Hochschule Wismar
University of Technology, Business and Design
Fachbereich Wirtschaft

Hochschule Wismar
University of Technology, Business and Design
Faculty of Business

Harald Mumm

Unterbrechungsgesteuerte Informationsverarbeitung

Heft 11 / 2004

W D P

Wismarer Diskussionspapiere / Wismar Discussion Papers

Der Fachbereich Wirtschaft der Hochschule Wismar, University of Technology, Business and Design bietet die Präsenzstudiengänge Betriebswirtschaft, Management sozialer Dienstleistungen, Wirtschaftsinformatik und Wirtschaftsrecht sowie die Fernstudiengänge Betriebswirtschaft, International Management, Krankenhaus-Management und Wirtschaftsinformatik an. Gegenstand der Ausbildung sind die verschiedenen Aspekte des Wirtschaftens in der Unternehmung, der modernen Verwaltungstätigkeit im sozialen Bereich, der Verbindung von angewandter Informatik und Wirtschaftswissenschaften sowie des Rechts im Bereich der Wirtschaft.
Nähere Informationen zu Studienangebot, Forschung und Ansprechpartnern finden Sie auf unserer Homepage im World Wide Web (WWW): http://www.wi.hs-wismar.de/.

Herausgeber: Prof. Dr. Jost W. Kramer
Fachbereich Wirtschaft
Hochschule Wismar
University of Technology, Business and Design
Philipp-Müller-Straße
Postfach 12 10
D – 23966 Wismar
Telefon: ++49/(0)3841/753 441
Fax: ++49/(0)3841/753 131
e-mail: j.kramer@wi.hs-wismar.de

Vertrieb: HWS-Hochschule Wismar Service GmbH
Phillipp-Müller-Straße
Postfach 12 10
23952 Wismar
Telefon:++49/(0)3841/753-574
Fax: ++49/(0) 3841/753-575
e-mail: info@hws-startupfuture.de
Homepage: www.hws-startupfuture.de

ISSN 1612-0884
ISBN 3-910102-54-9

JEL-Klassifikation Z00

Inhaltsverzeichnis

1 Einleitung

Der Gegenstand dieser Arbeit ist, die unterbrechungsgesteuerte Informationsverarbeitung an einem einfachen Anwendungsbeispiel zu erläutern, damit sie im Unterricht des Studienganges Wirtschaftsinformatik in kurzer Zeit behandelt werden kann. Diese Art der Informationsverarbeitung steht ungerechtfertigterweise oft nicht im Mittelpunkt des wissenschaftlichen Interesses. Aber spätestens seit den aktuellen Diskussionen und Schwierigkeiten in Deutschland um das LKW–Maut–System fragen sich viele Interessierte, warum die Realisierung so schwierig ist und wie derartige Systeme prinzipiell funktionieren. Aber auch in viel kleineren Anwendungen, wie z.B. in der Software eines Fahrrad–Ergometer–Rechners tritt diese Art der Informationsverarbeitung auf. Derartige Software ist im Gegensatz zu wissenschaftlich technischen oder auch komplizierten ökonomischen Programmen relativ einfach. Allerdings muß sie durch verschiedene externe Geräte unterbrechbar sein. Fährt z.B. ein LKW unter einer Maut–Brücke hindurch, wird die Maut fällig und das aktuelle Programm in der Onboard–Unit muß zugunsten eines Abrechnungsprogrammes unterbrochen werden. Ebenso sind während der Fahrt eines LKW laufend Signale des GPS–Systems zur satellitengesteuerten Navigation auszuwerten, damit die Software überprüfen kann, ob sich der LKW auf einer mautpflichtigen Straße befindet oder nicht. Die empfangenen Koordinaten sind dazu mit gespeicherten kartographischen Daten abzugleichen.

Auch der schon erwähnte Rechner eines Ergometers hat Aufgaben dieser Art zu lösen. Er zeigt z.B. den Puls, die Uhrzeit und die zurückgelegte Entfernung an. Ein Pulsgeber, eine Uhr und ein spezielles Gerät in der Nabe des Ergometer–Rades lösen ständig Unterbrechungen aus: Der Pulsgeber, wenn das Herz des Radlers schlägt, die Uhr, wenn sie tickt und das Rad, wenn es sich einmal herum gedreht hat. Damit der Rechner auf diese Unterbrechungen reagieren kann, muß er mit den dazugehörigen Geräten verbunden sein und seine Software muß diese Unterbrechungen zählen und diese Zahlen regelmäßig anzeigen.

Diese Arbeit soll Software für einen Fahrrad–Ergometer–Rechner entwickeln, damit er über ganz einfache Funktionen zur Anzeige der gezählten Ticks der Uhr und des Pulses des Sportlers verfügt. (Diese beiden Werte werden exemplarisch verwendet.) Wir bedienen uns hierzu wieder der diadaktischen Neumann–Maschine Paula aus [3] und des Simulators dieser Maschine aus [2].

Am Ende soll eine Simulation eines derartigen Rechners, die über das Internet aufrufbar ist, die Tragfähigkeit der verwendeten Konzepte demonstrieren. Eine Uhr und ein Pulsgeber sind Bestandteile der Simulation. Der Puls kann über zwei Schaltknöpfe erhöht oder verringert werden.

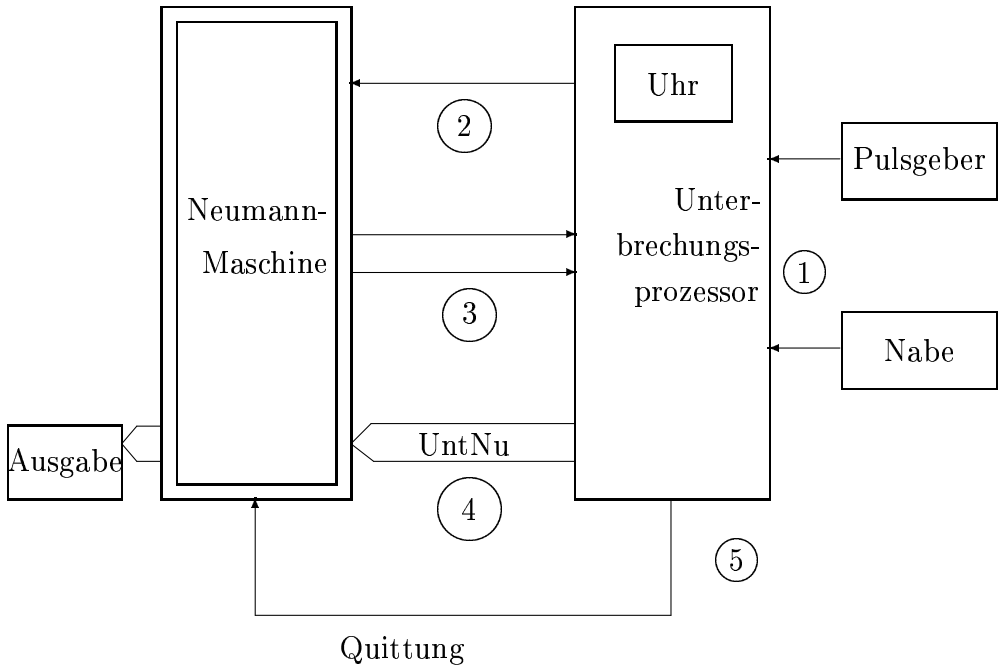
Als Implementierungssprache dient wieder die in [1] vorgestellte objektorientierte Entwurfssprache.

Der dafür selbst entwickelte Compiler dient der Übersetzung der Programme in die Maschinensprache der Neumann–Maschine Paula und ein in Java implementierter Simulator erlaubt die Abarbeitung des Beispielprogramms.

2 Der Anschluß der externen Geräte

Beim folgenden Bild handelt es sich um eine Abstraktion einer Abbildung aus [4], weil einige der dort betrachteten Fälle im vorliegenden Beispiel nicht vorkommen können, z.B. die Übernahme von Werten aus den Geräten Pulsgeber und Nabe. Sie lösen hier nur Unterbrechungen aus. Die Neumann–Maschine übernimmt nur vom Unterbrechungsprozessor gewisse Werte, nämlich die Identifizierungsnummer 'UntNu' einer Unterbrechung.

Externe Geräte wie Pulsgeber und Nabe sind über einen Unterbrechungsprozessor, der eine Uhr enthält, am Rechner (Neumann–Maschine) angeschlossen. Erfolgt ein Unterbrechungswunsch (1 oder 2) signalisiert der Rechner über zwei Bits (3), daß er eine Eingabe vom Unterbrechungsprozessor erwartet (die Nummer der Unterbrechung zu ihrer Identifizierung). Der Unterbrechungsprozessor stellt diese Nummer bereit (4) und signalisiert die Bereitstellung mit einem Bit (5), das hier Quittung heißt.



Legende: UntNu ist die Nummer der Unterbrechung und kann den Wert 50 (Uhr), 51 (Pulsgeber) oder 52 (Nabe) annehmen.

Abbildung 1: Schematische Darstellung des Anschlusses der externen Geräte

3 Der Aufbau der Software

Die Demonstrationssoftware soll objektorientiert entwickelt werden. Dazu werden die Klassen „Prozess“, Strom und Betrieb für die Verwalter–Software (Programm Ergometer) und S_Neutral für die Anwendungssoftware definiert. Die Schnittstellen der vom Programm Ergometer bereitgestellten Unterprogramme werden einem Anwendungsprogramm in der Klasse S_Neutral bekanntgegeben. Ihr Aufruf aus einem Anwendungsprogramm heraus wird Systemaufruf genannt.

Die Klasse S_Neutral ist insofern eine besondere Klasse, da sie nur aus Schnittstellendefinitionen der Methoden der Klasse Betrieb besteht. Sie ist notwendig, weil die Systemaufrufe hier nicht in Dienstaktionen versteckt werden sollen und der Compiler ihre Schnittstelle für die Syntaxprüfung ihres Aufrufes benötigt.

Wenn der Compiler in einem Anwendungsprogramm den Aufruf einer Methode aus dieser Klasse entdeckt, generiert er neben Anweisungen zur Kopie der Werte der aktuellen Parameter eine prozeßwechselnde Paula–Maschinen–Anweisung mit einem entsprechenden Parameter. Dieser Parameter heißt im Ergometer–Programm Aufruf und wird in der Methode setzeInDienstModus ausgewertet. Bevor die Unterprogramme des Programmes Ergometer im Auftrag eines Kunden abgearbeitet werden können, wird ihnen in der Methode setzeInDienstModus noch ein zusätzlicher Parameter übergeben, der die sogenannten Betriebsdaten bereitstellt.

In dieser Arbeit dient die Klasse Betrieb der Beschreibung der verwendeten Betriebsdaten, die hier aus den folgenden fünf Variablen bestehen:

1. Der Anzahl registrierter Uhrticks (ticks),
2. der Anzahl registrierter Drehungen des Rades (dreh),
3. der Anzahl registrierter Herzschläge (puls),
4. dem Puls (panz) für Herzschläge pro Minute und
5. der Adresse des gerade laufenden Prozesses (bereit).

Die Variable dreh wird in dieser Arbeit nicht weiter verwendet. Sie dient lediglich der Vorbereitung für den Fall, daß auch der zurückgelegte Weg in einer späteren Ausbaustufe gemessen werden soll.

Wir gehen hier von 6 Ticks pro Minute aus, d.h. nach 6 Ticks steht in der Variablen puls die medizinische Größe Puls (Herzschläge pro Minute). Nach 6 Ticks wird der Wert der Variablen puls in die Variable panz übernommen und die Variable puls auf Null gesetzt.

Die Nachrichten der Klasse Betrieb, die auch von Anwendungsprogrammen heraus sendbar sein sollen, heißen hier F1 und F2.

Wird im Anwendungsprogramm KundeA eine dieser Nachrichten an ein vorhandenes Objekt vom Typ S_Neutral gesendet, erfolgt ein Systemaufruf. Seine genaue Beschreibung erfolgt im nächsten Abschnitt.

Die Objekte der Klasse „Prozess“ verwalten die Registerinhalte eines Prozesses nachdem er unterbrochen wurde. Vor einem Wiederanlauf müssen diese Werte in die Register der Neumann–Maschine geladen werden. Diese Aufgabe

übernimmt das Mikroprogramm der Neumann–Maschine Paula bei Interpretation der dafür vorgesehenen Maschinenanweisung durch das Mikroprogramm. Sie wird vom Compiler bei der Übersetzung der Methode werdeAktiv beim Erreichen der Codegenerierung für den Befehl aktiviere generiert.

3.1 Die Simulation der Geräte

In der vorliegenden Ausbaustufe werden nur die Geräte Uhr und Pulsgeber unterstützt. Zu ihrer Simulation werden im Simulatorprogramm zwei Threads verwendet, die für eine bestimmte Zeit Schlafen gelegt werden und nach dem Aufwecken eine Unterbrechung der simulierten Neumann–Maschine auslösen, wodurch der gerade laufende Prozeß zugunsten der Unterbrechungsbehandlung unterbrochen wird. Zur Anzeige des Pulses muß der entsprechende Wert die Neumann–Maschine verlassen. In der vorliegenden Lösung wird diese Ausgabe vom Simulatorprogramm abgefangen und in seinem Ausgabefenster dargestellt.

4 Die Beschreibung der Funktionalität der Systemaufrufe

In diesem Abschnitt sollen lediglich diejenigen Systemaufrufe beschrieben werden, die der Verleihung von Kompetenz an ein Anwendungsprogramm durch den Verwalter Ergometer dienen.

Das Programm Ergometer stellt die Methoden F1 und F2 dem Anwendungsprogramm KundeA als Dienstleistung zur Verfügung, die über Systemaufrufe ausgeführt werden können.

Die erste Methode F1 dient der Übernahme einer Zahl aus den Betriebsdaten des Programmes Ergometer in einen Datenbereich des Anwendungsprogrammes KundeA, damit der KundeA diese Zahl z.B. ausgeben (anzeigen) kann.

Die zweite Methode F2 dient der Ausgabe (Anzeige) einer Binärzahl. Dazu muß sie zuerst in eine Zeichenkette umgewandelt werden. Zeichen für Zeichen wird dann anschließend (durch Angabe des ASCII–Codes) an den Simulator übergeben, der es in sein Ausgabefenster schreibt.

5 Die beteiligten Prozesse

Im Demonstrationsbeispiel werden außer einem Verwalter (Ergometer) zwei Prozesse (KundeA und KundeB) im Wechsel abgearbeitet.

5.1 Der Verwalter

Beim Booten (Starten) werden die Maschinencode–Dateien der Programme Ergometer, KundeA (Leerlauf) und KundeB (Anzeige) in den Hauptspeicher der Neumann–Maschine Paula geladen. Das Programm Ergometer dient als Verwalter und sorgt dafür, daß KundeA anlaufen kann. Durch Unterbrechungswünsche

seitens der Geräte Uhr, Pulsgeber und Nabe wird KundeA unterbrochen und dadurch dem Programm Ergometer das Zählen dieser Unterbrechungen ermöglicht. Der Verwalter kann die Unterbrechungen nicht selber zählen, weil er nicht unterbrechbar ist. Nachdem der KundeA ca. eine Minute gelaufen ist, ruft der Verwalter den Prozeß KundeB auf, der die relevanten Werte (zur Zeit nur den Puls) anzeigt. Anschließend soll der Verwalter wieder den Prozeß von KundeA starten, damit wieder auf Unterbrechungen reagiert werden kann.

5.2 Der KundeA (Leerlauf)

Das Programm KundeA dient eigentlich nur dem Versetzen des Rechners in einen unterbrechbaren Zustand. Es ist hier als Schleife definiert. Der Abbruchwert sollte so gewählt werden, daß KundeA ca. 70 Sekunden läuft, damit der Radler nicht zu lange auf die Anzeige seines Pulses warten muß.

5.3 Der KundeB

Der KundeB dient der Anzeige relevanter Werte, wie z.B. dem Puls auf einem Ausgabegerät. Da die Anzeige über den Systemaufruf F2 erfolgt, ist KundeB während der Laufzeit der Funktion F2 nicht unterbrechbar. Damit keine Unterbrechungen verloren gehen, sollte die Abarbeitung der Funktion F2 zwischen zwei Ticks möglich sein.

6 Der Quelltext

Nachfolgend wird der Quelltext der beteiligten Programme Ergometer, KundeA und KundeB abgedruckt. Das Programm Ergometer dient der Verwaltung zweier Kunden und der „intelligenten“ Unterbrechungssteuerung. Intelligent heißt, daß der gerade laufende Kunde so unterbrochen wird, daß er nach der Unterbrechung genauso weiterlaufen kann, als wenn es gar keine Unterbrechung gegeben hätte. Mit Verwaltung der Kunden ist einfach ein zyklischer Wechsel zwischen den beiden Kunden gemeint.

Das Programm KundeA dient dem Nichtstun. Es ist immer unterbrechbar. Das Programm KundeB dient der Anzeige relevanter Werte.

```
Programm Ergometer;
Konstante OffsetEpa=3;
Konstante KundeA=16777102;
//Adresse des KundenA
Konstante KundeB=16777211;
//Adresse des KundenB

Typ Prozess = Klasse (
  Epa   : Adresse;
  Nia   : Adresse;
  Basis : Adresse;
  Akkuw : ganzzahl;

Methode setzeNia(-> N: ganzzahl):
                         ganzzahl;
Beginn
 dieses := dieses - OffsetEpa;
    Nia := N ;
 liefereAlsFunktionswert(0);
Ende;



Methode holeBasis(-> N: ganzzahl)
                       : Adresse;
Beginn
 dieses := dieses - OffsetEpa;
 liefereAlsFunktionswert(Basis);
Ende;
Methode setzeEpaNegativ:ganzzahl;
Beginn
 dieses := dieses - OffsetEpa;
    Epa := 0-Epa;
 liefereAlsFunktionswert(0);
Ende;

Methode setzeAufStart: ganzzahl;
  Beginn
    dieses := dieses - OffsetEpa;
    Nia := Epa ;
    liefereAlsFunktionswert(0);
Ende;

Methode setzeInDienstModus(
             ->Aufruf: ganzzahl;
       ->bv: Adresse):ganzzahl;
Variable Uebergabe: Adresse;
Variable  KDA: Adresse;
Variable rc: ganzzahl;
Beginn
    Uebergabe := holeBasis+2;
    //RSA,FW,dieses,1.Par.
    Uebergabe^ := bv;
    falls Aufruf = 1 dann
       rc := setzeNia(9999);
    falls Aufruf = 2 dann
       rc := setzeNia(9998);
    rc := setzeEpaNegativ;
Ende;
Methode werdeAktiv:
             ganzzahlImAkku;
Beginn
   aktiviere;
Ende;
                     );

Typ Betrieb = Klasse(
     ticks : ganzzahl;
     dreh  : ganzzahl;
     puls  : ganzzahl;
     panz  : ganzzahl;
     //Anzeigewert Puls
     bereit: Prozess;

Methode holeBereit : Prozess;
Beginn
liefereAlsFunktionswert(bereit);
Ende;



Methode setzeBereit(-> p: Prozess)
                       : ganzzahl;
Beginn
bereit := p;
liefereAlsFunktionswert(0);
Ende;

Methode F1(-> wovon: ganzzahl) :
ganzzahl;
Variable Ergebnis: ganzzahl;
Beginn
 falls wovon = 1 dann
    Ergebnis := dreh;
 falls wovon = 2 dann
    Ergebnis := panz;
```

```
 falls wovon = 3 dann
    Ergebnis := ticks;
liefereAlsFunktionswert(Ergebnis);
 S_Fertig;
Ende;

//Anzeige von Werten
Methode F2(->z: ganzzahl) :
                         ganzzahl;
Variable lauf: ganzzahl;
Variable q: ganzzahl;
Variable r: ganzzahl;
Variable lae: ganzzahl;
Variable  bu: Reihe[1..8]
                    von ganzzahl;
Variable flag: ganzzahl;
Beginn
flag := 0;
lauf := 0;
falls z < 0 dann Beginn z := 0-z;
flag := 1;Ende;
falls z >= 1 dann
Beginn
  solange z >= 1   wiederhole
  Beginn
    q := qgs(z,10);
    r := mods(z,10);
    z:= q;
    lauf := lauf + 1;
    bu[lauf] := 48 + r;
    lae := lauf;
  Ende;
  falls flag = 1 dann gibAus(45);
  solange lauf >=1 wiederhole
  Beginn
    gibAus(bu[lauf]);
    lauf := lauf -1;
  Ende;
Ende
  sonst
    Beginn
     gibAus(48);  lae:=1; Ende;
gibAus(13);  //CR
liefereAlsFunktionswert(lae);
S_Fertig;
Ende;
```

```
Methode initBD: ganzzahl;
Beginn
ticks:= 0;
dreh := 0;
puls := 0;
panz := 0;
bereit := KundeA;
liefereAlsFunktionswert(0);
Ende;

Methode erhoeheTicks: ganzzahl;
Beginn
 falls ticks = 6 dann Beginn
 panz := puls;
 puls := 0;
 ticks:= 0;
                          Ende
 ticks := ticks + 1;
liefereAlsFunktionswert(0);
Ende;

Methode erhoeheUmdr: ganzzahl;
Beginn
 dreh := dreh + 1;
liefereAlsFunktionswert(0);
Ende;

Methode erhoehePuls: ganzzahl;
Beginn
  puls := puls + 1;
liefereAlsFunktionswert(0);
Ende;



     );

Variable Kunde: Prozess;
Variable bv: Betrieb;
Variable dummy: Reihe[1..4] von
ganzzahl;
Beginn
  bv := & dummy; //Platz Betr.da.
  rc := bv.initBD;
  solange wahr wiederhole Beginn
  Kunde  := bv.holeBereit;
  Aufruf := Kunde.werdeAktiv;
```

```
  falls Aufruf = 50   dann
        rc := bv.erhoeheTicks
  sonst
  falls Aufruf = 51   dann
        rc := bv.erhoehePuls
  sonst
  falls Aufruf = 52   dann
        rc := bv.erhoeheUmdr
  sonst
  falls Aufruf = 95 dann Beginn
   falls Kunde = KundeA
   dann  rc :=
   bv.setzeBereit(KundeB)
   sonst rc :=
   bv.setzeBereit(KundeA);
   Kunde := bv.holeBereit;
   rc := Kunde.setzeAufStart;
                            Ende
  sonst
  falls Aufruf = 99 dann
rc := Kunde.setzeEpaNegativ
  sonst rc :=
  Kunde.setzeInDienstModus(
              Aufruf,bv);
                           Ende;
Ende!

//Der Kunde zur Anzeige
Programm KundeB;
Typ S_Neutral = Klasse(
Methode F1(-> was: ganzzahl)
                  : ganzzahl;
Methode F2(->p1: ganzzahl)
                  : ganzzahl;
);
Variable v: S_Neutral;
Variable zahl: ganzzahl;
Variable rc: ganzzahl;
Beginn
//Zur Zeit nur Anzeige Puls
zahl := v.F1(2);
rc := v.F2(zahl);
Ende!
```

```
//Der Leerlaufprozess
Programm KundeA;
Variable lauf: ganzzahl;
Beginn
lauf := 1;
solange lauf <= 1000 wiederhole
lauf := lauf+1;
Ende!
```

7 Das Demonstrationsbeispiel

Das Demonstrationsbeispiel ist als Vernüpfung von einem Browser aufrufbar unter `http://www.wi.hs-wismar.de/~mumm/ergoapp/Ergo.html`.

Die Bedienung des Programmes ist relativ einfach. Mit dem Beginn laufen sofort die beiden Geräte Uhr und Pulsgeber an. Aber erst nach der Betätigung der Tasten Boot und Run werden die Unterbrechungen auch gezählt. Die Anzeige des Pulses erfolgt unten im großen Anzeigefenster.

Kommerzielle Computer verfügen heutzutage über eine Taktfrequenz von mehreren MHz und entsprechende Zeitgeber haben eine Frequenz von z.B. 100 Hz. Selbst wenn man nur von einer Taktfrequenz mit 100.000 Hz ausginge, käme man auf einen Quotienten aus Taktfrequenz und Frequenz der Ticks eines Zeitgebers von 1000. Bei der Simulation von Hardware mittels Software ergibt sich daraus ein Problem, weil der verwendete Simulator eine maximale Taktfrequenz von 1000 Hz hat. Wollte man die Größenverhältnisse der Hardware bei der Simulation beibehalten, müßte man für den Zeitgeber eine Frequenz von 1 Hz wählen, das hieße bei der Simulation alle Sekunde einen Tick. Die Herzfrequenz liegt im Realen bei ca. einem Hz. Bei realer Hardware liegt das Verhältnis von Frequenz des Zeitgebers zur Herzfrequenz bei einhundert. Bei einer Übertragung dieses Verhältnisses auf die Simulation ergäbe das eine Herzfrequenz von 0,01 Hz, also alle 100 Sekunden (oder 1,7 Minuten) einen Herzschlag. Bei noch kleinerer Taktfrequenz, z.B. nur 500 Hz müßte man also über 3 Minuten auf einen Herzschlag warten.

Zur Vermeidung von langen Wartezeiten für den Bediener wurde die Software auf ca. sechs Ticks und drei Herzschläge pro Minute eingestellt, wobei der fiktive Puls durch Knopfdruck erhöht oder reduziert werden kann. Die Taktfrequenz der Neumann–Maschine ist auf ca. 1000 Hz fest eingestellt.

Der KundeA ist ein Leerlaufprozess, der sich mit KundeB ablöst. KundeA kann immer durch Ticks oder andere Hardwareunterbrechungen unterbrochen werden.

Damit keine Ticks „verschluckt“ werden, muß die Unterbrechungsbehandlung für Zeitunterbrechungen zwischen zwei Ticks erledigt werden, bei unserer Simulation also innerhalb von einer Sekunde (entspricht 1000 Takten), was der hier verwendete Simulator kaum schafft. Dadurch kann es zu einer Differenz zwischen tatsächlichen und gezählten Unterbrechungen kommen.

Der KundeB, der die Anzeige der gezählten Unterbrechungen realisiert, läuft hauptsächlich im Kernmodus, bei dem keine Unterbrechungen erlaubt sind. Auch während seiner Laufzeit kann es bei unserem Simulator geschehen, daß Ticks oder andere Unterbrechungen „verschluckt“ werden. Angenommen er schafft es zwischen zwei Ticks, dann beträgt die Wahrscheinlichkeit, daß ein Herzschlag gerade in dieser Zeit auftritt, noch 0,01.

Damit man nicht zu lange auf eine Ausgabe warten muß, sollte der KundeA im Simulator nicht länger als 100 Sekunden laufen, wobei die Verarbeitung der aufgetretenen Ticks darin enthalten sein sollte.

8 Erfahrungen bei der Umsetzung

Das angestrebte Ziel der exemplarischen Offenlegung einer unterbrechungsgesteuerten Software wurde erreicht. Allerdings ergaben sich nach der Fertigstellung der Software unvorhergesehene Schwierigkeiten, die hauptsächlich auf die relativ geringe Abarbeitungsgeschwindigkeit der Java Virtual Machine zurückzuführen sind, weil der Simulator in Java geschrieben wurde. Die Auswirkungen machten sich in einer Diskrepanz zwischen tatsächlich geschehenen und registrierten Unterbrechungen bemerkar. Die simulierte Neumann–Maschine war nicht in der Lage, die Unterbrechungsbehandlung zwischen zwei Unterbechungen durchzuführen. Dadurch wurden einige Unterbrechungen „verschluckt“, d.h. nicht mitgezählt.

Der gewählte Prozeßwechsel zwischen Leerlauf und Anzeige ist vollkommen willkürlich und führt defacto zu einer Anzeige eines Pulses aus der Vergangenheit. In einer Weiterentwicklung sollen deshalb nach jeder Unterbrechung sofort der aktuelle Puls und die aktuelle Geschwindigkeit angezeigt werden, indem man einfach die Zeit seit der letzten Unterbrechung verwendet. Mit entsprechenden Umrechnungsformeln kann man damit den Puls und die Geschwindigkeit (km pro Stunde) berechnen.

Durch die erreichte Lauffähigkeit der Software sind Studierende mehr motiviert, diesen recht komplizierten Stoff zu verarbeiten, als wenn er nur theoretisch behandelt würde. Die Minimalität von Rechner und Sprache ermöglichen das Erreichen des angestrebten Zieles mit nur ca. 200 Quelltextzeilen für die Programme Ergometer, KundeA und KundeB. Bei der Verwendung von kommerzieller Hard- und Software müßte man mit einem Hundert- bis Tausendfachen von Quelltextzeilen rechnen, wodurch der Einsatz in der Lehre gefährdet wäre.

Der geschilderte Simulator zeigt noch keine Geschwindigkeit und nur einen historischen und nicht den aktuellen Puls an. In einer zweiten Variante, die unter `http://www.wi.hs-wismar.de/~mumm/ergoapp2/Ergo.html` ausprobiert werden kann, wird dieser Mangel beseitigt. Die Berechnung und Anzeige nimmt dabei der Verwalter selber vor. Zur Berechnung wird die vergangene Zeit seit der letzten Unterbrechung verwendet. Als Kunde gibt es nur den Leerlaufprozeß.

Literatur

[1] **Mumm**, Harald (2003): Entwurf und Implementierung einer objektorientierten Programmiersprache für die Paula-Virtuelle-Maschine, Hochschule Wismar, Fachbereich Wirtschaft 2003, WDP – Wismarer Diskussionspapiere Heft 8/2003, ISBN 3-910102-32-8.

[2] **Mumm**, Harald (2004): Die Wirkungsweise von Betriebssystemen am Beispiel der Tastatur-Eingabe, Hochschule Wismar, Fachbereich Wirtschaft 2004, WDP – Wismarer Diskussionspapiere Heft 2/2004, ISBN 3-910102-43-3.

[3] **Röck**, Hans (2002): Arbeitsmaterial Neumann-Maschine, Universität Rostock, Sommersemester 2002.

[4] **Röck**, Hans (2003): Arbeitsmaterial Betriebssysteme, Universität Rostock, Wintersemester 2003.

Autorenangaben

Prof. Dr. rer. nat. Harald Mumm
Fachbereich Wirtschaft
Hochschule Wismar
Philipp-Müller-Straße 14
Postfach 12 10
D – 23966 Wismar
Telefon: ++49 / (0)3841 / 753 450
Fax: ++49 / (0)3841 / 753 131
E-mail: harald.mumm@wi.hs-wismar.de

WDP - Wismarer Diskussionspapiere / Wismar Discussion Papers

Heft 01/2003	Jost W. Kramer: Fortschrittsfähigkeit gefragt: Haben die Kreditgenossenschaften als Genossenschaften eine Zukunft?
Heft 02/2003	Julia Neumann-Szyszka: Einsatzmöglichkeiten der Balanced Scorecard in mittelständischen (Fertigungs-)Unternehmen
Heft 03/2003	Melanie Pippig: Möglichkeiten und Grenzen der Messung von Kundenzufriedenheit in einem Krankenhaus
Heft 04/2003	Jost W. Kramer: Entwicklung und Perspektiven der produktivgenossenschaftlichen Unternehmensform
Heft 05/2003	Jost W. Kramer: Produktivgenossenschaften als Instrument der Arbeitsmarktpolitik. Anmerkungen zum Berliner Förderungskonzept
Heft 06/2003	Herbert Neunteufel/Gottfried Rössel/Uwe Sassenberg: Das Marketingniveau in der Kunststoffbranche Westmecklenburgs
Heft 07/2003	Uwe Lämmel: Data-Mining mittels künstlicher neuronaler Netze
Heft 08/2003	Harald Mumm: Entwurf und Implementierung einer objektorientierten Programmiersprache für die Paula-Virtuelle-Maschine
Heft 09/2003	Jost W. Kramer: Optimaler Wettbewerb – Überlegungen zur Dimensionierung von Konkurrenz
Heft 10/2003	Jost W. Kramer: The Allocation of Property Rights within Registered Co-operatives in Germany
Heft 11/2003	Dietrich Nöthens/Ulrike Mauritz: IT-Sicherheit an der Hochschule Wismar
Heft 12/2003	Stefan Wissuwa: Data Mining und XML. Modularisierung und Automatisierung von Verarbeitungsschritten
Heft 13/2003	Bodo Wiegand-Hoffmeister: Optimierung der Sozialstaatlichkeit durch Grundrechtsschutz – Analyse neuerer Tendenzen der Rechtsprechung des Bundesverfassungsgerichts zu sozialen Implikationen der Grundrechte -
Heft 14/2003	Todor Nenov Todorov: Wirtschaftswachstum und Effektivität der Industrieunternehmen beim Übergang zu einer Marktwirtschaft in Bulgarien
Heft 15/2003	Robert Schediwy: Wien – Wismar – Weltkulturerbe. Grundlagen, Probleme und Perspektiven
Heft 16/2003	Jost W. Kramer: Trends und Tendenzen der Genossenschaftsentwicklung in Deutschland
Heft 01/2004	Uwe Lämmel: Der moderne Frege
Heft 02/2004	Harald Mumm: Die Wirkungsweise von Betriebssystemen am Beispiel der Tastatur-Eingabe
Heft 03/2004	Jost W. Kramer: Der Einsatz strategischer Planung in der Kirche
Heft 04/2004	Uwe Sassenberg: Stand und Möglichkeiten zur Weiterentwick-

	lung des Technologietransfers an der Hochschule Wismar
Heft 05/2004	Thomas Gutteck: Umfrage zur Analyse der Kunden des Tourismuszentrum Mecklenburgische Ostseeküste GmbH
Heft 06/2004:	Anette Wilhelm: Probleme und Möglichkeiten zur Bestimmung der Promotioneffizienz bei konsumentengerichteten Promotions
Heft 07/2004:	Jana Otte: Personalistische Aktiengesellschaft
Heft 08/2004	Andreas Strelow: VR-Control – Einführung eines verbundeinheitlichen Gesamtbanksteuerungskonzepts in einer kleinen Kreditgenossenschaft
Heft 09/2004	Jost W. Kramer: Zur Eignung von Forschungsberichten als einem Instrument für die Messung der Forschungsaktivität
Heft 10/2004	Jost W. Kramer: Geförderte Produktivgenossenschaften als Weg aus der Arbeitslosigkeit? Das Beispiel Berlin
Heft 11/2004	Harald Mumm: Unterbrechungsgesteuerte Informationsverarbeitung