DAS SIGNALKONZEPT IN P E A R L :

Eine Methode zur systematischen, problemorientierten Behandlung von Programmausnahmesituationen

Herbert Brinkkötter
Klaus Nagel

TECHNISCHE UNIVERSITÄT BERLIN
Prozeßrechnerverbund-Zentrale

16.7.1980

ZUSAMMENFASSUNG

In der vorliegenden Arbeit werden Kriterien zur Klassifizierung von Programmausnahmesituationen entwickelt und die Mechanismen, die Programmiersprachen zur Behandlung derartiger Ausnahmen vorsehen, aufgezeigt.
Demgegenübergestellt wird das Signalkonzept in PEARL, so wie es im Rahmen des Projekts "Einführung und Anwendung der Programmiersprache PEARL in verschiedenen Anwendungsbereichen der TU-Berlin" realisiert wird /PRZ79/,/PRZ80a/,/PRZ80b/.

KLASSIFIZIERUNG VON AUSNAHMEN

"Ausnahmesituationen ('exceptions') bezeichnen Situationen, auf die eine Reaktion nicht unmittelbar in demselben Kontext, in dem die Situation erkannt wird, formuliert werden kann, sondern bei denen es wünschenswert ist, die Reaktion in einem anderen Kontext zu formulieren" /Jähnichen79, S. 21/. Demgegenüber steht die traditionelle Behandlung von Ausnahmen durch Laufzeitfehler, die in der Regel den Abbruch des Programms, bei nichtsequentiellen Programmen den Abbruch des verursachenden Prozesses bewirken. Nur in Einzelfällen ist eine Behandlung von Laufzeitfehlern im Programmtext möglich. Auch beschränkt sich die Unterstützung zumindest bei älteren Programmiersprachen auf die Erzeugung von Laufzeitfehlern für relativ triviale Ausnahmesituationen (End of File, Overflow, Bounds Violation). Für die Behandlung von problemorientierten Ausnahmen stellen diese jedoch keine Konstrukte zur Verfügung. Insbesondere sollten die Sprachen, die benutzerdefinierte Operationen auf komplexen Datenstrukturen erlauben, auch adäquate Hilfsmittel für die Behandlung von Ausnahmen, die auch über klassische Fehlersituationen hinausgehen können, zur Verfügung stellen.

Da Ausnahmen immer in Abhängigkeit von bestimmten Programmzuständen auftreten, sei es als Folge interner Zustände oder auch deren Zusammentreffen mit externen Bedingungen, könnte hierin ein erster Klassifizierungsansatz liegen.

So unterscheidet /Godenough75/ Ausnahmesituationen danach, ob die Ausgangszusicherung ('output-assertion') einer Operation nicht eingehalten werden kann ('range failure') oder die Eingaben der Operation nicht im zulässigen Wertebereich ('input-assertion') liegen ('domain-failure').

Problemorientierter aus der Sicht des Programmierers erscheint jedoch eine Klassifizierung von Ausnahmen anhand der Art einer sinnvollen Reaktion auf diese (wobei die Zuordnung einer Ausnahme zu einer Klasse immer vom gegebenen Programmkontext abhängt):

1. Ausnahmen, auf die man nur mit einem Hinweis reagieren kann, da eine Reparatur nicht (mehr) möglich ist. Die Ausnahmesituation ist aber nicht so schwerwiegend, daß die Abarbeitung des laufenden Prozesses abgebrochen werden müßte.
2. Ausnahmen, die man sinnvoll behandeln kann, wodurch die Gesamtfunktion des Programmsystems weiterhin gewährleistet ist.
3. Ausnahmen, die so schwerwiegend sind, daß darauf nur noch mit einem definierten Abbruch des laufenden Prozesses reagiert werden kann.

BEHANDLUNG VON AUSNAHMEN

Eine systematische Behandlung insbesondere von problemorientierten Ausnahmen ist in herkömmlichen Prozeß-Programmiersprachen nicht ohne Verlust an Programmstruktur sowie Speicher- und Laufzeit-Effizienz realisierbar.
So werden z.B. in Prozeß-Fortran /VDI78/ Ausnahmesituationen, die in einer Subroutine auftreten, üblicherweise durch die Rückgabe von sogenannten 'Error-Codes' nach außen bekanntgegeben.
In RTL/2 /RTL74/ können zur Ausnahmebehandlung formale Prozeduren benutzt werden, die im Ausnahmefall aufgerufen werden.

Beide Konzepte sind jedoch nicht geeignet, obige Kriterien zu erfüllen; dies gilt insbesondere bei Ausnahmen, die über klassische Fehlersituationen hinausgehen.

So ist die Behandlung über 'Error-Codes' ineffizient, da bei Erkennen der Ausnahme nicht direkt zur Behandlung übergegangen werden kann, sondern zum einen der 'Error-Code' gesetzt werden und zum anderen an jeder aufrufenden Stelle dieser explizit abgefragt werden muß.
Die Übergabe einr Fehlerbehandlungsroutine per Parametermechanismus führt aufgrund der Sichtbarkeitsregeln moderner Programmiersprachen zu Problemen bei der Ausnahmebehandlung, falls diese den Zugriff zu nicht prozedurlokalen Daten erfordert.

Notwendig erscheinen Sprachkonstrukte, die eine sichere und in den übrigen Programmtext strukturell integrierbare Formulierung von Programmteilen, die nur als Reaktion aufgrund einer Ausnahmesituation ausgeführt werden, erlauben.

Derartige Sprachelemente sind bekannt als Mechanismen zum 'Exception-Handling'.
Die Benennung von Ausnahmen ('exception definition') ermöglicht eine Zuordnung zwischen dem Auftreten der Ausnahme ('raising') und dem Programmteil, der für die Behandlung der Ausnahme vorgesehen ist ('Exception-Handler').
Benannte Ausnahmen sollen den in der jeweiligen Programmiersprache bestehenden Sichtbarkeitsregeln genügen; weiterhin wird allgemein gefordert, daß einer Ausnahme an jeder Programmstelle in ihrem Gültigkeitsbereich ein 'Handler' eindeutig zugeordnet ist /Jähnichen79/.

Die Programmiersprache ADA /ADA79/ für integrierte Rechnersysteme ('embedded computer systems') und auch neuere Systemimplementierungssprachen wie MESA /Geschke77/, ALPHARD /Levin77/ und CDL2 /Koster78/, sehen Mechanismen zur Ausnahmebehandlung vor. Diese genügen weitgehend den oben genannten Anforderungen; auf sie wird im folgenden nicht weiter eingegangen.

Die Prozeßprogrammiersprache PEARL /DIN78/,/PDV77/ stellt über das Signalkonzept ebenfalls eine in den übrigen Programmtext integrierbare Formulierung von Programmteilen (Signal-Reactions), die nur in bestimmten Situationen (Signal-Raising, Signal-Stimulation) ausgeführt werden, zur Verfügung. Daher ist auch das Signalkonzept in PEARL ein geeignetes Hilfsmittel zur Behandlung von Ausnahmen.

DAS SIGNALKONZEPT IN P E A R L

Jede Ausnahmesituation, die bei der Abarbeitung einer PEARL-Anweisung auftreten kann, ist mit einem in der Sprache vordefinierten Signal assoziiert. Zusätzlich können sogenannte freie Signale durch den Programmierer bestimmten Programmbausteinen (neudefinierte Operatoren etc.) zugeordnet werden. Während der Lebensdauer eines Signals ist diesem eine Reaktion eindeutig zugeordnet. Diese Reaktion kann eine Standardreaktion oder eine (problemorientierte, kontextabhängige) vom Programmierer zugeordnete Reaktion sein. Eine derartige Neuzuordnung ist an beliebiger Stelle im Anweisungsteil eines Programms möglich und unterliegt den in PEARL gültigen Scope-Regeln. Bei Auftreten eines Signals (Ausnahmesituation) wird die zugehörige Signalrekation ausgeführt. Die PEARL-Anweisung INDUCE erlaubt dem Programmierer das explizite Erzeugen eines Signals.

Das Problem der 'Printer-Spooling' wird hier zur näheren Erläuterung des Signalkonzepts herangezogen.
Hierbei soll einer Anzahl von Prozessen erlaubt werden, ihre Anforderungen bzgl. des Ausdruckens von Dateien in einer 'Printer-Spooling-Queue' abzulegen. Von dort werden sie durch einen 'Printer-Spool-Prozeß' nebenläufig ausgedruckt.

Das Beispiel* stellt einen Prozeß *p* dar, der durch Aufruf einer Prozedur *print* den Prozeß *spooler* beauftragt, eine Datei auszudrucken. Es werden hierbei zwei Ausnahmesituationen betrachtet:

a. die zu druckende Datei existiert nicht (*open_error*),
b. beim Drucken wird das Dateiende erreicht (*end_of_file*).

Hierbei wird das erste Signal an den Prozeß *p* hochgereicht, um dort kontextabhängig behandelt zu werden, wohingegen das Signal *end_of_file* lokal in Prozeß *spooler* zur Beendigung des Druckauftrags verwendet wird.

---

*Das von der Sprachdefinition abweichende Alphabet und die Unterstreichung der Schlüsselworte wurde aus didaktischen Gründen verwendet.

```
module (pcintec_spooling);
system;
   pcintec : lp ;
problem;
   specify (open_error,end_of_file) signal global ,
           pcintec dation out alphic dim(,) focmacd contcol(all),
   ...
   local data- and proceduce declacations
   ...
   pcint : proceduce (file_name chac(8) identical) ceent global;
      declace spool_file cef dation;
      start:
      begin
         on open_error: begin
                           call celease_spool_file(spool_file);
                           goto except;
                        end;
         spool_file :=cequest_spool_file;
         open spool_file by idf(file_name),old;
         call put_spool_file_to_buffec(spool_file);
         activate spoolec;
         cetucn;
      end;
      except:
      induce open_eccoc;
      goto stact;
   end /*pcint*/;
   spoolec : task;
      declace spool_file cef dation,
              line chac(80);
      on end_of_file: begin
                         close spool_file;
                         call celease_spool_file(spool_file);
                         tecminate;
                      end;
      spool_file := get_spool_file_fcom_buffec;
      cepeat
         get line fcom spool_file;
         put line to pcintec;
      end;
   end /*spoolec*/;
modend /*pcintec_spooling*/;

...
module;
...
problem;
   specify open_eccor signal global,
           pcint entcy (chac identical) ceent global;
   ...
   p : task;
      ...
      on open_eccoc: do samething on pcint file ...;
      ...
      call pcint(file);
      ...
   end /*p*/;
   ...
modend;
```

Die Behandlung des Signals *open_error* bedarf einer besonderen Erläuterung.
Um die Funktion des 'Printer-Spoolings' zu gewährleisten, muß sichergestellt werden, daß bei Auftreten einer Ausnahme in der Prozedur *print* die Konsistenz der prozedurlokalen Daten wiederhergestellt wird. Erst dann kann das zugehörige Signal in den aufrufenden Kontext zur Ausnahmebehandlung hochgereicht werden. Daher wird beim Auftreten des *open_error* zuerst prozedurlokal eine geeignete Anweisungsfolge ('Clean-up') abgearbeitet und anschließend das Signal erneut erzeugt. Dies erfolgt jedoch in einer Umgebung, in der dem Signal eine Reaktion aus dem Prozeß *p* zugeordnet ist.
Nach der Behandlung der Ausnahmesituation in *p* wird die Abarbeitung der Prozedur *print* fortgesetzt.

Das obige Vorgehen zeigt auf, wie die von Konstrukten zur Ausnahmebehandlung geforderte 'Clean-up'-Möglichkeit, die das PEARL-Signalkonzept per se nicht unterstützt, realisierbar ist.

Falls bei der Ausnahmebehandlung der Wert eines Parameters (hier z.B. der Eingangsparameter *file_name*) geändert werden muß, so muß dieser mit dem Parameter-Übergabe-Mechanismus 'call by reference' übergeben werden.
Wie obiges Beispiel verdeutlicht, bietet das Signalkonzept dem PEARL-Programmierer ein tragfähiges Hilfsmittel zur systematischen, problemeorientierten Behandlung von Ausnahmesituationen, das weitgehend den heutigen Anforderungen an Mechanismen zum 'Exception-Handling' genügt.

LITERATURVERZEICHNIS

/ADA79/ Preliminary ADA reference manual and rationale. Sigplan Notices 14,6 (1979).

/DIN78/ DIN 66253 Teil 1, Programmiersprache PEARL. Basic PEARL. Beuth Verlag GmbH, Berlin/Köln(1978).

/Geschke77/ Geschke, C.M., Satterthwaite, E.H.: Exception Handling in MESA. Xerox Palo Alto Research Center (1977).

/Goodenough75/ Goodenough, J.B.: Exception Handling: Issues and a Proposed Notation. CACM, Vol. 18,12(1975).

/Jähnichen79/ Jähnichen, S.: "Exception Handling"in sequentiellen Programmen; Dissertation, TU-Berlin, FB 20 (1979).

/Koster78/ Koster, C.H.A., Stahl, H.M.: Proposed changes to CDL2; Interner Bericht, TU Nijmegen (1978).

/Levin77/ Levin, R.: Program Structures for Exceptional condition Handling; PhD Thesis; Department of computer Science, Carnegie-Mellon University (1977).

/PDV77/ Full PEARL.Language Description. PDV-Berichte; KFK-PDV 130; Karlsruhe (1977).

/PRZ79/ PRZ4.1-1079: Das Signalkonzept in PEARL. Entwicklungsnotiz; Prozeßrechnerverbund-Zentrale, Projekt PEARL; TU-Berlin (1979).

/PRZ80a/ PRZ4.2-0380: Signalkonzept für HP 1000 - Implementierungshinweise. Entwicklungsnotiz; Prozeßrechnerverbund-Zentrale, Projekt PEARL; TU-Berlin (1980).

/PRZ80b/ PRZ4.3-0580: Beschreibung der vordefinierten Signale. Entwicklungsnotiz; Prozeßrechnerverbund-Zentrale, Projekt PEARL; TU-Berlin (1980).

/RTL74/ RTL/2 Language Specifications. ICI Control Research Laboratory; RTL/2 Reference: 1 (1974).

/VDI78/ VDI/VDE 3556, Prozeß-FORTRAN 75. VDI/VDE Richtlinie; Entwurf (1978).