliothek eingetauscht. Auf diesen Zusammenhang begründen sich die Probleme, die vielfach auftreten und u.a. den gemeinsamen Betrieb von Software und Hardware verschiedener Hersteller unmöglich machen oder sehr erschweren. Da sich die Industrie in bezug auf gemeinsame, standardisierte Schnittstellen jeglicher Art nach wie vor erstaunlich zurückhaltend zeigt, findet man in vielen Bereichen der Technik solche „Inkompatibilitäten“. Die in Abbildung 2 gezeigten Module innerhalb des Anwendungsprogramms sind keine Treiber, obwohl sie eine ähnliche Funktion erfüllen, nämlich die Verständigung zwischen Hardware und Software zu ermöglichen.

## 2.5 GPIB

GPIB (General Purpose Interface Bus) besteht aus einem Standard-Kabel und -Protokoll für die Verbindung von Geräten mit einem Computer und wird auch HPIB, IEEE-488 und IEC-625 genannt. Er stellt eine schnelle Verbindung für bis zu 15 Geräte zur Verfügung, wobei der Bus, ebenso wie z.B. der USB, von einem Gerät zu nächsten verläuft. GPIB ist aber ein paralleler Bus. GPIB ist seit 1975 ein Industriestandard (IEEE-488). 1987 wurde vom IEEE Standard Komitee eine Anpassung mit der Bezeichnung IEEE-488.2 vorgenommen. Das Ansprechen der Geräte am Bus erfolgt prinzipiell über die Adresse des jeweiligen GPIB-Interface und die Adresse des betreffenden Gerätes am Bus und somit also über lediglich zwei Zahlen. Die GPIB-Schnittstellenkarte wird in den Rechner eingebaut und eine dem verwendeten Betriebssystem entsprechende Treibersoftware installiert. Diese bietet softwareseitig eine Anzahl von Funktionen an, mit denen sich Karte und Bus bedienen lassen. Da diese Funktionen zum größten Teil spezifisch für den GPIB-Bus beschaffen sind, kann das Betriebssystem-API<sup>1</sup> keine entsprechenden Funktionen beinhalten, die die Treiberfunktionen aufrufen und für beliebige Anwendungen bereitstellen. Die Anwendung, die den Bus benutzen möchte, muss demzufolge selbst auf die Treiberfunktionen zugreifen, was erstens nur dem privilegierten Nutzer erlaubt ist und ausserdem seitens der Anwendungssoftware ein treiberspezifisches Modul erfordert. Das Unvollständigsein/Fehlen ebendieses GPIB-Moduls bei TestPoint verursachte, wie im folgenden dargestellt, zunächst Probleme.

## 2.6 VISA

VISA (Virtual Instrument Software Architecture) ist eine Funktionsbibliothek von der Firma Agilent, die nach Vorgaben der VXIplug&play System Alliance entwickelt wurde, um den Betrieb von Software unterschiedlichster Hersteller auf einem System zu ermöglichen.

---

<sup>1</sup>API = Application Programming Interface

**VXI** VME Extensions for Instrumentation - beinhaltet GPIB-Befehle

**SCPI** Standard Commands for Programmable Instruments

Abbildung 3 zeigt eine grobe Übersicht über die verschiedenen Kommunikationsprotokolle für Messgeräte.

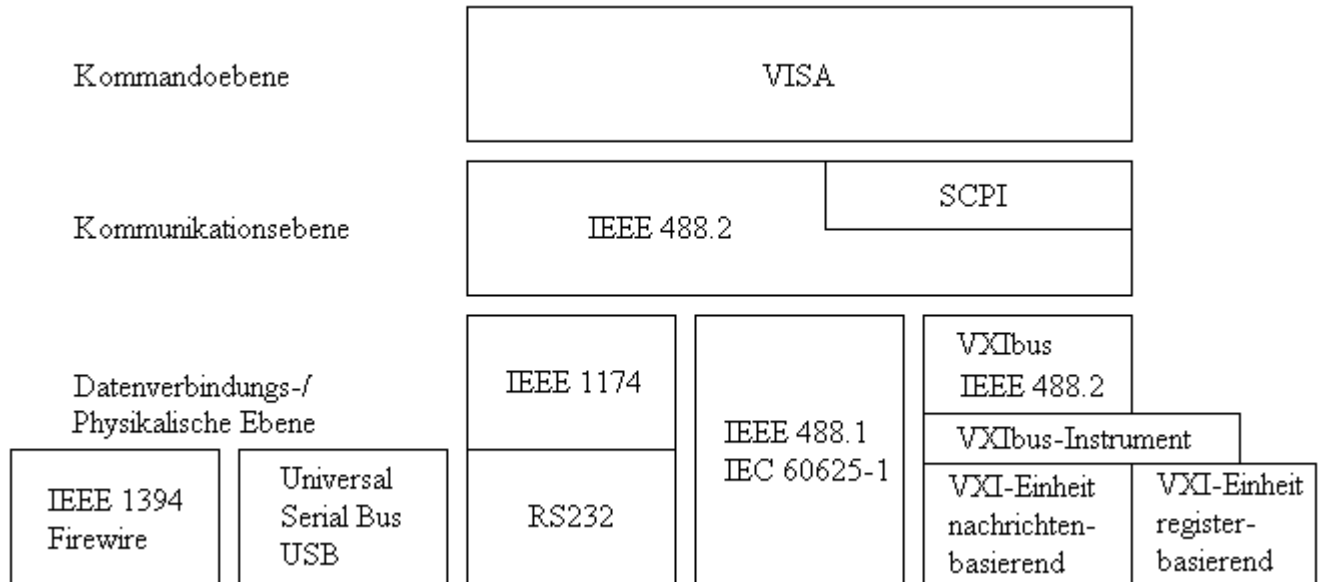


Abbildung 3: Protokollstack

### 3 Der Versuchsaufbau

Im vorliegenden Projekt standen zwei Programme zur Verfügung, die hinsichtlich ihrer Verwendbarkeit für die in den Kapiteln eins und zwei genannten Zwecke getestet und im Hinblick auf den nötigen Aufwand für die Realisierung eines Testaufbaues hin verglichen werden sollten.

- Vee Pro 6.01 von Agilent Technologies (ehem. Teil v. Hewlett-Packard) und
- TestPoint 3.3/4.1 von Capital Equipment Corporation (Keithley)

Hardwareseitig wurde zunächst folgender PC verwendet:

- PC Pentium 90
- 64 MB RAM
- IEEE 433.2-Schnittstellenkarte (GPIB), Agilent /
- IEEE 433.2-Schnittstellenkarte (GPIB), National Instruments
- Windows 98

später stand dann ein anderer Rechner zur Verfügung:

- PC Pentium III 1 GHz
- 256 MB RAM
- IEEE 433.2-Schnittstellenkarte (GPIB), National Instruments
- Windows 2000

Folgende Messgeräte wurden verwendet:

- Multimeter PREMA 6047, Prema GmbH
- Multimeter M-3610D/M-3650D, Voltcraft

## 4 Vee Pro

Das Programm, das zum Einsatz kam, war Vee Pro Version 6.01 vom 15.08.2000. Dieses Programm stammt offensichtlich aus der UNIX-Welt und wurde auf Windows-32 portiert. Die Portierung ist sehr gut gelungen, das Programm ist vor allem sehr stabil und relativ ressourcenfreundlich, wie der erfolgreiche Einsatz auf dem ersten Rechner zeigt. Erfreulicherweise funktioniert sogar eine Zwischenablage zwischen mehreren Instanzen von HPVee, so dass zwischen verschiedenen, geöffneten Dokumenten Elemente ausgetauscht werden können.

### 4.1 Die Installation

Die Installation verlief, wie unter Windows gewohnt, komfortabel und unkompliziert. Für die Entwicklungsumgebung werden ca. 160 MB Festplattenplatz benötigt.

### 4.2 Hardware-Zugriff

Der Zugriff auf beide GPIB-Schnittstellenkarten funktionierte ohne Probleme. Die Geräteerkennung von Vee erkannte zunächst die Interface-Karte und dann alle angeschlossenen Geräte.

### 4.3 Die Vee-IDE

Das Programm Vee bietet eine IDE <sup>2</sup> innerhalb deren sich ein oder mehrere Fenster öffnen lassen. Diese können das Programm selbst oder ein sogenanntes „Panel“ enthalten, von dem später noch die Rede sein wird. Die Vee-IDE bietet einen grafischen Programmeditor, einen Compiler, Interpreter und Debugger. Das Vee-Programm, eine Datei mit dem Suffix „.vee“ wird vom Compiler übersetzt und es entsteht eine Binärdatei, die vom Interpreter ausgeführt wird. Die Binärdatei kann auch als eigenständige Datei mit dem Suffix .vxe abgespeichert werden. Wenn auf dem jeweiligen Rechner lediglich Vee-Programme ausgeführt und nicht entwickelt werden sollen, reicht es aus, statt der IDE eine Laufzeitumgebung (runtime environment) zu installieren. Diese interpretiert die „.vxe.“-Binärdatei.

Das Vee-Programm wird zunächst innerhalb eines oder mehrerer Programmfenster in der IDE erstellt. Beim Start zeigen die im Programm vorhandenen Ausgabelemente die Daten sofort an, so dass mit dem Programm bereits gearbeitet werden kann. Oft genügt das Programm aber nicht den Ansprüchen in bezug auf die Ergonomie, da es viele Elemente beinhaltet, die von den interessanten Daten oder Bedienelementen ablenken. Deshalb können ausgewählte Programmelemente, in ein sogenanntes **Panel** überführt werden, ein Extrafenster, dass dann als GUI <sup>3</sup> des Vee-Programms fungiert. Prinzipiell lassen sich alle Programmelemente in ein Panel bringen, doch hat dies sicher nur bei visuell relevanten Elementen einen Sinn. Wenn das Programm dann gestartet wird, werden die Ausgaben sowohl innerhalb des Erstellungsfensters ausgegeben, als auch in den Elementen im Panel. Die Lage, Grösse, Farbe sowie weitere Attribute der Programmelemente können sowohl im Panel als auch im Erstellungsfenster geändert werden und die Bedienung erfolgt weitgehend mit der Maus.

---

<sup>2</sup>Integrated Development Environment = Integrierte Entwicklungsumgebung

<sup>3</sup>Graphical User Interface = Grafische Benutzerschnittstelle

## 4.4 Die Sprache Vee

Konzeptionell ist Vee eine grafische Programmiersprache. Alle Sprachelemente und -konstrukte werden dabei grafisch dargestellt und haben prinzipiell den gleichen Aufbau. Sie sind viereckig, haben einen Namen und mehrere Datenein- und -ausgänge. Die Datenausgänge sind mit den Dateneingängen der anderen Elemente über Linien verbunden, die den Datenfluss repräsentieren. Die Datenverbindungen werden in Abhängigkeit von ihrem jeweiligen Zweck im Programm in administrative Verbindungen und reine Datenverbindungen unterteilt. Die administrativen Verbindungen steuern den Programmablauf, d.h. sie starten und stoppen Programmelemente. In Vee erkennt man den Unterschied sowohl an der Farbe der Linien, (administrative Datenstränge sind z.B. grau) als auch daran, wo die Linien starten und enden. Die Programmelemente in Vee haben links meist Eingänge für Nutzdaten und rechts deren Ausgänge. Oben ist ein Eingang für administrative Daten, unten der entsprechende Ausgang. Die Nutzdaten werden noch einmal hinsichtlich ihres Datenformates unterschieden. So werden beispielsweise Daten aus Feldelementen andersfarbig dargestellt, als normale Integer-Daten oder Zeichenketten. In Abbildung 4 ist ein einzelnes Element mit je einem administrativen (A) Eingang und Ausgang sowie je einem Nutzdaten- (B) -ein- und -ausgang dargestellt. In herkömmlichen

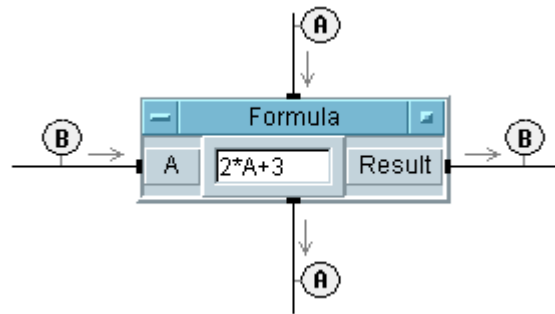


Abbildung 4: Vee-Programm-Element

Programmiersprachen, wie z.B. C oder Pascal, wird beim Aufruf von Funktionen sehr grosser Wert auf den Typ von Daten gelegt. Es ist z.B. elementar für den Erfolg der Datenverarbeitung, ob einer Funktion eine Variable vom Typ einer Zeichenkette (**string**) übergeben wird oder ein Gleitkommatyp (**real**). Moderne Compiler überprüfen deshalb den Typ der übergebenen Variablen sehr genau, wobei ein falscher Datentyp ein unbedingtes Abbruchkriterium darstellt. In Vee stellt das Einbinden eines Programmelementes einen Funktionsaufruf dar. Die Datenstränge, die das Element mit anderen Elementen verbinden, bestimmen die Art der übergebenen Variablen. Leider prüft Vee beim Compilieren nicht immer deren Typ und die an sich sehr reichhaltige Online-Hilfe gibt keinen rechten Aufschluss darüber, welchen Typ manche Funktionen erwarten. Deshalb endet der Programmlauf in unklaren Fällen mitunter in einem Laufzeitfehler. Der integrierte Debugger gibt dann jedoch sehr schnell Hinweise darauf, wo es zu dem Fehler gekommen ist.

Vee unterstützt prinzipielle Programmelemente, wie:

- Funktionsaufrufe implizit (als Programmelemente)
- Funktionsaufrufe explizit (als separat gelagerten und beliebig aufrufbaren Code)
- (Rekursion)
- globale und lokale Variablen
- Felder
- Kontrollstrukturen

sowie noch andere, bietet damit den Sprachumfang einer High-Level-Sprache.

Weiterhin steht eine grosse Anzahl Funktionen zur Verwendung in eigenen Programmen zur Verfügung, die u.a. folgende Themen abdecken:

Array	Matrix	Time-Date	Complex	Data-Filtering
Bessel	Panel	Real-Teile	String	SystemInfo
Bit	Potenz	Signal-Prozessing	Statistik	Matlab

#### 4.4.1 Multitasking

Multitasking ist eine Möglichkeit, Programme auf einem Rechner mit nur einem Prozessor quasiparallel auszuführen. Zunächst möchte ich die drei Begriffe „Task“, „Prozess“ und „Thread“ kurz erklären.

**Task** Ein Task ist eine Aufgabe im weitesten Sinne, die der Rechner zu erledigen hat, also ein Programm. Wenn auf dem Rechner zwei Anwendungsprogramme laufen, dann sind zumindest zwei Tasks aktiv.

**Prozess** Als Prozess bezeichnet man die gesamte Softwareumgebung eines Tasks: sein ihm zugewiesener Speicher, Umgebungsvariablen Code- Daten- und Stacksegmente etc. Wenn ein Anwendungsprogramm gestartet wird, wird zunächst vom Betriebssystem ein Prozess erzeugt innerhalb dessen die Threads des Anwendungsprogrammes laufen.

**Thread** Der Thread ist der Teil eines Tasks, der auf der CPU ausgeführt wird, der Programmcode.

Ein Task besteht also aus zumindest einem Prozess und einem initialen Thread. Vee unterstützt prinzipiell das vom Betriebssystem angebotene Multithreading. Beim Betrachten des Programmes lässt sich meist leicht erkennen, wo Vee den Datenfluss in separate Threads aufteilt.

#### 4.4.2 Rekursion und Rückkopplung

Bei Rekursionen sind stets der Ein- und Austrittspunkt der Rekursion kritische Punkte, denn wenn der Eintrittspunkt falsch oder nicht definiert wurde, beginnt der Prozess in einem unklaren Zustand, beim fehlerhaften Festlegen des Austrittspunktes kann sich der Prozess ins unendliche ziehen. In Vee sind Rekursionen zwar möglich, aber sie benötigen offenbar ein Schleifenelement, um kompiliert zu werden. Da die Schleife eigentlich gerade durch die Rückkopplung entsteht, sollte eine weitere Schleife an sich unnötig sein.

In den folgenden beiden Funktionen (Pascal) wird dies noch einmal verdeutlicht und gleichzeitig dargestellt, dass eine rekursive Lösung in bezug auf die Grösse des benötigten Codes nicht immer kürzer ist. Wie man sieht, kommt die rekursive Lösung aber ohne Schleife aus, da die Rückkopplung eine Schleife impliziert.

```
function rfak(n : integer) : extended;
begin
  if n < 1 then result:= 1
  else result:= n * rfak(n - 1);
end;

function nfak(n : integer) : extended;
var i : integer;
begin
  result:= 1;
  for i:= 1 to n do result:= result * i;
end;
```

In Abbildung 11 ist eine Rekursion, wie sie in Vee implementiert wird, dargestellt. Die Schleife dient dabei nur der Initiierung der Rekursion - die Schleifenvariable wird nicht verwendet, statt dessen die Variable *ii*, die bei jedem Durchlauf um eins inkrementiert und ausgegeben wird.

#### 4.4.3 Programmbeispiele

Um das Konzept der Programmierung mit Vee vorzustellen, werden hier einige Beispiele gezeigt, die deren Prinzip verdeutlichen sollen. Es werden Kontrollstrukturen verwendet, die äquivalent zu anderen Programmiersprachen sind. Es wird der entsprechende Code in C bzw. Pascal gezeigt. Da Vee äusserlich eine grosse Ähnlichkeit mit einem Programmablaufplan (PAP) hat stellt Abbildung 5 den PAP einer If-Then-Konstruktion dar.

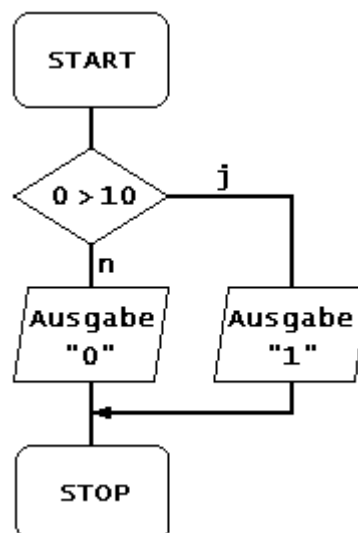


Abbildung 5: Programmablaufplan

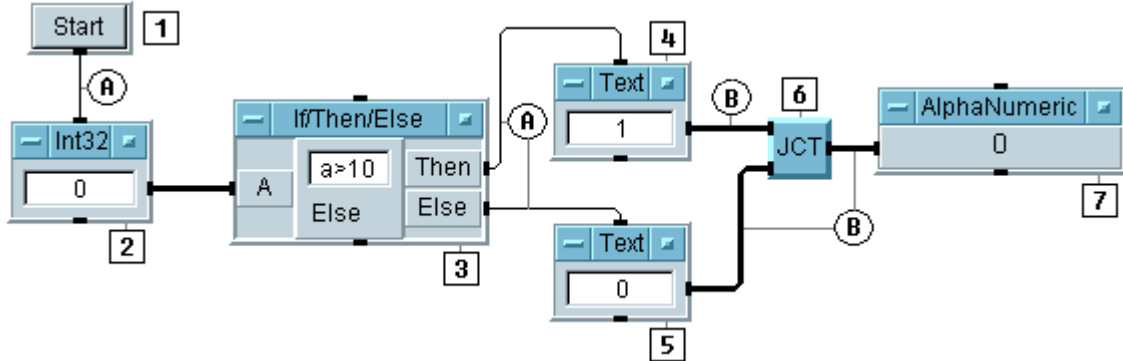


Abbildung 6: if-then-else-statement

Hier nun derselbe Ausdruck in C und in Pascal:

```
if (0 > 10) printf("1"); else printf("0");
if 0 > 10 then write('1') else write('0');
```

Ich möchte das in Abbildung 6 dargestellte, vollständige Vee-Programm erläutern. Die Nummerierung wurde nachträglich angebracht. 1 bis 7 sind Programmelemente. Der Nutzdatenfluss (B) ist in Abbildung 6 etwas dicker dargestellt. [A] ist der administrative Datenfluss. Wie zu erkennen ist, ist nicht immer ein expliziter administrativer Strang notwendig, manchmal ist auch durch die entsprechenden Datenverbindungen die Programmstruktur klar. Wenn also im Laufe des Programmes ein Element an ein anderes Nutzdaten ausgibt, so ist deren Eintreffen für den Empfänger ebenfalls ein administratives Signal, diese Daten zu verarbeiten. Manche Elemente benötigen aber zwingend einen administrativen Datenanschluss, wie z.B. das Element „gate“, das einen Schalter für Datenverbindungen darstellt.

„Start“[1]	Programmstart. Dieses Element wird zuerst ausgeführt.
„Int32“[2]	32-Bit-Integer Konstante mit dem Wert 0
„If/Then/Else“[3]	if-then-else-Verzweigung, in der der eingehende Wert mit 10 verglichen wird
„Text“[4]/[5]	Textkonstanten mit den Werten „1“ und „0“
„JCT“[6]	sog. <i>junction</i> , der in Vee immer dann benötigt wird, wenn mehr als eine Datenquelle an eine Senke angeschlossen werden soll.
„AlphaNumeric“[7]	eine Alphanumerische Ausgabe.

Nach dem Programmstart wird `Int32(0)` mit 10 verglichen und wenn `0 > 10` „1“ ausgegeben, ansonsten „0“. Um die wenig Übersichtliche Darstellung auf das für den Anwender wesentliche zu beschränken, können „AlphaNumeric“ und „Int32“ in ein Panel überführt werden. (Abbildung 7).

In Abbildung 8 ist der Aufbau einer `for-to`-Schleife dargestellt. Offensichtlich ist der Aufbau hier sehr einfach und übersichtlich. Das Element „Start“ [1] stellt den Startpunkt des Threads dar. Der administrative Datenfluss führt weiter zu dem Element `For-Count` [2], das die eigentliche Schleife darstellt. Die Zählvariable wird während des Schleifendurchlaufes ausgegeben und an das Ausgabeelement „Logging AlphaNumeric“ [3] weitergegeben. Das Element „Logging AlphaNumeric“ ist ein listenförmiges Ausgabeelement, das sowohl Zeichenketten, als auch numerische Werte als Eingabeparameter zulässt.

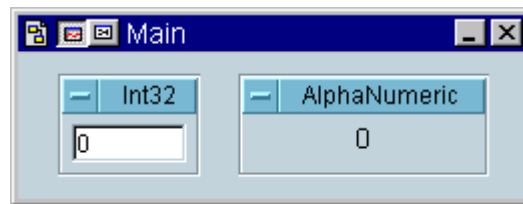


Abbildung 7: if-then-else-statement - Panel

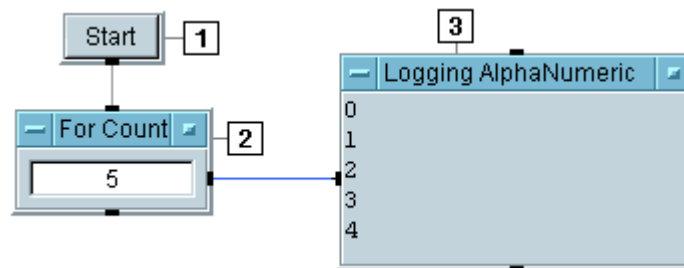


Abbildung 8: for-to-statement

Die Darstellung in C und Pascal ist ebenfalls kurz:

```
for (i=0; i<5; i++) printf("%d\n", i);
```

```
for i:= 0 to 4 do writeln(i);
```

Um eine numerisch begrenzte Schleife ohne das `for-to`-Statement zu erzeugen, bietet sich eine `while-do`-Schleife an. Die Implementation in C und Pascal ist nicht wesentlich komplizierter als vorher:

```
i = 0;
while (i < 5) printf("%d\n", i++);
```

```
i:= 0;
while i < 5 do begin
  writeln(i);
  inc(i);
end;
```

Abbildung 9 zeigt den Aufbau einer `while-do`-Schleife. Wie deutlich zu sehen ist, hat die Komplexiertheit gegenüber der `for-to`-Schleife deutlich zugenommen. Dies liegt vor allem daran, dass für den lesenden und schreibenden Zugriff auf Variable jeweils ein grafisches Programmelement nötig ist. Bereits bei diesem sehr einfachen Beispiel zeichnet sich eins der Hauptprobleme bei diesem grafischen Programmierkonzept ab, nämlich, dass bereits ein mittleres Programm, sehr schnell an Übersichtlichkeit verliert. Um diesem Fakt etwas entgegenzusteuern bieten viele graphische Programmelemente die Möglichkeit, sie zu verkleinern. In Abbildung 10 derselbe Aufbau noch einmal verkleinert. Ausgehend vom „Start“-Element wird zunächst die globale Laufvariable `ii` mit dem Wert der `Int32` Konstante (0) initialisiert.

Gleichzeitig wird das „Until Break“-Element gestartet, welches daraufhin immer wieder das „Get ii“-Element aufruft. Das „Get ii“-Element liest den Wert der Laufvariablen *ii* und gibt ihn dann an drei Elemente weiter: die Ausgabeeinheit „Logging AlphaNumeric“, die Formel „Formula“, die den Wert um 1 erhöht, und die „If/Then/Else“-Einheit, die den aktuellen Wert von *ii* mit dem Endwert (5) vergleicht und gegebenenfalls an das „Stop“-Element weitergibt. Solange also der Ausdruck  $ii < 5$  erfüllt ist, wird *ii* erhöht, ansonsten wird beendet. In Anbetracht der sich schnell verlierenden Übersicht ist es wichtig, möglichst die einfachsten Konstrukte für die Lösung eines Problems zu finden.

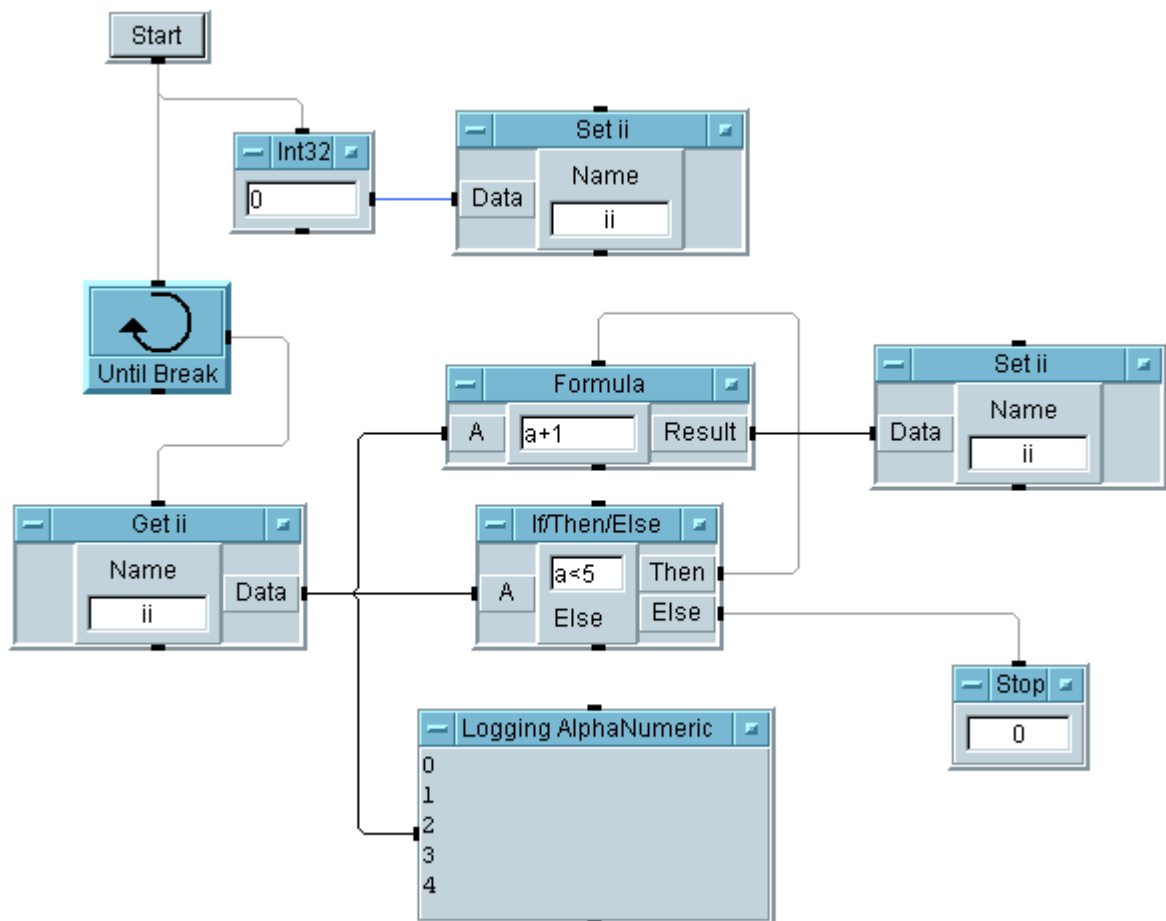


Abbildung 9: while-do-statement

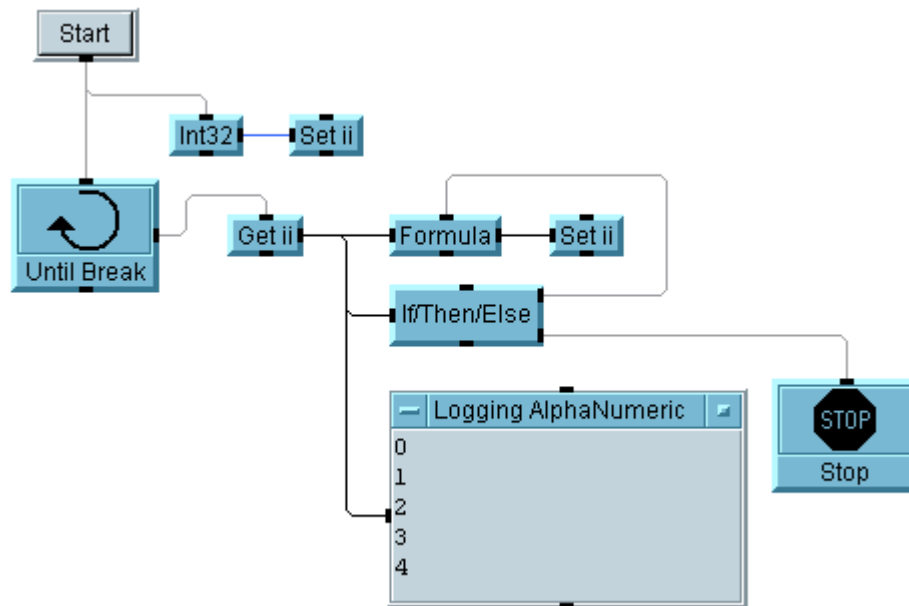


Abbildung 10: while-do-statement - minimiert

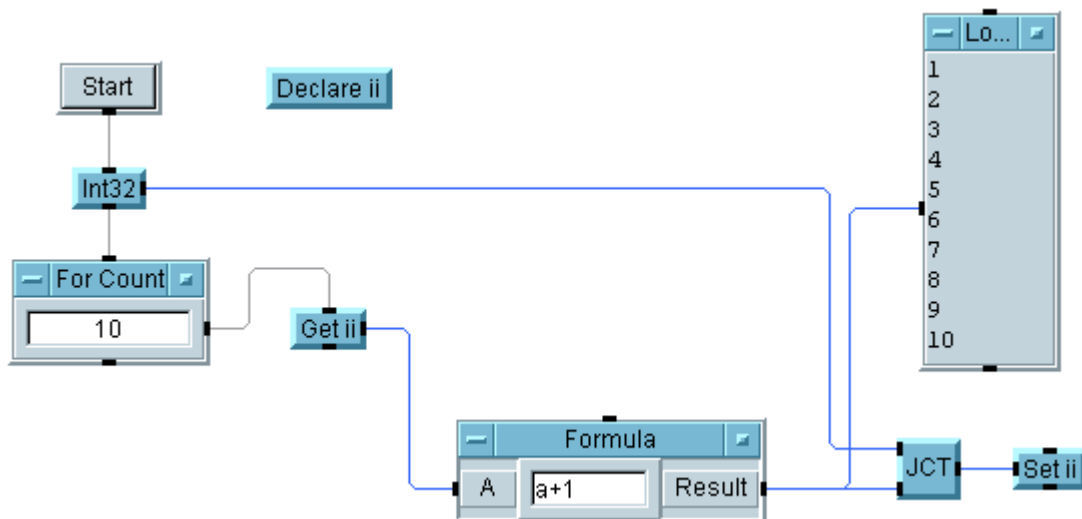


Abbildung 11: Rekursion

Wie bereits an den Beispielen in den Abbildungen 6, 9 und 9 ersichtlich ist, lassen sich Daten an eine Datensenke ausgeben. Die Visualisierung geschieht über die Datenausgabe an ein entsprechendes Ausgabeelement (Logging Alphanumeric). Ebenso können Daten auch an ein Messgerät übergeben werden.

Für diesen Zweck verfügt Vee über den sogenannten **Instrument Manager**. Wenn die Hardware und Software korrekt installiert sind, listet Vee im **Instrument Manager** alle verfügbaren bzw. angeschlossenen externen Messgeräte auf. Es werden mehrere Methoden angeboten, mit dem Peripheriegerät zu kommunizieren. Als naheliegendste Variante bietet sich das „**Direct I/O**“-Modul an.

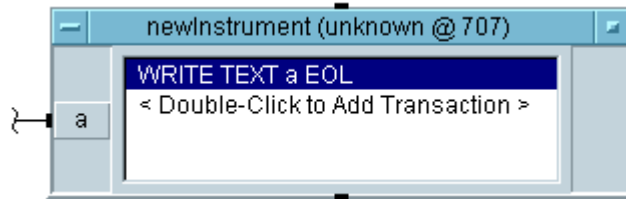


Abbildung 12: Direct I/O Modul

Das „**Direct I/O**“-Modul überträgt an das und von dem Peripheriegerät ASCII-Zeichenfolgen. Diese können je nach verwendetem Gerät Daten enthalten aber auch Instruktionen an das Gerät bzw. Statusinformationen vom Gerät. Für den Programmierer sind natürlich in diesem Zusammenhang detaillierte Information über die vom jeweiligen Gerät erwarteten Daten unumgänglich. Das Programm, das entsteht, beinhaltet das in Abbildung 2 erwähnte „Modul f. Messgerät“, das in das Anwendungsprogramm integriert ist.

## 4.5 Das Testprogramm

Das Vee-Programm, das entstand, um die Möglichkeiten der Sprache zu untersuchen, und auch um ein praktisches Anwendungsbeispiel zu haben, sollte folgende Eigenschaften haben:

- wesentliche Bedienfunktionen des Messgerätes übernehmen und visualisieren,
- Daten aus dem Messgerät übernehmen und in verschiedener Form darstellen
- Daten aus dem Messgerät in Dateien abspeichern

Für die Entnahme der Messwerte aus dem Messgerät existieren folgende Modi:

**Dauerbetrieb** die Messwerte werden permanent vom Messwert angefordert und übertragen. Die Anzahl der Messwerte pro Zeiteinheit richtet sich nur nach den konstruktiven Gegebenheiten des Gerätes sowie der Integrationszeit

$$n = \frac{T_m}{T_i}$$

$n$  Anzahl der Messwerte,  $T_m$  Messdauer,  $T_i$  Integrationszeit

**Intervallbetrieb** die Messwerte werden jeweils nach einem einstellbaren Zeitintervall angefordert.

**Startbetrieb** die Datenaufnahme beginnt sofort mit der Aktivierung des Programms und läuft solange, bis das Programm deaktiviert wird.

**Zeitbetrieb** die Datenaufnahme erfolgt innerhalb zweier einstellbarer Zeitpunkte. So können auch bedienerunabhängig Messungen vorgenommen werden.

Die Daten werden in ein wählbares Verzeichnis abgelegt, wobei für jede Messreihe jeweils eine Headerdatei angelegt wird, die Informationen zur Messreihe an sich beinhaltet, sowie eine Datendatei. Die Art der zu messenden Werte lässt sich zwischen Spannung, Strom und Widerstand wählen. Die Art der Spannung bzw. des Stroms (Gleich-/Wechsel-) ist wählbar. Bei Widerstandsmessungen steht die Wahl zwischen Zwei- und Vierleitermethode. Für alle drei Größen lassen sich die vom Messgerät vorgegebenen Messbereiche und die Integrationszeit auswählen. Ausserdem werden die Daten grafisch in einem Diagramm angezeigt.

Die Abbildungen 24, 25 und 26 zeigen Programmablaufpläne der wichtigsten Komponenten des Testprogramms.

Im folgenden sind die Inhalte von je einer Headedatei und einer Datendatei abgebildet.

```
PREMA 6047
Integrationszeit: 400 ms
Messbereich: 0.2 mA
Messgroesse: I~
Startzeit:30.10.2002 14:23:05
Stopzeit:30.10.2002 14:23:33
Startbetrieb
Intervallbetrieb

1 Wed 30/Oct/2002 14:23:05 000001.0086E-2
2 Wed 30/Oct/2002 14:23:05 000001.0090E-2
3 Wed 30/Oct/2002 14:23:05 000001.0073E-2
4 Wed 30/Oct/2002 14:23:06 000001.0073E-2
```

## 5 TestPoint

### 5.1 Die Installation

Die Installation des Programmes TestPoint ging windowsgemäss einfach vonstatten und bot somit keine Probleme. Als Kopierschutz dient ein Dongle, der in die erste (sofern mehrere vorhanden sind) der parallelen Schnittstellen (LPT1) gesteckt wird und auf andere Geräte, die „durchgeschleift“ werden, erfreulich neutral reagiert.

In einer Konfiguration **Rechner-Dongle-Scanner-Drucker** liefen alle Geräte problemlos.

Testpoint beansprucht im Vergleich zu Vee wesentlich weniger Platz. Der Installationsordner belegt nur etwa 10MB. Leider gibt sich das Programm in bezug auf die CPU-Last weit weniger genügsam, denn es erzeugt auch bei völliger Bewegungslosigkeit 100% Last, lässt sich aber gern von anderen Anwendungen einschränken, so dass nach wie vor mit anderen Programmen gearbeitet werden kann. Möglicherweise horchen einige Module im Pollingbetrieb an Hardware-reports.

Ein Problem bereitete das Ansprechen der GPIB-Karte, so dass nach Rücksprache mit der Hotline von der Firma Keithley die aktuelle Version 4.1 von Test Point installiert wurde. Bekanntermassen ergeben sich immer wieder Probleme, wenn Programm und Karte nicht aus dem selben Hause stammen. Da aber erfahrungsgemäss National-Instruments-Karten mit den meisten Programmen zusammenarbeiten, wurde die Agilent-Karte gegen eine solche ersetzt.

### 5.2 Hardware-Zugriff

Die Korrespondenz mit den Herstellern von Software (Keithley) und Hardware (National Instruments) ergab, dass TestPoint im Standard-Lieferzustand nicht mit der GPIB-Software-Schnittstelle kommunizieren kann. Ein entsprechendes Treibermodul ist bei Keithley für einen nicht unbeträchtlichen Betrag zu erwerben. Die NI-Karte bietet allerdings eine als weitere, neuere Softwareschnittstelle, VISA. Da die TestPoint Version 3.3 kein VISA-Modul beinhaltet, wurde die Version 4.1 installiert, die sich kostenfrei von der Netzpräsenz des Herstellers laden lässt. Sie funktioniert auch ohne Dongle, dann aber ohne Hardwarezugriffe. Mit dem enthaltenen VISA-Modul war die Kommunikation mit dem Messgerät nach Angabe der entsprechenden GPIB-Adresse des Messgerätes (“GPIB0::7::INSTR“) problemlos möglich.

### 5.3 Die Testpoint-IDE

Auch TestPoint verfügt, wie Vee, über eine IDE, einen Integrierten Debugger und einen Interpreter. Die IDE erzeugt Dateien mit dem Suffix `.tst`.

### 5.4 Die Sprache TestPoint

Das Konzept von Testpoint unterscheidet sich aber in zweierlei Hinsicht von Vee:

1. Testpoint ist keine grafische Programmiersprache. Die Programmierung erfolgt zwar zu einem grossen Teil grafisch, per Drag’n Drop, aber das entstehende Programm repräsentiert sich weitgehend textlich.
2. Testpoint arbeitet nicht, wie Vee, mit dem Datenstrom-Konzept, sondern das Programm baut sich aus einem System von Aktionen auf, die zu den Programmelementen gehören und andere Elemente benutzen und beinhalten.

In TestPoint werden die Programmelemente als Objekte bezeichnet und auch weitgehend aus objektorientierter Sicht betrachtet. Das Konzept ist aktionsorientiert und der Aufbau des Programmes spiegelt die möglichen Aktionen der einzelnen Objekte wieder. Der Datenfluss wird, wie auch bei anderen Programmiersprachen, durch Kontrollstrukturen u.a. gesteuert und ist nicht explizit sichtbar. In TestPoint gibt es für jedes Programm zumindest ein Panel, auf dem Bedien- und Visualisierungselemente abgelegt werden können. Zur Laufzeit werden Resultate und Prozessdaten nur im Panel angezeigt.

Alle TestPoint-Objekte besitzen ein sogenanntes Objekt-Detail-Fenster, das im allgemeinen aus vier Teilen besteht:

„**Settings**“ enthält einige veränderbare Eigenschaften des Objektes

„**Actions**“ enthält eine Liste mit Aktionen, die beim Aufruf oder Aktivieren des Objektes ausgeführt werden. Diese Listen stellen den weitaus grössten Teil des TestPoint-Programmes dar.

„**Comments**“ Platz für Kommentare

„**XRef**“ eine Liste mit all jenen anderen Objekten, die das betrachtete Objekt referenzieren.

Weiterhin enthält die IDE ein Menu-Fenster mit durch Icons dargestellten Programmelementen (Objekten) das in TestPoint „Stock“ genannt wird und ein Objekts-Fenster pro Programm. Das Objects-Fenster, dass auch in Abbildung 19 zu sehen ist, dient als Träger der im Programm verwendeten Objekte.

Die Programmierung erfolgt nun so, dass Objekte mit der Maus aus dem Stock in das Objekts-Fenster gezogen und abgelegt werden. Jedes Objekt erhält daraufhin ein Objekt-Detail-Fenster mit einer leeren Action-List. Objekte können nun aus dem Objekts-Fenster in die Action-List anderer Objekte gezogen werden, wobei der zugehörige Quellcode in der Action-List von der IDE automatisch erstellt wird. Manche Einträge in diesem Quellcode müssen dennoch von Hand editiert werden, z.b. Variablenwerte, Grenzen von Laufvariablen, Zeichenketten etc. Leider ist es nicht möglich, den gesamten Quelltext manuell einzugeben. Die IDE lässt dies zwar zu, so dass der Quelltext so aussieht, wie der automatisch korrekt erstellte, das Programm arbeitet aber nicht korrekt.

Ein Beispiel:

Eine Schleife soll von 1 bis zu einem Wert laufen, der im Dateneingabeobjekt „**Data-Entry1**“ hinterlegt ist. Das Schleifenobjekt „**Loop1**“ wird aus dem Objekts-Fenster in eine Action-List gezogen und der Rumpf des Schleifenquelltextes automatisch erzeugt. Korrekterweise muss nun auch das Objekt „**Data-Entry1**“ aus dem Objekts-Fenster in den Schleifenquellcode gezogen werden. Es ist möglich, manuell in den Quelltext den Eintrag „**Data-Entry1**“ zu machen, aber TestPoint erkennt nicht, dass das Objekt gemeint ist und interpretiert „**Data-Entry1**“ als Zeichenkette. Das Programm wird nun zwar kompiliert und ist auch lauffähig, aber die Schleife wird nicht durchlaufen, weil als obere Schleifengrenze ein unsinniger Wert eingegeben wurde.

Dies birgt für den Anfänger gewisse Schwierigkeiten, zumal die meisten Anwender aus ihrer Erfahrung mit anderen Programmiersprachen heraus dazu neigen werden, Objektnamen manuell einzugeben. Prinzipiell ist aber die Syntax nach kurzer Eingewöhnung recht einfach und es lassen sich sehr schnell Programme erstellen. Da der Code auf die Aktions-Listen der verschiedenen Objekte verteilt ist, ergibt sich automatisch ein modularer Aufbau des Programmes. Als angenehm fällt auch auf, dass der Code sehr kompakt und übersichtlich ist.

### 5.4.1 Variablen

Variablenwerte lassen sich sowohl einzeln in den verschiedenen Objekten ablegen, als auch explizit in einem separaten Variablenobjekt, dem „Container“. Dieses Objekt ist ungemein leistungsfähig. In eine Struktur eingebettet (Schleifen etc.) können beliebige Datenwerte (Zeichenketten, Zahlen etc.) in das Containerobjekt geschrieben oder angefügt werden. Das Objekt steht danach für die Datenausgabe, -visualisierung oder -speicherung als kompakte und gut handhabbare Struktur zur Verfügung. Einem Datenausgabeobjekt (Diagramm), in TestPoint „Graph“, muss beispielweise nur noch das Containerobjekt übergeben werden. Die auf der Abszisse abgetragenen Werte werden implizit aus dem Index der im Container enthaltenen Werte angenommen. (Darstellung: Containerwert[Index] über Index). Äquivalent dazu kann das Containerobjekt an eine Numerische Datenausgabe übergeben werden, die die Werte verbal darstellt. Es ist auch möglich, innerhalb eines Diagrammes die Daten mehrerer Container darzustellen. Dem Containerobjekt kann als Inhalt auch der Inhalt eines oder mehrerer anderer Containerobjekte zugewiesen werden. Abbildung 13 zeigt die Vorgehensweise.

### 5.4.2 Multitasking

Prinzipiell unterstützt TestPoint das Multithreading von vornherein, da jede Aktions-Liste einen Task repräsentiert. Manchmal ist aber der „gleichzeitige“ Ablauf unerwünscht, z.B. wenn zwei Threads versuchen, gleichzeitig schreibend auf eine Ressource zuzugreifen. Zu diesem Zweck lässt sich das Multithreading für jedes Panel auch abschalten oder das Objekt Task verwenden, welches erweiterte Einstellmöglichkeiten und vor allem auch dessen Verhinderung, z.B. durch Einsatz von „critical sections“ erlaubt. „critical sections“ sind Codeabschnitte, die nicht von mehr als einem Task gleichzeitig ausgeführt werden dürfen. Dadurch wird exklusiv auszuführender Code auch in dieser Form ausgeführt.

1) Set	Math1	to 1
2) Linear series	Loop1	from 1 to 1000 , step by 1
3) Calculate	Math1	with x=Loop1
4) Append to	Container1	from Math1 ,
5) End	Loop1	
6) Draw graph	Graph1	with Container1 ,
7) Store in	Container2	from Container1 , Container1 ,
8) Set	Grid1	to Container2

Abbildung 13: das Container-Objekt

### 5.4.3 Programmbeispiele

Ich möchte auch hier mit einigen simplen Beispielen das Programmierkonzept verdeutlichen. In Abbildung 14 ist das Panel eines einfachen TestPoint-Programmes zu sehen. In die Eingaben „Data-Entry1/2“ gibt der geneigte Benutzer je eine Zahl ein, die dann vom Programm auf Gleichheit überprüft werden. Gegebenenfalls wird dann von „Display1“ „equal“ oder „not equal“ ausgegeben. Mit Click auf „Pushbutton1“ wird der Vergleich angestoßen.

Abbildung 15 zeigt die zu „Pushbutton“ gehörende Action-List des vorliegenden Programms. Das Objekt, welches den Vergleich vornimmt, das Conditional-Objekt TEST. In TEST ist der Ausdruck  $x = y$  hinterlegt, wobei zu Laufzeit an  $x$  der Wert von „Data-Entry1“ und an  $y$  der von „Data-Entry2“ übergeben werden. Es wird also, um eine Parallele zu anderen Sprachen zu ziehen, eine Methode des Conditional-Objektes mit zwei Parametern aufgerufen. Dieses

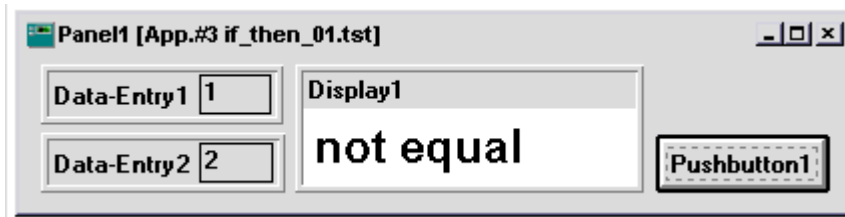


Abbildung 14: Ein TestPoint-Programm - Panel

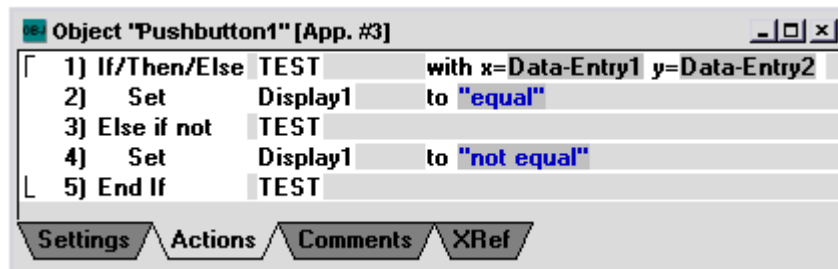


Abbildung 15: Action-List mit if-then-else-Statement

Beispiel ist äquivalent zu dem in Abbildung 6 dargestellten Vee-Programm. Es soll erwähnt werden, dass alle Einträge in der Action-List bis auf die auszugebenden Zeichenketten nicht mittels Tastatur eingegeben, sondern per Drag'n Drop abgelegt wurden.

In Abbildung 16 ist ein Panel zu sehen, das für zwei Programme, die mit jeweils einer `for-to`-Schleife bzw. einer `while-do`-Schleife realisiert wurden, gleich ist. Beide Programme verhalten sich gleich. Sie nehmen aus „Data-Entry1/2“ zwei Zahlenwerte entgegen, die als untere und obere Schleifengrenze fungieren sollen. Beim Click auf „Pushbutton1“ wird die Schleife abgearbeitet und die Werte der Laufvariablen in „Grid1“ ausgegeben. Die Abbildungen 17 und 18

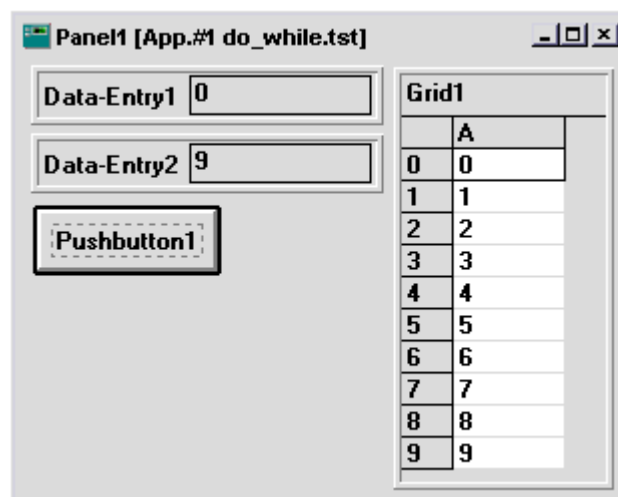


Abbildung 16: statements for-to und while-do: Panel

zeigen jeweils die Action-List der „Pushbutton1“-Objekte in beiden Programmen. Das Programm in Abbildung 17 muss wohl nicht weiter erläutert werden. In dem in Abbildung 18 dargestellten Programm werden zwei Instanzen des TestPoint-Objektes „Math“ verwendet. In

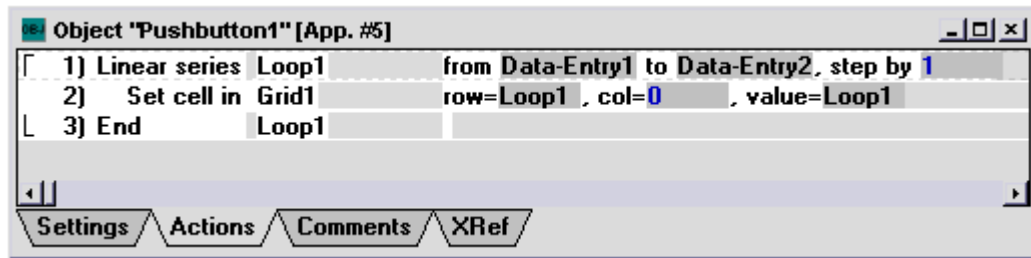


Abbildung 17: for-to-statement Action-List, Pushbutton1

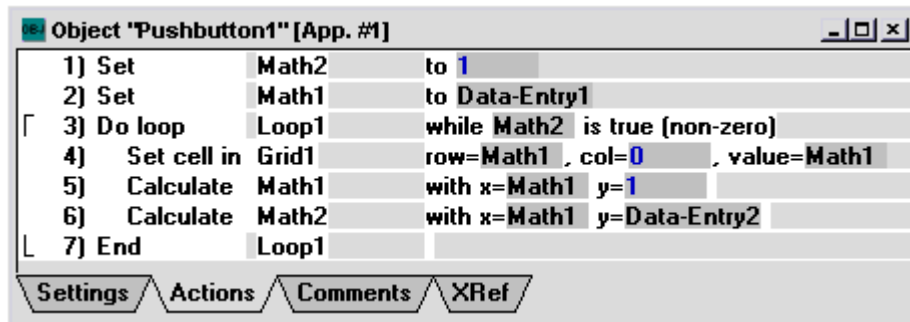


Abbildung 18: while-do-statement Action-List/ „Pushbutton1“

„Math1“ wird die Aufsummierung der Laufvariablen erledigt und in „Math2“ die Laufvariable mit der oberen Schlingengrenze verglichen. Zunächst werden beide Math-Objekte initialisiert. „Math2“ erhält den Wert 1, weil als Laufkriterium der Schleife „Math2“ ungleich 0 sein muss. „Math1“ hat als Wert zunächst den Startwert aus „Data-Entry1“ und dieser wird in Zeile 4 in das Grid ausgegeben. In den Zeilen 5 und 6 wird der Wert von „Math1“ um 1 erhöht bzw. der Wert von Math1 mit der oberen Grenze verglichen. Erwähnenswert ist, dass die zweifellos vorhandene Laufvariable nicht explizit deklariert werden musste, sondern Teil der beteiligten Objekte ist. Abbildung 19 zeigt das Objects-Fenster von „Panel1“ des eben beschriebenen Programmes. Das Objekt „Math“ ist auch dahingehend interessant, als mit ihm sowohl numerische Daten als auch Zeichenketten verarbeitet werden können.

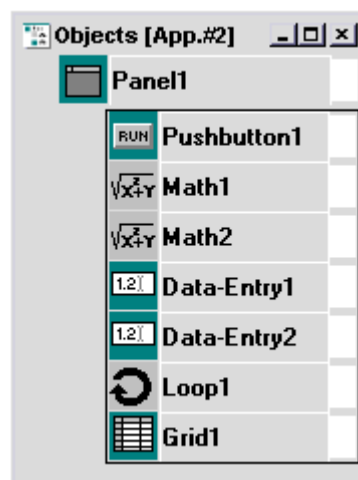


Abbildung 19: Objects-Fenster

In den Abbildungen 20 und 21 ist ein Programm zu sehen, das über die serielle RS-232 Schnittstelle mit einem Multimeter M-3610D kommuniziert. Anfangs gab es Probleme mit der Konfiguration der seriellen Schnittstelle.

Dies konnte aber nach Rücksprache mit Herrn Jürgen Schreiner <sup>4</sup> und Herrn Andreas Köllmann <sup>5</sup>, beide ihrerseits von Keithley zertifizierte TestPoint-Ingenieure, behoben werden. Die seriellen Schnittstellen sind laut Aussage von Herrn Köllmann unter TestPoint nicht immer unproblematisch. Im vorliegenden Aufbau wurde zunächst ein Multimeter verwendet und später zwei gleichzeitig, wobei beide an jeweils einem seriellen Port betrieben wurden. Das ganze funktionierte sehr gut, wobei die Datenübertragungsrate der Multimeter weniger als ein Messwert pro Sekunde beträgt.



Abbildung 20: Auslesen des Multimeters - Programmoberfläche

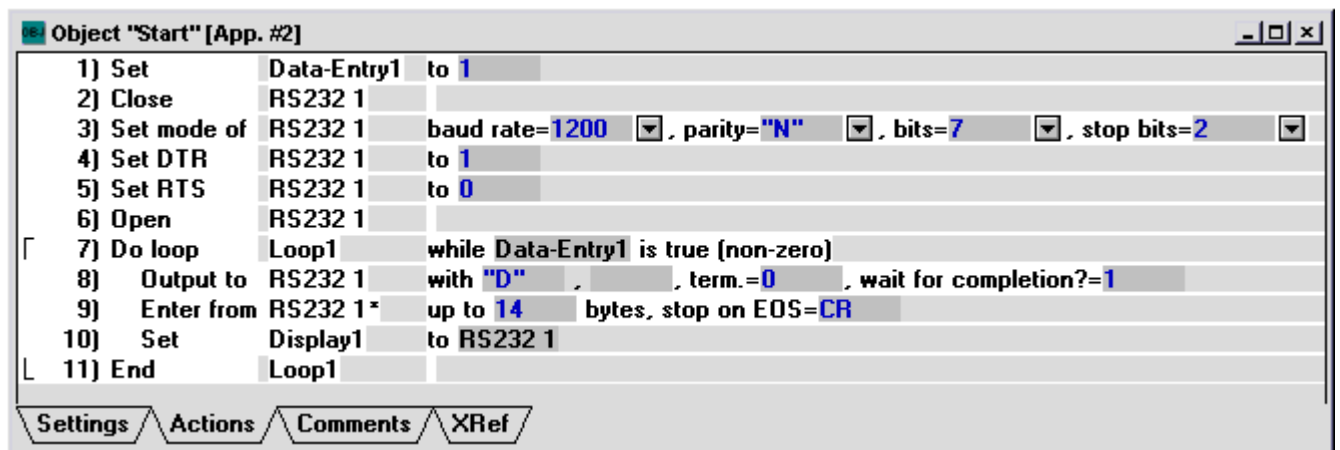


Abbildung 21: Auslesen des Multimeters - Programmcode

<sup>4</sup><schreiner.juergen@T-Online.de>

<sup>5</sup><akoellmann@aol.com>

## 5.5 Das Testprogramm

Um die beiden Programmsysteme direkt vergleichen zu können, wurde ein ähnliches Testprogramm wie in Vee auch in TestPoint entwickelt. Es sollte eine ähnliche Funktionalität, wie das unter 5.1 beschriebene Programm aufweisen, was auch gelang. Abbildung 22 zeigt das Panel des Programms. Die Parameter des Messgerätes werden in einer Header-Datei abgelegt, deren Name sich aus Datum und Uhrzeit zusammensetzt. Die reinen Messdaten werden in ebenfalls in eine Textdatei geschrieben, wobei Inhalt und Aufbau beider Dateien beliebig sind. Die Namen der Datendatei und der Headerdatei sind gleich um zuordenbar zu sein. Unterschiedlich sind nur die Dateierweiterungen.

Beispiel:

Headerdatei: d:\normank\data\testpoint\140103\_152420.had

```

TestPoint 4.1
Datendatei [d:\normank\data\testpoint\140103_152420.tpd]
PREMA 6047
Wechselspannung 0.2V
Integrationszeit 400ms
Intervallbetrieb [10s]
Zeitbetrieb von: [15:24:30] bis [15:25:30]
Startzeit: [15:22:45]
Stopzeit: [19:15:01]

```

Datendatei: d:\normank\data\testpoint\140103\_152420.tpd

```

1      11:53:33 31/01/03      0
2      11:53:34 31/01/03      0
3      11:53:34 31/01/03      0
4      11:53:35 31/01/03      0
5      11:53:35 31/01/03      0
6      11:53:35 31/01/03      0
7      11:53:36 31/01/03      0
8      11:53:36 31/01/03      0
9      11:53:37 31/01/03      0

```

Die Abbildungen 27, 28 und 29 zeigen zwei Programmablaufpläne, die den Programmaufbau exemplarisch an zwei wesentlichen Routinen verdeutlichen sollen. Abbildung 27 zeigt den Aufbau der Prozedur, die beim Einschalten des Schalters „Messen“ ausgeführt wird. In Abhängigkeit von der Betriebsart: „Intervallbetrieb“/„Dauerbetrieb“ der jeweilige Task aufgerufen. Abbildung 28 zeigt den Aufbau der beiden Tasks. Abbildung 29 zeigt die Struktur der Prozedur, die das Multimeter mit festgelegten Startwerten Initialisiert.

## 6 Fazit

Die Einarbeitung in das Programmierkonzept von Vee verlief schneller und intuitiver, bei der Bedienung von TestPoint ist ein Handbuch hilfreich. Vee ist durch das auf den ersten Blick erkennbare Programmierkonzept leicht erlernbar und auch für die grafische Darstellung einfacher Konstrukte besser geeignet. Die Entwicklung des Vee-Testprogrammes dauerte dann aber wesentlich länger, als die des TestPoint-Programmes. Dies ist sowohl auf die konzeptionellen Unterschiede beider Systeme zurückzuführen, als auch auf den Umstand, das zuerst das Vee-Programm entstand und bei der Arbeit mit TestPoint die Erfordernisse und Herangehensweisen (z.B. beim Ansprechen des Multimeters) bereits bekannt waren. Es lässt sich aber sagen, dass die Entwicklung mit TestPoint, abgesehen von den anfänglichen Schwierigkeiten mit der Anbindung der Peripherie, ungleich schneller vonstatten geht, als mit Vee und die sich Komplexität der entstehenden Programme eher in Grenzen hält.

Als Beispiel führe ich hier die Erstellung benutzerdefinierter Funktionen an. In Vee wird das Auslagern vorhandenen Codes in mehrfach verwendbare Module eher dadurch erschwert, dass dem Nutzer nicht nachvollziehbare Hürden, z.B. bei der Variablenübergabe, in den Weg gelegt werden, wodurch unnötig viel Code im Hauptprogramm vorgehalten wird. In TestPoint ist das bewusste Auslagern von Code zunächst gar nicht nötig, da ja bereits jedes verwendete Objekt einen eigenen Code-Teil (Action-List) besitzt. Dadurch entsteht auch ohne Zutun des Nutzers eine modulare Programmstruktur, die sich gut warten lässt. Da ein Objekt durchaus mehrfach verwendet werden kann, wird der Objektcode zur Funktion/Prozedur. Insbesondere bei grösseren Projekten ist auch die Objekt-Referenzliste sehr hilfreich, da sie den Überblick über das Programm erleichtert.

Die Vorteile von TestPoint zeigen gleichzeitig auch die Nachteile von Vee auf. Der wesentliche Nachteil ist, wie bereits mehrfach erwähnt, die unverhältnismässige Komplexität selbst einfacher Routinen, die die scheinbaren Vorteile der visualisierten Programmstruktur schnell zunichte macht und die Programmierung sehr erschwert. So vermittelt das fertige Vee-Programm den Eindruck, den die Schaltkarte eines Kraftwerks auf einen Laien macht. Auch die Wartbarkeit und die Wiederverwendbarkeit einmal erstellten Codes leiden darunter.

Vee hat in dieser Hardware- und Softwarekonstellation mit seiner völlig problemlosen Hardwareanbindung ein grosses Plus für sich verbuchen können. Ob dieses Verhalten aber ein realer Mehrwert gegenüber TestPoint ist, oder nur Zufall, konnte hier nicht umfassend geklärt werden, da hierfür Tests mit den Produkten von mehr als nur zwei Herstellern nötig wären. Für die schnelle Anbindung von Messgeräten ist TestPoint deshalb zu bevorzugen.

## 7 Anhang

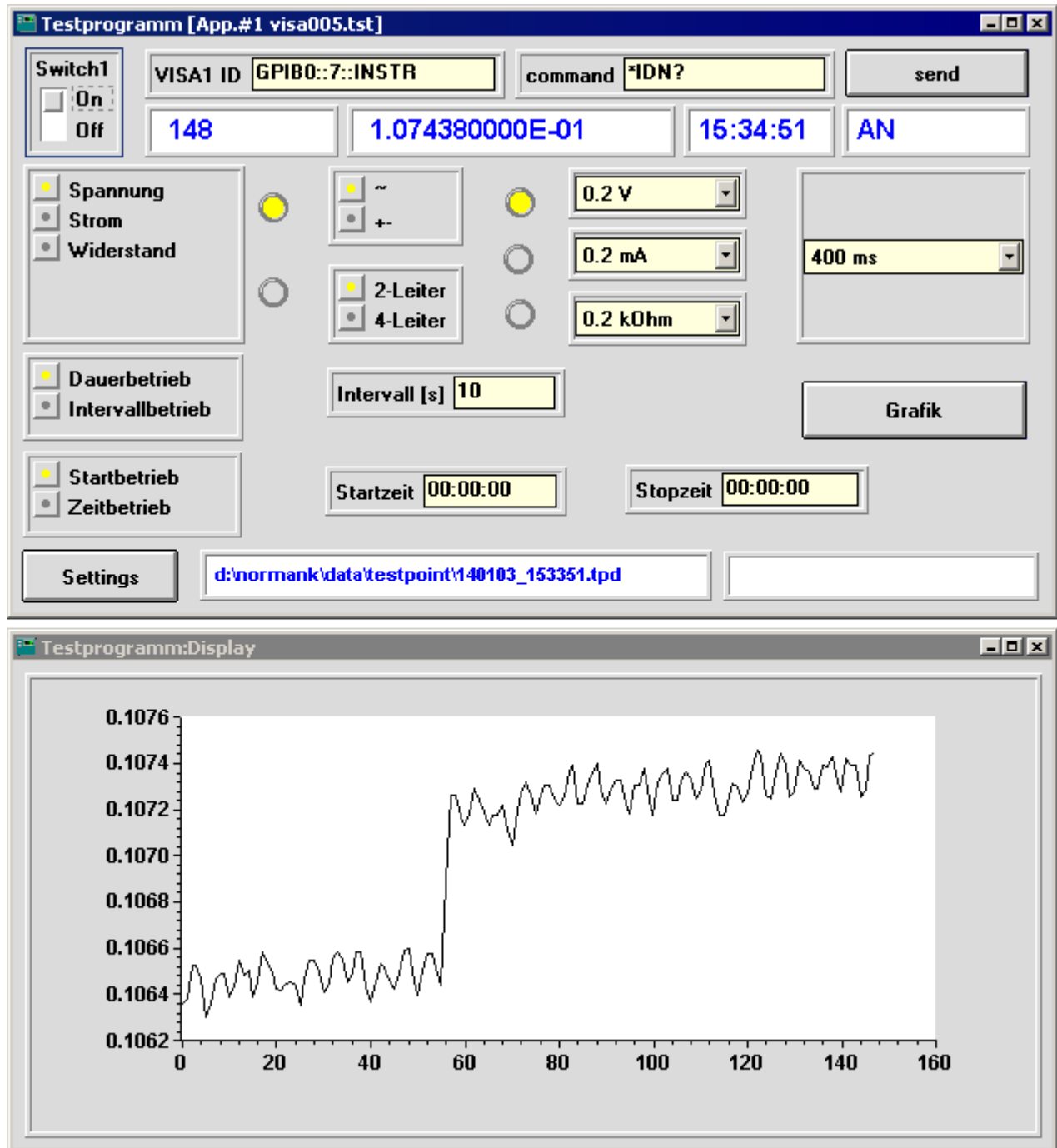


Abbildung 22: TestPoint Testprogramm

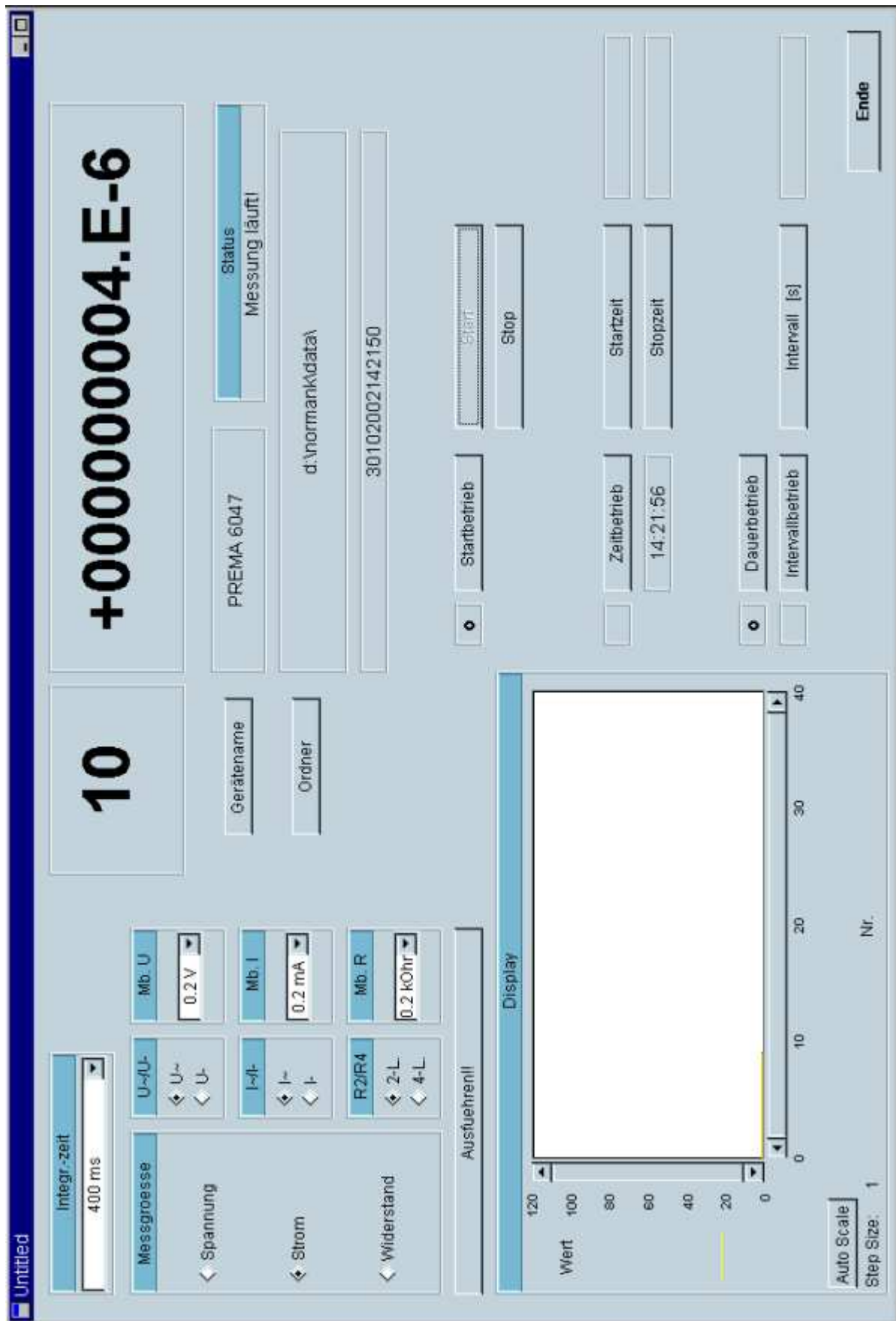
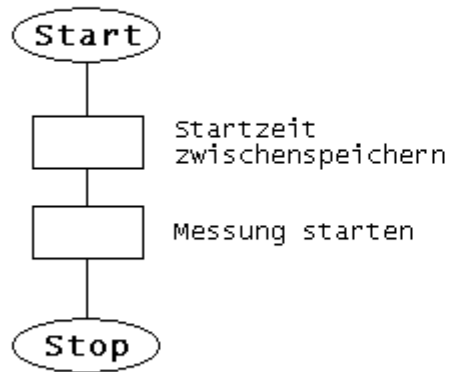


Abbildung 23: Vee Testprogramm

### Startbetrieb & Zeitbetrieb

#### Startbutton



#### Stopbutton

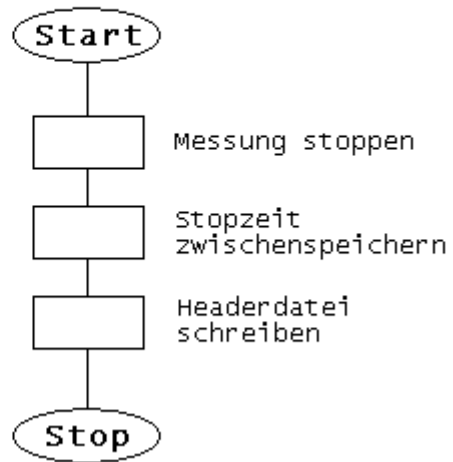
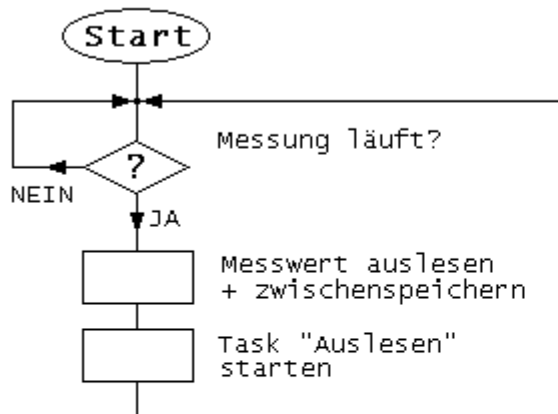


Abbildung 24: Programmablaufplan Vee - Starten/Stoppen

#### Messen



#### Auslesen

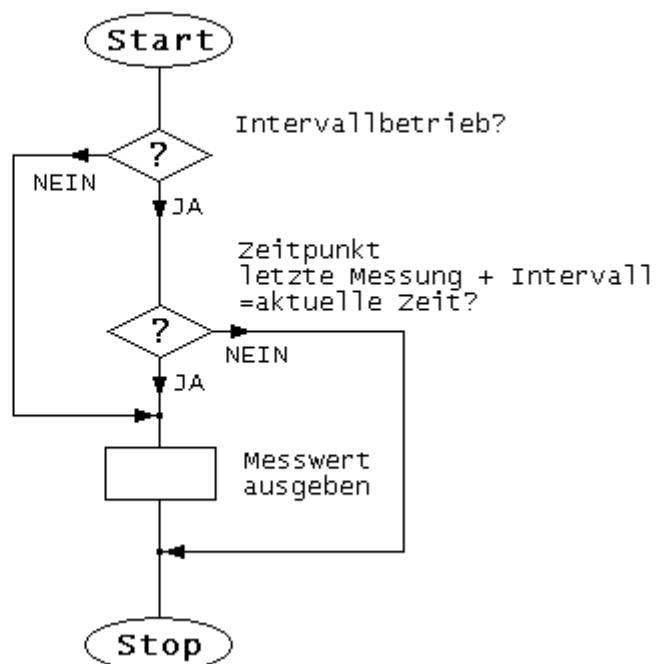


Abbildung 25: Programmablaufplan Vee - Tasks: Auslesen/Messen

## Zeitbetrieb

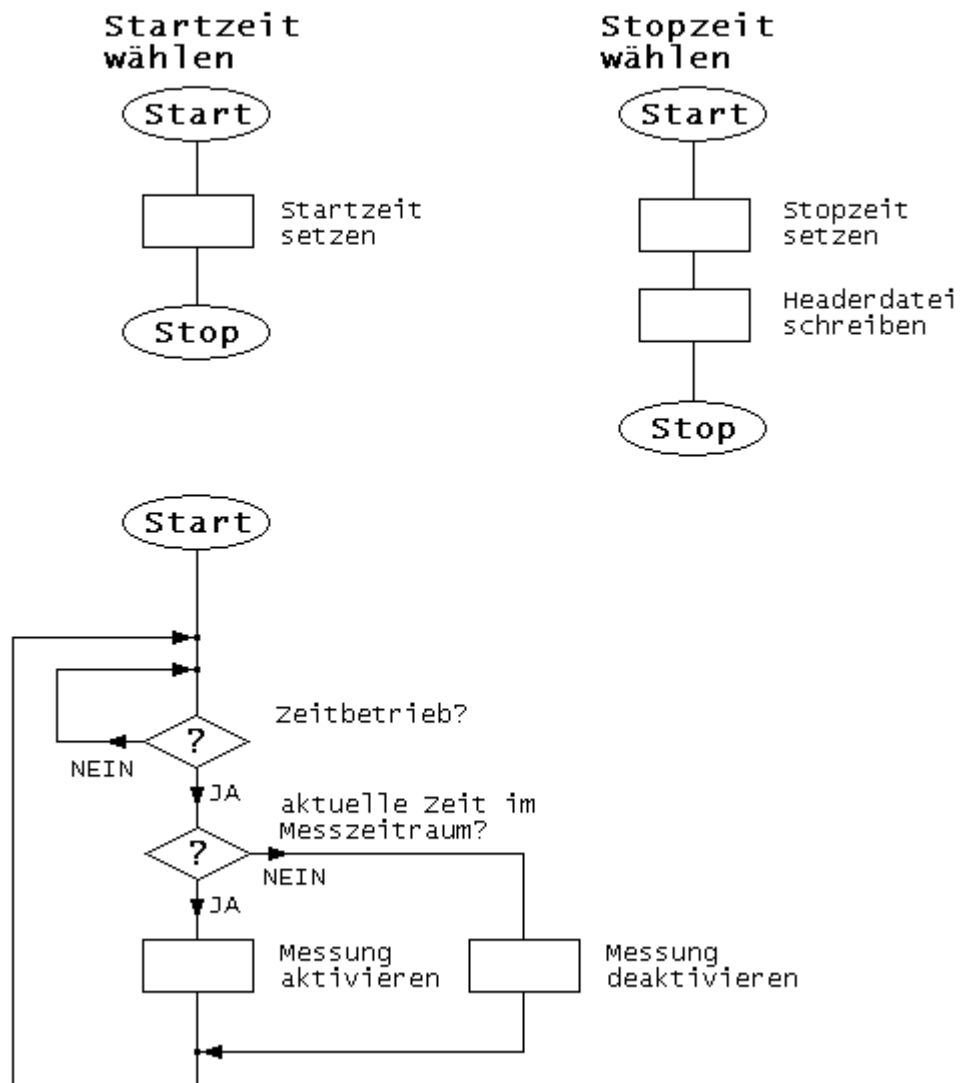


Abbildung 26: Programmablaufplan Vee - Startzeit;Stopzeit/Task: Zeitbetrieb

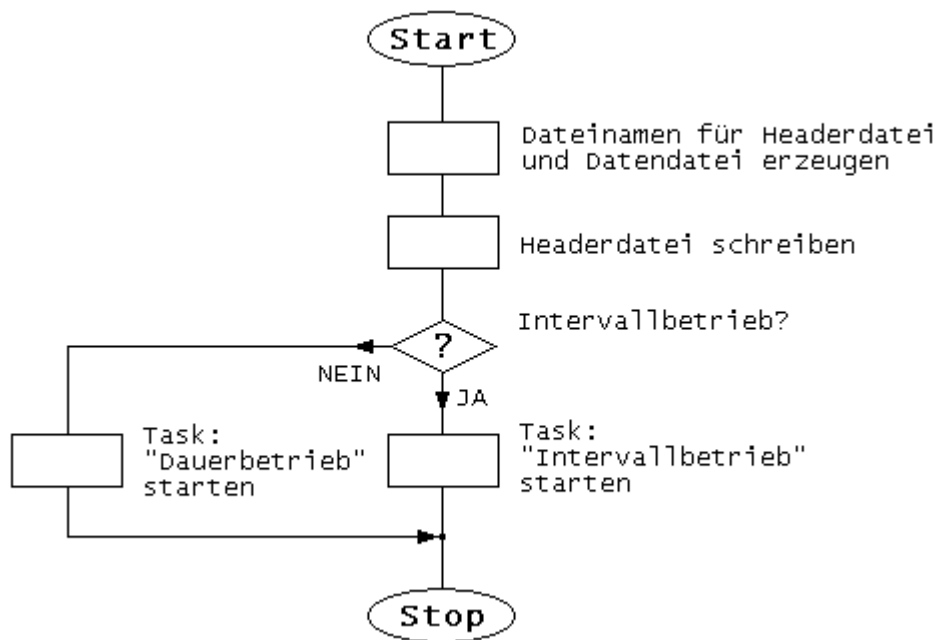


Abbildung 27: Programmablaufplan TestPoint - „Messen“

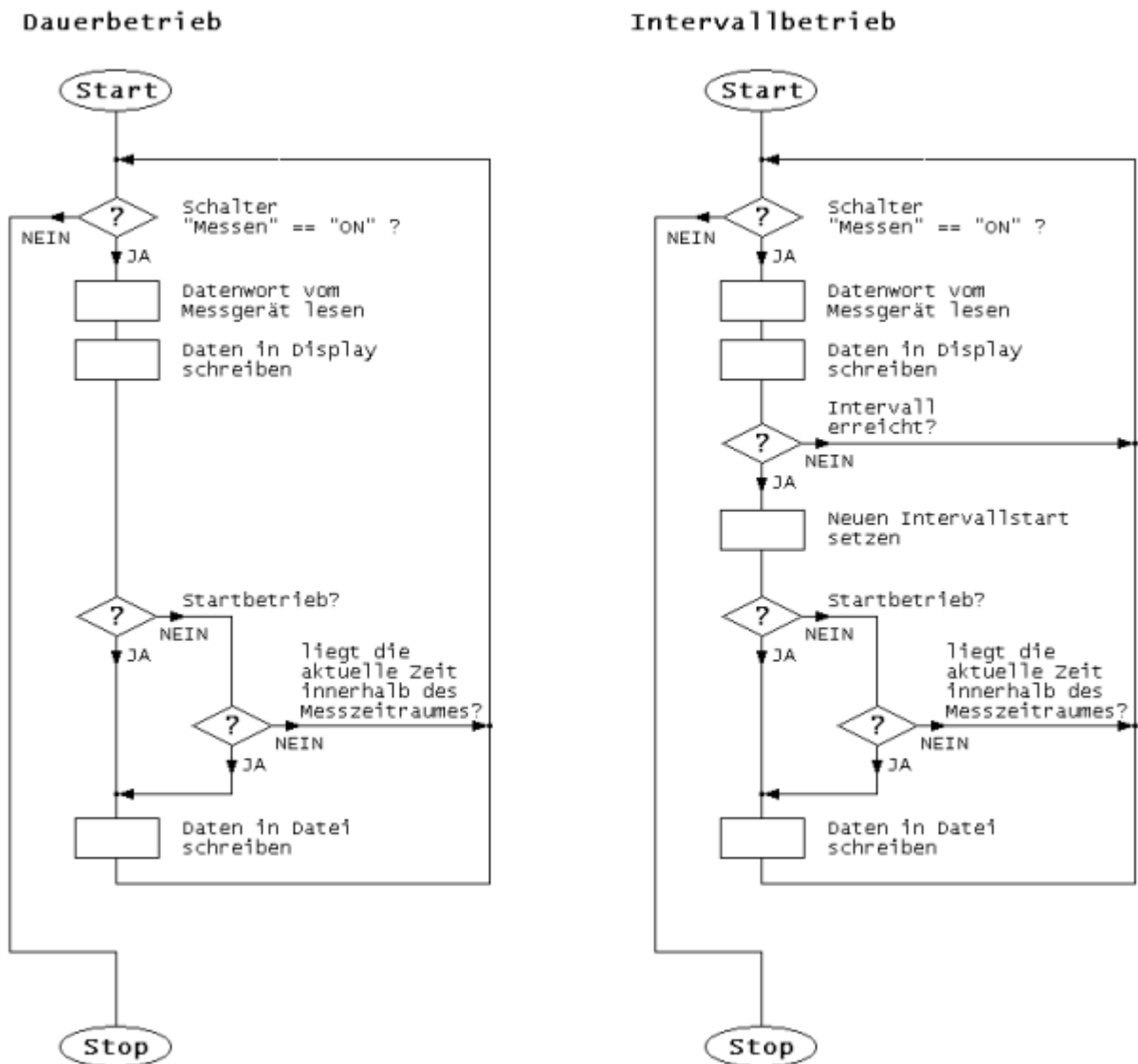


Abbildung 28: Programmablaufplan TestPoint - Dauer-/Intervall- und Start-/Zeitbetrieb

## Initialisieren

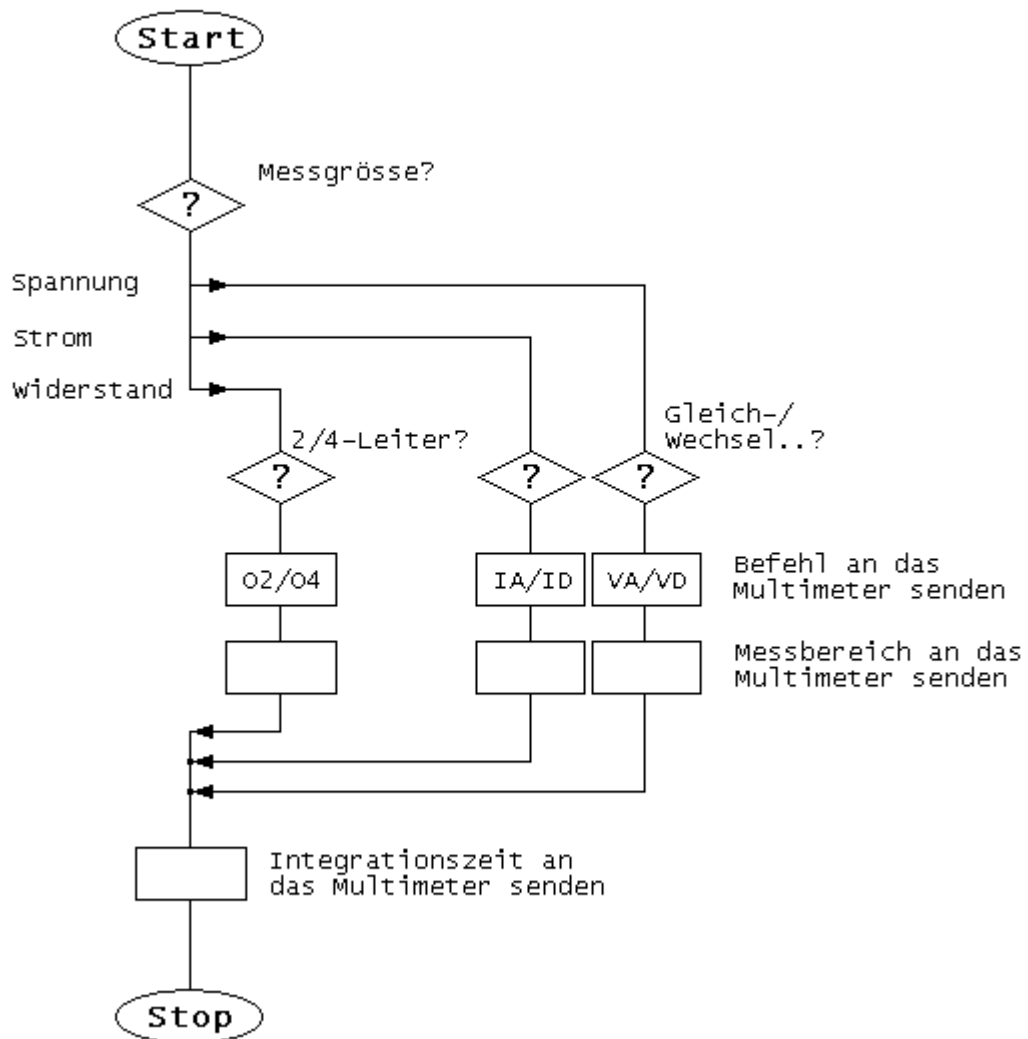


Abbildung 29: Programmablaufplan TestPoint - Initialisierung

## Literatur

- [1] KEITHLEY. *TestPoint - Benutzerhandbuch*. Keithley Instruments GmbH.
- [2] KEITHLEY. *TestPoint - Techniques & Reference*. Capital Equipment Corp. 1996.
- [3] HP VEE. *Controlling Instruments with HP Vee*. Hewlett-Packard Company 1997.
- [4] HP VEE. *HP Vee Advanced Programming Techniques*. Hewlett-Packard Company 1997.
- [5] HP. *HP Multimeter 34401A - Bedienungsanleitung*. Hewlett-Packard Company 1992.
- [6] PREMA. *Digitalmultimeter DMM 6047 - Benutzerhandbuch*. PREMA GmbH
- [7] MICHAEL TISCHER. *PC-Intern 5*. Data Becker 1997.
- [8] ROBERT SEDGEWICK. *Algorithmen*. Addison Wesley 1997.
- [9] W.BEITZ UND K.-H. GROTE. *Dubbel, Taschenbuch für den Maschinenbau*. Springer 1997.