KERNFORSCHUNGSZENTRUM

KARLSRUHE

Januar 1973 KFK 1711

Institut für Datenverarbeitung in der Technik

Ein PL/1-Unterprogrammpaket für die diskrete Eventsimulation

J. Nehmer, D. Hilse, M. Rupp

GESELLSCHAFT
FÜR
KERNFORSCHUNG M.B.H.

KARLSRUHE

GESELLSCHAFT FÜR KERNFORSCHUNG M.B.H.
KARLSRUHE

KERNFORSCHUNGSZENTRUM KARLSRUHE

November 1972 KFK 1711

Institut für Datenverarbeitung in der Technik

Ein PL/1-Unterprogrammpaket für die diskrete Eventsimulation

J. Nehmer
D. Hilse
M. Rupp

Gesellschaft für Kernforschung m.b.H., Karlsruhe

Kurzfassung

Der Aufwand für den Aufbau von Simulationsmodellen kann wesentlich reduziert werden, wenn für die notwendigen Grundoperationen der Zeitsynchronisation und Zufallszahlenerzeugung Bibliotheksunterprogramme zur Verfügung stehen.

Der Bericht beschreibt die Funktionen und Aufrufkonventionen eines Unterprogrammpaketes für die diskrete Eventsimulation in PL/1, das für die IBM-Rechenanlagen des KFZK entwickelt wurde.

Abstract

A PL/1 Subroutine Package for Discrete Event Simulation

The expense for construction simulation models may be reduced noticeable by providing a library subroutine package supporting the basic operations for time synchronization and random number generation.

In this report a subroutine package for discrete event simulation in PL/1 is described running on the IBM 360/370 configuration of the KFZK.

Inhalt:

1. Einleitung

Die Benutzung von höheren Programmiersprachen wie ALGOL, FORTRAN oder PL/1 für Aufgaben der Simulation von kontinuierlichen und diskreten Prozessen erfordert in der Regel erhebliche Eigenprogrammentwicklungen. Wesentliche Erleichterungen für den Anwender stellen Bibliotheks-Unterprogrammpakete dar, die eine Basisunterstützung bei der Generierung von Zufallszahlen, der Zeitsynchronisation von Aktivitäten in einem Simulationsmodell und der Sammlung und Auswertung von Statistiken bilden.

Die ALGOL-ähnliche Programmiersprache SIMULA (Simulation Language) verdankt diesem Unterprogrammpaket ihren Namen, obwohl sie längst als 'General Purpose'-Sprache über den begrenzten Einsatz von speziellen Simulationssprachen (z.B. GPSS) hinaus Anwendung gefunden hat. Auch die höhere Programmiersprache PL/1, die im Institut für Datenverarbeitung in der Technik der GfK zur Simulation von Betriebssoftware eingesetzt wird, bietet auf diesem wichtigen Gebiet praktisch noch keine standardmäßige Unterstützung.

Der vorliegende Bericht beschreibt ein Unterprogrammpaket für die diskrete Eventsimulation in PL/1 - einer speziellen Form der diskreten Simulation [1] - das im IDT entwickelt wurde.

2. Arbeitsweise der Eventsteuerung

Die Eventsteuerung besteht aus einer Reihe von externen PL/1-Prozeduren, die Funktionen zur Manipulation der Events und der Generierung von Zufallszahlen enthalten. Sie werden mit der MAIN-Prozedur und den übrigen Prozeduren des Simulationsmodells, für die der Benutzer verantwortlich ist, zu einem lauffähigen Programmsystem verbunden.

Im folgenden werden Aufbau und Arbeitsweise der Eventsteuerung näher erläutert.

2.1. Datenstrukturen

Grundelemente der Datenorganisation sind Events, die eine eindeutige Beschreibung aller, zu einem definierten Zeitpunkt bekannten Aktivitäten des Simulationsmodells enthalten. Sie sind potentielle

Mitglieder in zwei unterschiedlichen Datenformationen, der Eventkette und den Delayketten.

Im folgenden werden die Strukturen von Events, der Eventkette und den Delayketten angegeben, ohne jedoch auf die Mechanismen für den Transport von Events zwischen diesen Ketten einzugehen.

2.1.1. Das Event

Ein Event wird durch eine PL/1-Datenstruktur mit folgendem Inhalt dargestellt:

```
DCL  1  EVENT
        2  CHAIN_P POINTER,        /+ Verkettungspointer +/
        2  TIME BIN FIXED (31),    /+ Aktivierungszeitpunkt +/
        2  PRIO BIN FIXED (15),    /+ Priorität +/
        2  ENTRY BIN FIXED (15),   /+ Prozedurnummer +/
        2  SV_AREA POINTER;        /+ Savearea +/
```

Die Elemente haben im einzelnen folgende Bedeutung: Der Pointer CHAIN_P dient zum Verketten von Events in den bereits genannten Event- und Delayketten. TIME bezeichnet den Zeitpunkt für die Aktivierung eines Events (Ereigniseintritt), während PRIO die Priorität eines Events kennzeichnet. PRIO wird immer nur bei Zeitgleichheit mehrerer Events ausgewertet, die in der Eventkette vorhanden sind.

ENTRY kennzeichnet die Nummer einer PL/1-Prozedur mit festem Namen, die beim Ereigniseintritt aktiviert werden soll. Ihr wird beim Aufruf der Pointer SV_AREA übergeben, der von der Prozedur zur Ablage von Parametern und temporärer Variablen benutzt werden kann. Die Zuordnung einer Savearea bietet außerdem ein einfaches Mittel, um die Prozeduren auf PL/1-Ebene reentrant zu organisieren.

Die damit definierten Eventprozeduren müssen den symbolischen Namen MST_i tragen, wobei i mit 1 beginnen und in durchlaufender Reihenfolge max. 999 betragen darf. Diese Festlegung auf vorgeschriebene Namen für Eventprozeduren wurde aus technischen Grün-

den getroffen, da PL/1 den Variablentyp 'PROCEDURE' nicht kennt und die Herstellung von Beziehungen zwischen externen Prozeduren mit unbekanntem Namen relativ umständlich ist.

2.1.2. Die Eventkette

Die Eventkette wird aus Events gebildet, die über ihren Pointer CHAIN_P miteinander verkettet sind (Bild 2.1). Ein Startpointer (EV_HEAD) zeigt auf das erste Element der Kette.

In der Kette sind Events nach aufsteigenden Aktivierungszeiten (Parameter TIME) geordnet, bei Zeitgleichheit nach abfallenden Prioritäten (Parameter PRIO, kleine Zahl = große Priorität). Bei Zeit- und Prioritätsgleichheit werden Events FIFO (First In First Out) geordnet.

2.1.3. Die Delayketten

Die Delayketten stellen ein Warteschlangensystem von sequentiellen Bedienstationen (Single Server) dar. Bedienstationen können statisch oder dynamisch eingerichtet werden und mit vier unterschiedlichen Bedienungsmerkmalen versehen werden:

FIFO
LIFO
PRIORITY
RANDOM

Alle Bedienstationen mit gemeinsamem Bedienungsmerkmal werden in einer Delaykette zusammengefaßt, so daß vier voneinander unabhängige Queue-Systeme existieren (Bild 2).

Simultane Service Requests an eine Bedienstation, die von Eventprozeduren ausgehen, führen zur Ausbildung von bedienstation-spezifischen Warteschlangen, die die zugehörigen Events enthalten. Kontrollblöcke für Bedienstationen in einem der vier Queue-Systeme werden linear durch Vorwärts- und Rückwärtsverzeigerung verkoppelt. Die Warteschlangen an den Bedienstationen, die durch zyklische Vorwärtsverzeigerung von Events realisiert werden, sind in allen vier Queue-Systemen gleichartig organisiert. Dadurch

konnten die Programme für die Delayketten-Manipulation im wesentlichen identisch aufgebaut werden.

2.2. Funktionen der Eventsteuerung und Aufrufkonventionen

Während in den vorangegangenen Kapiteln ausschließlich die Datenstrukturen zur Verwaltung von Events behandelt wurden, wird im folgenden auf die Kontrollstruktur und die Funktionen der Eventmanipulation näher eingegangen.

Die anschließende Deklarationsliste gibt einen Überblick über alle Prozeduren der Eventsteuerung:

```
DCL
   EV_CTL   ENTRY (CHAR(1)),
   EV_CRT   ENTRY (BIN FIXED (31), BIN FIXED(15), BIN FIXED(15),
                   POINTER)          RETURNS (BIN FIXED(15)),
   EV_DEL   ENTRY (BIN FIXED(15), POINTER) RETURNS (BIN FIXED(15)),
   EV_DNE   RETURNS (BIN FIXED(15)),
   EV_ERSE  RETURNS (BIN FIXED(15)),
   EV_POST  ENTRY (BIN FIXED(31), BIN FIXED(15), BIN FIXED(15),
                   POINTER)  RETURNS (BIN FIXED(15),
   EV_CTBS  ENTRY (CHAR(1), POINTER) RETURNS (BIN FIXED(15),
   EV_DTBS  ENTRY (CHAR(1), POINTER) RETURNS (BIN FIXED(15),
   EV_DLAY  ENTRY (CHAR(1), POINTER, BIN FIXED(15), BIN FIXED(15),
                   POINTER, CHAR(4))  RETURNS (BIN FIXED(15)),
   EV_ACT   ENTRY (CHAR(1), POINTER)  RETURNS (BIN FIXED(15)),
   EV_TIME  RETURNS (BIN FIXED(31)),
   EV_NORM  ENTRY;
```

Bei Benutzung der Prozeduren müssen die zugehörigen Deklarationen explizit angegeben werden.

2.2.1. Eventaktivierung

```
CALL EV_CTL (MODUS)      /+ Event Control +/

DCL MODUS CHAR(1);       /+ 'O' = Optional, 'N' = Run +/
```

Beschreibung:

Nach Durchführung von system- und anwendungsspezifischen Normierungen und Initialisierungen (EV_NORM) wird durch Aufruf dieser Prozedur die Simulationsphase gestartet. Die Kontrolle wird erst am Ende des Simulationslaufes an die aufrufende Prozedur zurückgegeben.

Nach Ausgabe des Textes: 'BEGIN OF SIMULATION' fragt das System im Falle MODUS = 'O' nach dem gewünschten Modus des Simulationslaufes. Bei Angabe von 'TEST' oder 'T' wird in den Testmodus gegangen, der nach jeder Rückkehr der Kontrolle an die EV_CTL die Ausgabe aller gegenwärtig in der Eventkette notierten Events auslöst. Bei Angabe von 'RUN' oder 'R' wird in den Laufmodus gegangen, der eine Produktionsphase einleitet und alle Kontrollausdrucke unterdrückt. Der Laufmodus kann auch direkt durch MODUS = 'N' beim Aufruf der EV_CTL eingestellt werden.

Nach Beendigung der Modus-Initialisierung inspiziert EV_CTL zyklisch die Eventkette und führt dabei folgende Operationen aus:

- Ist die Eventkette leer, wird die Kontrolle an die aufrufende Prozedur (im allgemeinen die MAIN-Prozedur) zurückgegeben. Das Ende eines Simulationslaufes ist erreicht.
- Das erste Event der Kette wird entfernt und zum aktuellen erklärt.
- Die globale Simulationszeit (SIMTIME) wird nach der Zeitangabe des aktuellen Events korrigiert.
- Die im Event spezifizierte Eventprozedur wird unter Angabe des Parameters SV_AREA aktiviert: CALL MST_i (SV_AREA)

Das Wiedereinbringen des aktuellen Events in die Eventkette wird durch eine noch zu behandelnde Funktion der Eventsteuerung durchgeführt und ist Aufgabe der Eventprozeduren.

2.2.2. Manipulation der Eventkette

Im folgenden werden Aufrufkonventionen und Wirkungsweise aller Funktionen zur Manipulation von Events in der Eventkette beschrieben.

```
RT_CODE = EV_CRT (TIME, PRIO, ENTRY, SV_AREA);  /+ Create Event +/

DCL  RT_CODE BIN FIXED(15);  /+ Return-Code: O=OK, 1=Time<O +/
DCL  TIME    BIN FIXED(31);  /+ Aktivierungszeitpunkt +/
DCL  PRIO    BIN FIXED(15);  /+ Priorität +/
DCL  ENTRY   BIN FIXED(15);  /+ Prozedurnummer +/
DCL  SV_AREA    POINTER;     /+ Savearea +/
```

Beschreibung:

Ein neues Ereignis wird kreiert und unter Beachtung von TIME und PRIO in die Eventkette eingereiht. TIME ist eine Relativzeit, die sich auf den aktuellen Simulationszeitpunkt (SIMTIME) bezieht.

```
RT_CODE = EV_DEL (ENTRY, SV_AREA);  /+ Delete Event +/

DCL  RT_CODE BIN FIXED(15);  /+ Return-Code: O=OK,
                                1=nicht identifizierbares Event +/
DCL  ENTRY BIN FIXED(15);    /+ Prozedurnummer +/
DCL  SV_AREA  POINTER;       /+ Savearea +/
```

Beschreibung:

Das Event mit der Identifikation ENTRY und SV_AREA wird aus der Eventkette entfernt und der Speicherplatz freigegeben.

```
RT_CODE = EV_DNE;  /+ Delete New Element +/

DCL RT_CODE BIN FIXED(15);  /+ Return-Code: O=OK, 1=verbotener
                               Delete-Versuch +/
```

Beschreibung:

Das aktuelle Event (das sich bereits nicht mehr in der Eventkette befindet) wird gelöscht und der Speicherplatz freigegeben.

Ein verbotener DELETE-Versuch liegt vor, wenn das Event durch einen vorangegangenen Aufruf von EV_DNE bereits gelöscht oder durch die POSTPONE-Funktion wieder in die Eventkette zurückgeschleust wurde.

```
RT_CODE = EV_ERSE;  /+ Erase Event +/
```

```
DCL  RT_CODE BIN FIXED(15);  /+ Return-Code: O=OK +/
```

Beschreibung:

Mit Ausnahme des aktuellen Events (das sich nicht mehr in der Eventkette befindet) werden alle Events aus der Eventkette entfernt und der Speicherplatz freigegeben. Durch den Aufruf der Funktion wird somit die Voraussetzung für die gezielte Beendigung eines Simulationslaufes geschaffen.

```
RT_CODE = EV_POST (TIME,PRIO,ENTRY,SV_AREA);  /+ Postpone Event +/
```

```
DCL  RT_CODE BIN FIXED(15);  /+ Return-Code: O=OK,
                                1=verbotener Post-Versuch,
                                2=Time < O +/
DCL  TIME BIN FIXED(31);     /+ Aktivierungszeitpunkt +/
DCL  PRIO BIN FIXED(15);     /+ Priorität +/
DCL  ENTRY BIN FIXED(15);    /+ Prozedurnummer +/
DCL  SV_AREA  POINTER;       /+ Savearea +/
```

Beschreibung:

Das aktuelle Event wird unter Beachtung von TIME und PRIO mit den angegebenen Parametern erneut in die Eventkette eingereiht.

Ein verbotener POST-Versuch liegt vor, wenn der Bereich für die Aufnahme des aktuellen Events durch einen vorangegangenen DELETE- oder POST-Request bereits gelöscht ist. TIME ist auch hier - wie in EV_CRT - eine Relativzeit.

2.2.3. Manipulation der Delayketten

Für die Manipulation von Events in den Delayketten stehen vier Funktionen zur Verfügung:

```
RT_CODE = EV_CTBS (ORDNG, BED_P);  /+ Create Bedienstation +/

DCL  RT_CODE BIN FIXED(15);  /+ Return-Code: O=OK, 1=unbekanntes
                                Bedienungsmerkmal +/
DCL  ORDNG CHAR(1);          /+ Bedienungsmerkmal, 'L'=Lifo,
                                'F'=Fifo, 'P'=Priority,
                                'R'=Random +/
DCL  1 BED_ST BASED(BED_P)   /+ Bedienstation +/
       2 TYP CHAR(4),        /+ Bedienstation-Typ +/
       2 INDEX BIN FIXED(31);/+ Typenindex +/
DCL  BED_P POINTER;
```

Beschreibung:

Die Funktion legt in der Delaykette mit dem Bedienungsmerkmal ORDNG eine Bedienstation mit der Identifikation (TYP,INDEX) statisch an. Das Attribut 'statisch' bedeutet, daß diese Bedienstation nur durch eine entsprechende DELETE-Funktion aufgelöst werden kann.

```
RT_CODE = EV_DTBS(ORDNG,BED_P);  /+ Delete Bedienstation +/

DCL  RT_CODE BIN FIXED(15);  /+ Return-Code: O=OK,
                                1=unbekanntes Bedienungsmerkmal,
                                2=Delete auf dynamische Bedienstation,
                                3=Bedienstation belegt,
                                4=unbekannte Bedienstation +/
```

(Die Deklarationen für ORDNG und BED_P sind der Funktion EV_CTBS zu entnehmen.)

Beschreibung:

Die Funktion entfernt aus der Delaykette mit dem Bedienungsmerkmal ORDNG die Bedienstation mit der Identifikation (TYP,INDEX).

Die Bedienstation kann nur gelöscht werden, wenn sich kein Event mehr in der zugeordneten Warteschlange aufhält.

```
RT_CODE = EV_DLAY(ORDNG,BED_P,PRIO,ENTRY,SV_AREA,RESULT);
```

```
                                /+ Delay Event +/
DCL  RT_CODE BIN FIXED(15);     /+ Return-Code: O=OK,
                                   1=unbekanntes Bedienungsmerkmal +/
DCL  PRIO BIN FIXED(15);        /+ Priority +/
DCL  ENTRY BIN FIXED(15);       /+ Prozedurnummer +/
DCL  SV_AREA  POINTER;          /+ Savearea +/
DCL  RESULT CHAR(4);            /+ 'CONT'=Continue, 'STOP'=Delayed +/
```

(Die Deklarationen für ORDNG und BED_P sind der Funktion EV_CTBS zu entnehmen.)

Beschreibung:

Bei Ausführung der Funktion wird abgeprüft, ob die Bedienstation mit der Identifikation (TYP,INDEX) bereits existiert. Bei negativem Ergebnis (existiert nicht) wird die Bedienstation in der Queue mit dem Bedienungsmerkmal ORDNG angelegt und RESULT='CONT' gesetzt. Das Event verbleibt in der Eventkette. Die Bedienstation wird als 'dynamisch' gekennzeichnet.

Existiert die Bedienstation bereits, so wird RESULT='STOP' zurückgemeldet, das Event aus der Eventkette entfernt und in die Delaykette überführt.

```
RT_CODE = EV_ACT(ORDNG,BED_P);  /+ Activate Event +/
```

```
DCL  RT_CODE BIN FIXED(15);     /+ Return-Code: O=OK,
                                   1=unbekanntes Bedienungsmerkmal,
                                   2=leere statische Bedienstation,
                                   3=unbekannte Bedienstation +/
```

(Für die Deklarationen von ORDNG und BED_P vgl. EV_CTBS.)

Beschreibung:

Durch diese Funktion setzt der momentane Eigentümer die Bedienstation frei. Befinden sich weitere Events in der Queue, so wird unter Beachtung des Ordnungsprinzips der Queue das nächste zu aktivierende Event bestimmt und in die Eventkette zurückgeschleust.

Befindet sich kein weiteres Event in der Queue und trägt die Bedienstation das Merkmal 'dynamisch', dann wird sie aus dem Queue-System entfernt. 'Statische' Bedienstationen können auf diese Weise nicht entfernt werden.

2.2.4. Hilfsfunktionen

Für die Abfrage der Simulationszeit steht eine weitere Funktion zur Verfügung:

```
SIMTIME = EV_TIME;  /+ Gib Simulationszeit +/
```

```
DCL  SIMTIME BIN FIXED(31);
```

Beschreibung:

Die übergebene Simulationszeit gibt den Aktivierungszeitpunkt des aktuellen Events an.

```
CALL EV_NORM;  /+ Eventnormierung +/
```

Beschreibung:

Die Prozedur muß vor Benutzung einer der übrigen Funktionen der Eventsteuerung aufgerufen werden und führt die erforderlichen Speicherallokationen und -normierungen durch.

3. Zufallszahlengeneratoren und -transformatoren

Die Simulation stochastischer Ereignisse setzt die Erzeugung von Zufallszahlen gewünschter Verteilung voraus. Im vorliegenden System wird eine Reihe einfach verwendbarer Zufallszahlengeneratoren und -transformatoren zur Verfügung gestellt. Ihre Eigenschaften und Aufrufkonventionen werden im folgenden Abschnitt beschrieben.

3.1. Wahl des geeigneten Zufallszahlengenerators

Unter einem Zufallszahlengenerator versteht man ein Unterprogramm, das aufgrund eines determinierten Algorithmus beim Aufruf eine reelle Zahl liefert, die die Realisierung einer gleichverteilten Zufallsvariablen darstellt. Die Anzahl der auf diese Weise erzeugten Zufallszahlen ist durch die Wortlänge der digitalen Rechenanlage begrenzt. Dadurch muß eine solche Folge zwangsläufig periodisch sein. Als Periodenlänge bezeichnet man diejenige Anzahl unterschiedlicher Elemente dieser Folge, die bis zur ersten Wiederholung eines der Elemente erzeugt wurden. Während die maximale Periodenlänge von der Rechenanlage abhängt, wird die aktuelle Periodenlänge von der Wahl geeigneter mathematischer Verfahren bestimmt.

Alle bisher bekanntgewordenen Erzeugungs-Algorithmen haben den grossen Nachteil: sie arbeiten rekursiv, d.h. eine gerade erzeugte Zufallszahl ist das Ergebnis einer endlichen Operation an der zuvor generierten. Diese Abhängigkeit oder Korrelation der Zufallszahlen untereinander ist in den meisten Anwendungsfällen unerwünscht. Sie gehört neben den Momenten zu den charakteristischen Größen eines Zufallszahlengenerators bestimmter Verteilung. Die Untersuchung dieser Größen mittels statistischer Tests liefert eine Aussage über die "Güte" eines Zufallszahlengenerators.

Für die Wahl eines geeigneten Zufallszahlengenerators sollten folgende Kriterien berücksichtigt werden:

1. gute statistische Testergebnisse,
2. möglichst große Periode,
3. kurze Ausführungszeit.

Zwei mathematische Verfahren für die Erzeugung von gleichverteilten Zufallszahlen

1. einfaches multiplikatives Kongruenzverfahren [6]
2. das Lehmer-Verfahren [5,7]

wurden vier verschiedenen statistischen Tests [3,4] unterzogen. Die entsprechende Irrtumswahrscheinlichkeit betrug $\alpha = 1\%$ (Wahrscheinlichkeit dafür, daß eine Hypothese abgelehnt wurde obwohl sie zutraf). Die Testergebnisse sind in der Tabelle 1A im Anhang dargestellt.

Danach erfüllt der Lehmer-Generator die drei aufgestellten Kriterien am besten. Er besitzt nicht nur die besseren Testwerte, sondern nach [5] auch eine vierfach größere Periode und, da er in Assembler/360 geschrieben wurde, eine sehr kurze Ausführungszeit.

Aufgrund dieser günstigen Eigenschaften wurde er als Basisgenerator für die folgenden Zufallszahlentransformatoren gewählt.

3.2. Die Zufallszahlentransformatoren

Zufallszahlentransformatoren transformieren die durch einen Basisgenerator gelieferten gleichverteilten Zufallszahlen in Zufallszahlen gewünschter Verteilung.

Als Maßstab für die Wahl geeigneter Transformationsverfahren dienten die unter 3.1. spezifizierten Auswahlkriterien für die Zufallszahlengeneratoren.

Die Tabelle 2A im Anhang stellt die Ergebnisse der notwendigen statistischen Tests dar. Es wurde nicht nur die Wirkung verschiedener Anfangswerte für den in 3.1. gewählten Basisgenerator auf die Testergebnisse untersucht, sondern darüberhinaus auch ein nach dem einfachen multiplikativen Kongruenzverfahren arbeitender Basisgenerator (13579) zum Ergebnis-Vergleich herangezogen. Die Testergebnisse bestätigen noch einmal die Richtigkeit der in 3.1. getroffenen Entscheidung für den Basisgenerator nach Lehmer. Für einige Transformatoren lagen verschiedene Transformationsverfahren vor. So arbeitet beispielsweise GAUSS1 mit einer Summe von 12 gleichverteilten Zufallszahlen, während sich GAUSS2 mit fünf und anschließender Bolshev-Approximation [2] begnügt. Aufgrund besserer Testergebnisse und kürzerer Ausführungszeit wurde das Transformationsverfahren von GAUSS2 für den Transformator GAUSS gewählt.

Die Tabelle 1 im Anh. zeigt eine Zusammenstellung untersuchter Zufallszahlentransformatoren. Darin ist

1. die Verteilung,
2. der Name zugehöriger Prozedur,
3. die Attribute evtl. benötigter Parameter (p1, p2), mit

   BIN FIXED(31) (FI)
   BIN FLOAT (FL)

4. die Bedeutung der Parameter,
5. der Gültigkeitsbereich erzeugter Zufallszahlen, wobei $(a,b) \triangleq a < x < b$, $[a,b] \triangleq a \leq x \leq b$ bedeutet,
6. die beiden Erwartungswerte, der Mittelwert und die Varianz in Abhängigkeit von den Parametern,

übersichtsmäßig enthalten.

3.3. Aufrufkonventionen für die Zufallszahlentransformatoren

Die oftmals gestellte Forderung nach einer großen Zahl unabhängiger Zufallszahlentransformatoren wurde in dem vorliegenden System durch eine Einrichtungsphase für Zufallszahlentransformatoren realisiert, die ihrer eigentlichen Benutzung vorangehen muß. In ihr wurden Zufallszahlentransformatoren gewünschter Verteilung erzeugt und mit einem eindeutigen Identifikationsmerkmal versehen, unter dem sie später aufrufbar sind.

Entsprechend werden Funktionen für die Erzeugung, den Aufruf, die zwischenzeitliche Deaktivierung und das Löschen von Zufallszahlentransformatoren unterschieden.

3.3.1. Erzeugung des Zufallszahlentransformators 'name'

Durch den Aufruf der Hilfsfunktion R_CRT (Random-Create) wird dem Zufallszahlentransformator 'name' ein Speicherbereich zur Verfügung gestellt, in dem der Anfangswert des Basisgenerators und die mitübergebenen Parameter der Verteilung 'name' abgelegt werden. Mit der Zuweisung der Speicherbereichsanfangsadresse

(Pointer) an den Namen der Hilfsfunktionen ist die Einrichtungsphase des Zufallszahlentransformators 'name' beendet.

Syntax:

pointer = $R_CRT [(p1[,p2])]

Die Funktionsprozedur R_CRT ist eine Generic-Function. Der entsprechende Funktionseingang wird aufgrund der Attribute der für eine Definition der gewünschten Verteilung 'name' notwendigen Parameter (p1 bzw. p2) gefunden. Sowohl 'pointer' als auch die Parameter (p1 bzw. p2, sofern vorhanden) müssen explizit deklariert werden. Die Parameter können die Attribute

1. BIN FIXED(31) (FI)
2. BIN FLOAT (FL)

besitzen.

Die Zuordnung der Parameter und deren Attribute zu der gewünschten Verteilung 'name' ist aus der Tabelle 1 ersichtlich.

3.3.2. Aufruf des Zufallszahlentransformators 'name'

Der Aufruf der Funktionsprozedur

zz = $name (pointer)

liefert eine Zufallszahl, die die Realisierung einer Zufallsvariablen der Verteilung 'name' darstellt. Die zuvor mit R_CRT ermittelte Anfangsadresse (pointer) des 'name'-spezifischen Speicherbereiches muß als Eingabeparameter übergeben werden. Entsprechend der Art des aufzurufenden Transformators (diskreter bzw. quasistetiger) hat die Zufallszahl zz die Attribute BIN FIXED(31) bzw. BIN FLOAT. Da für die Erzeugung einer Zufallszahl unter Umständen mehrere Aufrufe des benutzertransparenten Basisgenerators erforderlich sind, ändert sich jeweils der Inhalt des ersten Speicherwortes des Speicherbereiches, entsprechend der generierten Bezugszahl des Basisgenerators.

3.3.3. Anhalten und Restart des Zufallszahlentransformators 'name'

Um die Reproduzierbarkeit einer Folge erzeugter Zufallszahlen der Verteilung 'name' zu gewährleisten, kann durch den Aufruf der Hilfsfunktion

```
int = $R_SAVE (pointer)
```

die Bezugszahl des Basisgenerators für einen späteren Restart, mit Hilfe der Funktion

```
CALL $R_RSET (pointer,int),
```

des Zufallszahlentransformators 'name' zwischengespeichert werden. Die Variable 'int' besitzt das Attribut BIN FIXED(31), während 'pointer', die Anfangsadresse des 'name'-spezifischen Speicherbereiches, vom Typ POINTER ist.

3.3.4. Löschen des Zufallszahlentransformators 'name'

Eingerichtete Zufallszahlentransformatoren können zu jedem Zeitpunkt eines Simulationslaufes durch den Aufruf der Funktion RANDOM_DELETE in der Form

```
CALL $R_DLTE (pointer)
```

gelöscht werden. Die Bedeutung von 'pointer' entspricht der unter 3.3.1.

Ein ausführliches Beispiel für die Handhabung der Hilfsfunktionen bei der Erzeugung von Zufallszahlen gewünschter Verteilung findet sich im Abschnitt 5.1.3. dieses Berichtes.

4. Dateiorganisation unter TSO

Sämtliche Funktionen der Eventsteuerung und der Zufallszahlengenerierung wurden in Form von Unterprogrammen erstellt. Sie entsprechen den Regeln des auf der IBM 360/65 zur Verfügung stehenden interaktiven Teilnehmersystems TSO und sind dort als Datensätze im Objekt-Code abgelegt. Eine Ausnahme bildet die Funktion der Eventkontrolle (EV_CTL), die im Quellcode abgespeichert ist und als Schnittstelle zu den Anwendungsprogrammen gesondert übersetzt werden muß. Alle Datensätze, die zur Handhabung des Simulationspaketes dienen, sind entsprechend ihrer Beschreibung in untergegliederte Dateien (partitioned data sets) zusammengefaßt.

Die Namen der Dateien bestehen aus dem einheitlichen Vorsatz "EVSIM" und einem ihrer Beschreibung gemäßen Nachsatz.

Die Dateiorganisation sieht folgende Untergliederung vor:

- EVSIM.PLI: Eventkontrolle im Quell-Code
- EVSIM.OBJ: Funktionen der Eventsteuerung und Zufallszahlengenerierung im Objekt-Code
- EVSIM.CLIST: Kommando-Prozeduren zur Handhabung des Simulationssystems
- EVSIM.DATA: Eingabe-Datei für den Linkage-Editor
- EVSIM.LOAD: Hilfsfunktion zur anwendungsabhängigen Ablaufsteuerung der Link-Prozedur

5. Handhabung der Eventsimulation

5.1. Forderung an Form und Aufbau der Anwendungsprogramme

Der Anwender des Simulationssystems muß beim Erstellen seiner Programme einige Regeln und Konventionen beachten, die anhand des Ablaufschemas (Bild 5.1.) erläutert werden sollen.

Der Benutzer ist sowohl für den Entwurf des Hauptprogrammes (Prozedur 'Options (Main)'), das die Modellinitialisierung enthält, als auch der modellbeschreibenden Programme verantwortlich, die im folgenden Eventprozeduren (MST) genannt werden.

5.1.1. Das Hauptprogramm

Das Hauptprogramm (Prozedur 'Options (Main)') trägt einen frei wählbaren Namen. Es muß in der aufgezeigten Reihenfolge folgende Deklarationen und Prozeduraufrufe enthalten:

```
SIM:  PROC  OPTIONS(MAIN);
DCL   EV_NORM  ENTRY,
      EV_CTL   ENTRY(CHAR(1)),
      EV_CRT   ENTRY (BIN FIXED(31), BIN FIXED, BIN FIXED, POINTER)
               RETURNS (BIN FIXED);
        .
        .
        .

      Simulationsvorbereitung
      - Modellaufbau
      - Modellnormierung
          .
          .
          .


      CALL EV_NORM;


      Eventgenerierung:
        RC1=EV_CRT(....);
        RC2=EV_CRT(....);
          .
          .
          .


      CALL EV_CTL(.);


      Auswertungsphase:
      - Auswertung
      -(Sprung zur Simulationsvorbereitung)
          .
          .
          .

END SIM;
```

5.1.2. Die Eventprozeduren

Das eigentliche Modell wird aus einer Reihe von Eventprozeduren gebildet. Wie unter Abschnitt 2. bereits ausgeführt, müssen die Eventprozeduren Namen der Form MST_i (i=1...n) tragen.

Die Eventkontrolle aktiviert jeweils die Eventprozedur, deren zugehörige Prozedurnummer im aktuellen Event angegeben ist. Die Eventprozeduren erhalten ihrerseits Anweisungen zur Eventmanipulation, die ausführlich unter Abschnitt 2. beschrieben wurden.

Dem Anwender sind beim Entwurf der Eventprozeduren mit Ausnahme der Namensgebung keine Beschränkungen auferlegt. Eine Eventprozedur hat folgenden formalen Aufbau:

```
MST_i: PROC(P);
DCL   P   POINTER;
      .
      .
      .
      .
END MST_i;
```

5.1.3. Benutzung der Zufallszahlentransformatoren

Sämtliche Deklarationen für die Zufallszahlentransformatoren befinden sich in der Datei TSØ045.EVSIM.PLI unter dem Membernamen RANDCL (Random-Declare).

Sie müssen in den Quellcode des Benutzers mit Hilfe des %INCLUDE-Statements eingebunden werden. Zusätzlich ist eine logische Verbindung zwischen dem Foreground-Job des Benutzers und dieser Datei herzustellen. (In TSO mit dem ALLOCATE-Statement, in Batch mit der DD-Karte.)

Der dazu notwendige 'file-name' gibt den DD-Namen der Datei an, auf den sich das INCLUDE-Statement bezieht. (Aufgrund von momentanen Inkompatibilitäten zwischen dem PL/1-Optimizing- und dem -(F)-Compiler sind die Deklarationen im RANDCL PL/1 (F)-Compiler-unverträglich!)

Anhand eines Beispiels soll nun die Erzeugung einer Zufallszahl der Verteilung 'GAUSS' verdeutlicht werden.

Die Tabelle 1 im Anh. (letzte Zeile) zeigt, daß für die Definition des Zufallszahlentransformators 'GAUSS' zwei Parameter p1 und p2 mit den Attributen BIN FLOAT notwendig sind.

Sie mögen MUE und SIGMA heißen und seien bei ihrer Deklaration mit entsprechenden Werten initialisiert worden.

Das TSO-Protokoll des Benutzers enthält eine ALLOCATION der Form:

```
ALLOCATION DATASET ('TSØ045.EVSIM.PLI') FILE (file_name)
```

Eine Eventprozedur (MST_17) könnte folgendes Aussehen haben:

```
MST_17: PROC(SAVE_P);
        %INCLUDE  file_name (RANDCL);
         .
         .
         .
        DCL MUE BIN FLOAT INIT(...),      /+ Parameter +/
            SIGMA BIN FLOAT INIT(...),    /+ Parameter +/
            P  POINTER,                   /+ Anfangsadresse des
                                             'GAUSS8-spezifischen
                                             Speicherblocks +/
            ZZ BIN FLOAT,                 /+ GAUSS-verteilte Zufalls-
                                             zahl +/
            INT BIN FIXED(31);            /+ Evtl. Ablage der Bezugs-
                                             zahl des Basisgenerators +/
         .
         .
         .
        P=$R_CRT(MUE,SIGMA);              /+ Erzeugen des ZZT-GAUSS +/
         .
         .
         .
        ZZ=$GAUSS(P);                     /+ Aufruf des ZZT-GAUSS +/
         .
         .
         .
        ZZ=$GAUSS(P);
         .
         .
         .
        INT=$R_SAVE(P);                   /+ Ablage der aktuellen
         .                                   Bezugszahl des Basis-
         .                                   generators +/
         .
```

```
        CALL $R_RSET(P,INT);        /+ Restart des Basisgenerators +/
          .
          .
          .
        ZZ=$GAUSS(P);
          .
          .
          .
        CALL $R_DLTE(P);            /+ Löschen des ZZT-GAUSS +/
          .
          .
          .
END MST_17;
```

5.2. Benutzung der Eventsimulation unter TSO

Zur Benutzung der Eventsimulation werden dem Anwender unter TSO drei Kommandoprozeduren zur Verfügung gestellt:

- EVLINK: ein wiederverarbeitbarer Loadmodul wird erzeugt
- EVLG: ein Loadmodul wird erzeugt und ausgeführt
- EVNORM: TSO-spezifische Allokationen werden gelöscht.

Die Prozeduren müssen vom Anwender bei Gebrauch kopiert werden. Sie sind als Datensätze unter TSO mit folgenden Namen abgelegt:

'TSØO45.EVSIM.CLIST(EVLINK)'
'TSØO45.EVSIM.CLIST(EVLG)'
'TSØO45.EVSIM.CLIST(EVNØRM)'

5.2.1. Die Kommandoprozedur EVLINK

Der Benutzer kann sich mit Hilfe der Kommandoprozedur EVLINK einen modellspezifischen Loadmodul erstellen.

Die Kommandoprozedur verlangt die Angabe folgender Keyword-Parameter:

- MSTOBJ
- MSTANZ
- MAINOBJ
- LOADM

Gebrauchsvorschriften der Parameter:

- MSTOBJ (op)

  op gibt den Namen des partitioned organisierten Datasets an, der Objekt-Moduln der Eventprozeduren enthält.

Gefordert wird:

1) alle Members von op müssen einheitlich den Namen MSTi, mit i=1...n, tragen;
2) es müssen n Members vorhanden sein;
3) keine Eventprozedur, die von der EV_CTL aktiviert werden soll, darf einen Prozedurnamen mit einer Nummer größer als n tragen.

(Kein Defaultwert!)

- MSTANZ (op)

  op gibt die Anzahl der in MSTOBJ enthaltenen Members an (op=n).

(Defaultwert = 1)

- MAINOBJ (op)

  op gibt den Namen des Datasets an, in dem der Objekt-Code des Hauptprogrammes steht.

(Kein Defaultwert!)

- LOADM (op)

  op gibt den Namen einer Load-Datei an, in die der erzeugte Loadmodul abgelegt wird.

(Defaultwert: EV(TEMPNAME))

Wirkungsweise:

Abbildung 5.2.1. zeigt das Ablaufschema der Kommandoprozedur. Das Hilfsprogramm UPDATE schreibt die angegebene Anzahl der Eventprozeduren in eine Datei. Mit Hilfe dieser Datei werden im Quellcode der EV_CTL vom Makro-Prozessor die Deklarationen und Referenzen der anzuschließenden MST-Funktionen erzeugt. Danach wird die EV_CTL übersetzt und dem Linkage-Editor übergeben.

Zweite Aufgabe des Hilfsprogramms ist es, Include-Statements für die Objekt-Moduln der angebotenen Eventprozeduren zu generieren. Dieser Datensatz dient dem Linkage-Editor ebenso als "Secondary-Input" wie die Include-Statements, die auf die Objekt-Module der Eventsteuerung und Zufallszahlengenerierung zugreifen. Zum "Primary-Input" für den Linkage-Editor zählt schließlich noch der Objekt-Modul der Hauptprozedur.

Der Linkage-Editor bindet die vorliegenden Objektmoduln zum gewünschten Loadmodul zusammen. Der erzeugte Loadmodul ist entweder sofort ausführbar oder aber vom Linkage-Editor wiederverarbeitbar. Damit können weitere modellspezifische Objektmoduln eingebunden werden, wie etwa eine externe Modellinitialisierungsprozedur.

Beispiele zum Aufruf der Prozedur:

```
exec EVLINK 'mstobj(MST.OBJ) mstanz(9) mainobj(SIM.OBJ)
             loadm(LOAD1)'
ex   EVLINK 'mstobj(''TSOxxx.yy.OBJ'') mstanz(5)
             mainobj(EVSIM.OBJ) loadm(EVLOAD(Z1))'
```

5.2.2. Die Kommandoprozedur EVLG

Die Kommandoprozedur EVLG (EV-LINK - Go) stellt eine Erweiterung der Kommandoprozedur EVLINK dar. Der Aufruf der Kommandoprozedur EVLG bewirkt, daß ein modellspezifischer Loadmodul erstellt und ausgeführt wird. Die Prozedur besitzt folgende Keyword-Parameter:

- MSTOBJ
- MSTANZ
- MAINOBJ

- LOADM
- IN
- OUT

Die Parameter MSTOBJ, MSTANZ, MAINOBJ und LOADM sind gemäß den Vorschriften im vorhergehenden Abschnitt anzuwenden.

Gebrauchsvorschriften der Parameter IN und OUT

- IN (op)

  op gibt den Namen des Datasets an, der der System-File "Sysin" zugeordnet wird.

(Defaultwert = * (Terminal))

- OUT (op)

  op gibt den Namen des Datasets an, der der System-File "Sysprint" zugeordnet wird.

(Defaultwert = * (Terminal))

Wirkungsweise:

Der Ablauf der Kommandoprozedur EVLG gleicht dem Ablauf der Kommandoprozedur EVLINK. Hinzu kommt der Aufruf des erzeugten Loadmoduls.

Der erzeugte Loadmodul ist vom Linkage-Editor weiterverarbeitbar.

Beispiele zum Aufruf der Prozedur:

```
exec EVLG 'mstobj(MST.OBJ) mstanz(11) mainobj(SIM.OBJ)
           loadm(LOAD1) in(EING.DATA)'
exec EVLG 'mstobj(''TSOxxx.yy.OBJ'') mstanz(7)
           mainobj(EVSIM.OBJ) loadm(EVLOAD(L2))
           out(AUSG.DATA)'
```

5.2.3. Die Kommandoprozedur EVNORM

Die Kommandoprozedur EVNORM dient dazu, von den Prozeduren EVLINK und EVLG erzeugte oder allokierte Datensätze zu löschen bzw. freizugeben.

Die Kommandoprozedur muß aufgerufen werden, wenn eine der Prozeduren EVLINK oder EVLG fehlerhaft beendet wurde. Der Aufruf der Kommandoprozedur bewirkt, daß die benutzten Datensätze der Eventsimulation deallokiert und temporär angelegte Hilfsdateien gelöscht werden.

Im Normalfall des fehlerfreien Ablaufs der Prozeduren EVLINK bzw. EVLG ist die Funktion der Kommandoprozedur EVNORM impliziert.

Literatur

[1] T.H. Naylor et al.
Computer Simulation Techniques
John Wiley & Sons, New York, 1966

[2] L.N. Bolshev
On Transformations of Random Variables
Theory of Probability and its Applications
Vol.4, No.2, 1959, pp. 129-141

[3] G.C. Canavos
A comparative analysis of two concepts in the
generation of uniform pseudo-random numbers
Proceedings-ACM, National Meeting, 1967, pp. 485-501

[4] S. Koller
Neue graphische Tafeln zur Beurteilung statistischer
Zahlen
Steinkopff-Verlag, 1969

[5] W. Liniger
On a Method by D.H. Lehmer for the Generation of
Pseudo Random Numbers
Numerische Mathematik 3, 1961, S. 265-270

[6] J. Moshman
Random Number Generators
In A. Ralston, H.S. Wilf:
Mathematical Methods for Digital Computers, Vol.2, 1967

[7] W.H. Payne. J.R. Rabung, T.P. Bogyo
Coding the Lehmer Pseudo Random Number Generator
Comm. of the ACM, Vol.12, No.2, Febr. 1969, pp. 85-86

Tabelle 1

Diskrete/quasistetige Zufallszahlentransformatoren

Verteilung	Name	Attribute p1	Attribute p2	Bedeutung der Parameter p1	Bedeutung der Parameter p2	Gültigkeitsbereich	Erwartungswerte Mittelwert	Erwartungswerte Varianz
Gleich-	RANDI	-	-	-	-	[1,m]	m/2	$m^2/12$ (m=2**31-2)
Gleich-	RADIS	FI	-	Obere Intervallgrenze	-	[0,p1]	p1/2	$(1+2/p1)*(p1)^2/12$
Binomial-	BINOM	FI	FL	Umfang	Einzelwahrsch.	[0,p1]	p1*p2	p1*p2*(1-p2)
Poisson-	POISS	FL	-	Mittelwert	-	[0,∞)	p1	p1
Gleich-	RANDF	-	-	-	-	(0,1)	1/2	1/12
Exponential-	EXPON	FL	-	Kehrwert des Mittelwertes	-	[0,∞)	1/p1	$1/(p1)^2$
Erlang-	ERLANG	FL	FI	Kehrwert des Mittelwertes	Ordnung	[0,∞)	1/p1	$1/((p1)^2*p2)$
Gauß-	GAUSS	FL	FL	Mittelwert	Standardabw.	(-∞,+∞)	p1	$(p2)^2$

Tabelle 1A

Test von Zufallszahlengeneratoren
(Irrtumswahrscheinlichkeit α=1%)

Verfahren		Einfaches Kongr.	Lehmer
Anfangswert		13579	9699691
Stichprobenumfang		1000	1000
Anzahl der Klassen		10	10
Test auf Gleichverteilung im Intervall (0,1)			
χ^2-Test	Freiheitsgr.	9	9
	Testwert	12.02	3.66
	Zul. Wert	21.67	21.67
Mittelwert	Berechnet	0.5057	0.5026
	Erwartet	0.5000	0.5000
	Abs. Abw.	0.0057	0.0026
	Zul. Abw.	0.0238	0.0238
Varianz	Berechnet	0.0811	0.0859
	Erwartet	0.0850	0.0850
	Abweich.	-0.0039	+0.0009
	Zul. Abw.	-0.0096, +0.0101	-0.0096, +0.0101
Test auf Unabhängigkeit erzeugter Zufallszahlen			
Runtest einf.	Testwert	0.1266	0.1265
	Zul. Wert	2.58	2.58
var. Länge	Freiheitsgr.	7	7
	Testwert	7.47	10.33
	Zul. Wert	18.48	18.48

Tabelle 2A

Test von Zufallszahlentransformatoren. Irrtumswahrsch. α=1%

Verteilung	Anfangswerte	$\chi^2(9)$-Verteilungs-Test	Mittelwert		Varianz	
			Berechnet	Abs. Abw.	Berechnet (s^2)	Abweich. ($s^2 - \sigma^2$)
GAUSS1	9699691	6.19	5.0282	0.0282	0.9708	-0.0292
	223092871	11.69	5.0145	0.0145	1.0454	+0.0454
	13579	13.46	5.0410	0.0410	0.9766	-0.0234
	Zul. Werte	21.67	5.0000	0.0816	1.0000	-0.1126, +0.1182
GAUSS2	9699691	12.02	4.9944	0.0036	1.0092	+0.0092
	223092871	11.47	5.0166	0.0166	1.0186	+0.0186
	13579	8.76	5.0670	0.0670	0.9916	-0.0084
	Zul. Werte	21.67	5.0000	0.0816	1.0000	-0.1126, +0.1182
EXPON	9699691	11.19	1.9897	0.0103	3.8934	-0.1066
	223092871	13.56	2.0159	0.0159	4.0792	+0.0792
	13579	15.66	1.9120	0.0880	3.4058	-0.5942!
	Zul. Werte	21.67	2.0000	0.1632	4.0000	-0.4503, +0.4730

Tabelle 2A

(Fortsetzung)

Verteilung	Anfangswerte	$\chi^2(9)$-Verteilungs-Test	Mittelwert		Varianz	
			Berechnet	Abs. Abw.	Berechnet (s^2)	Abweich. ($s^2 - \sigma^2$)
ERLANG	9699691	11.35	1.9959	0.0059	4.0266	+0.0266
	223092871	12.93	2.0237	0.0237	4.2458	+0.2458
	13579	15.66	1.9118	0.0882	3.4058	-0.5942!
	Zul. Werte	21.67	2.0000	0.1632	4.0000	-0.4503, +0.4730
BINOM	9699691	5.32	4.9830	0.0170	2.3870	-0.1130
	223092871	8.05	5.0590	0.0590	2.4720	-0.0280
	13579	10.38	5.0720	0.0720	2.5734	+0.0734
	Zul. Werte	21.67	5.0000	0.1290	2.5000	-0.2814, +0.2956
POISS	9699691	3.44	0.5180	0.0180	0.5002	+0.0002
	223092871	7.52	0.4910	0.0090	0.4824	-0.0176
	13579	1.81	0.5080	0.0080	0.5165	+0.0165
	Zul. Werte	21.67	0.5000	0.0410	0.5000	-0.0563, +0.0591

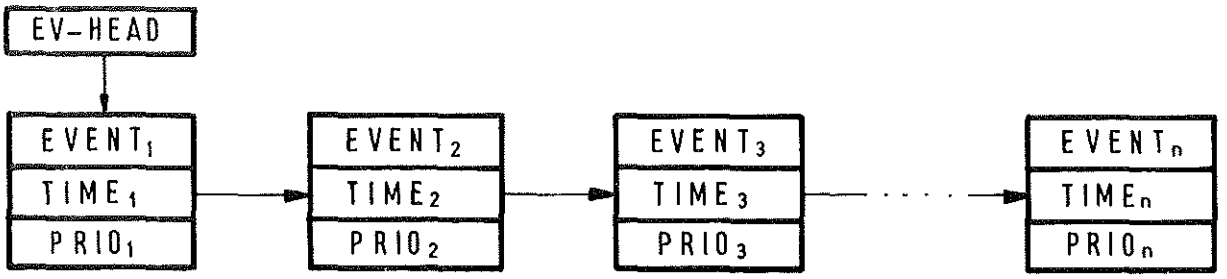

Bild 2.1
Organisation der Eventkette
($TIME_R \leq TIME_{R+1}$, $PRIO_K > PRIO_{K+1}$)

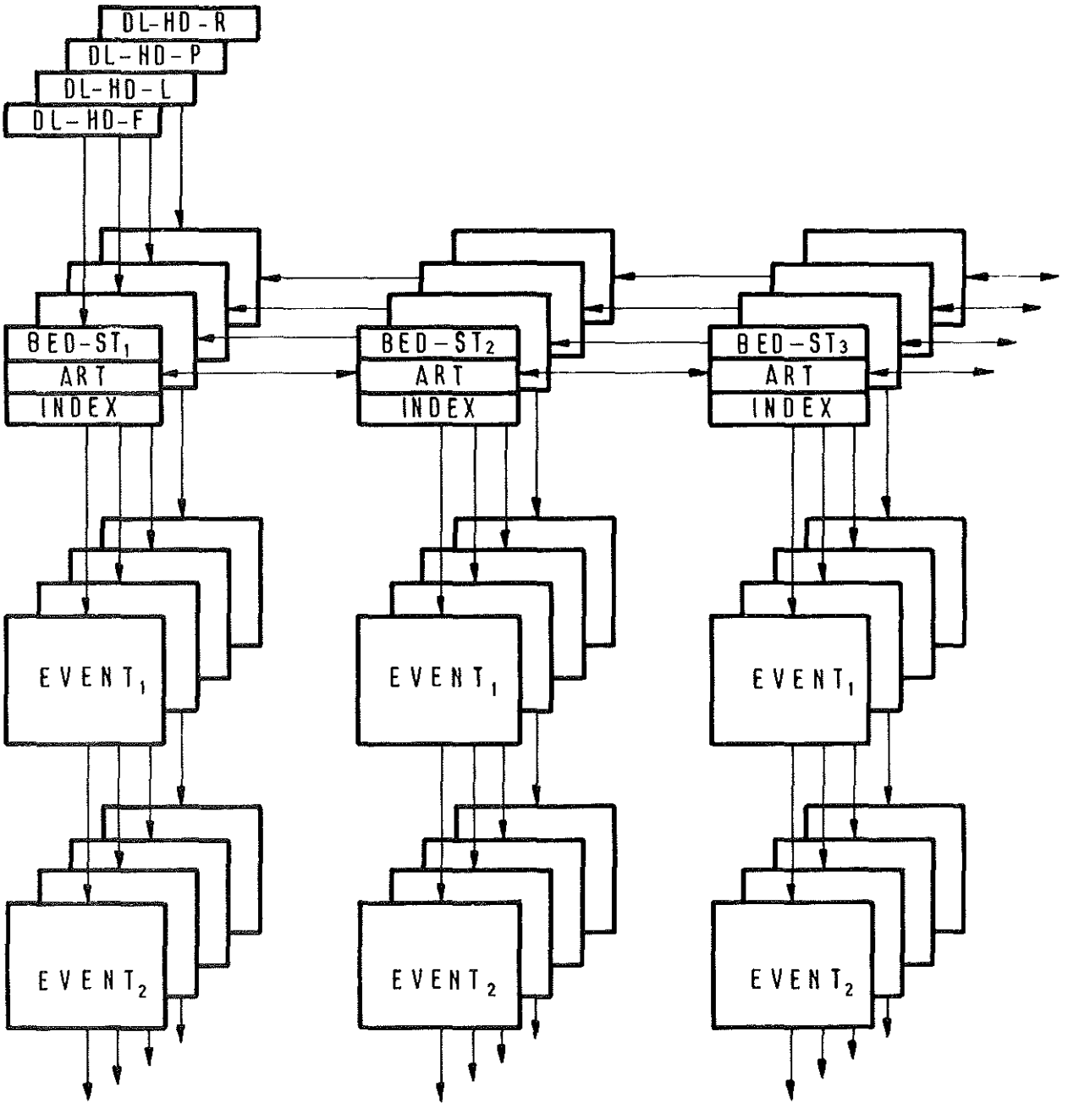

Bild 2.2
Organisation der Delayketten (LIFO, FIFO, PRIORITY, RANDOM)

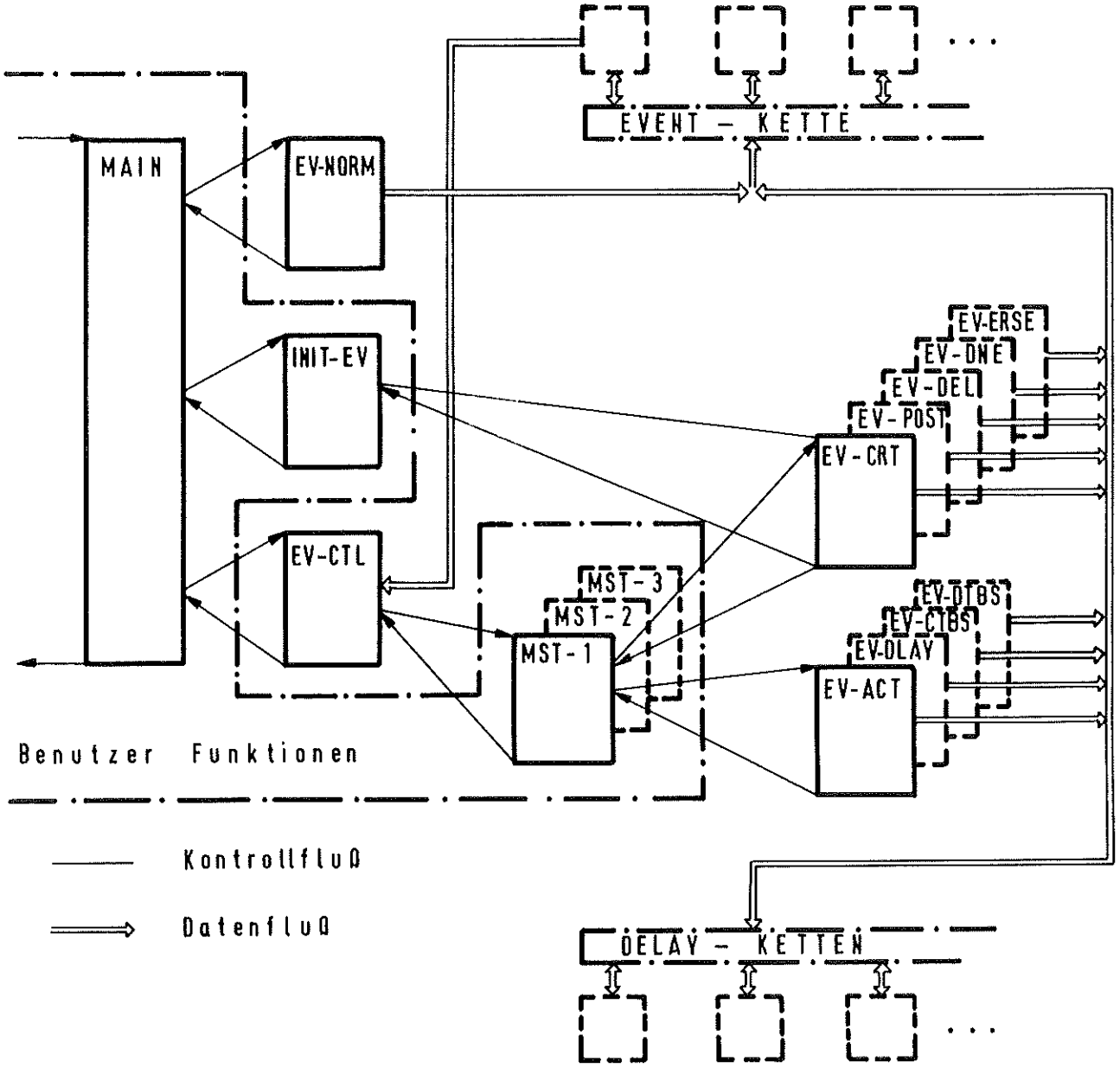

Bild 5.1

Ablaufschema der Eventsimulation

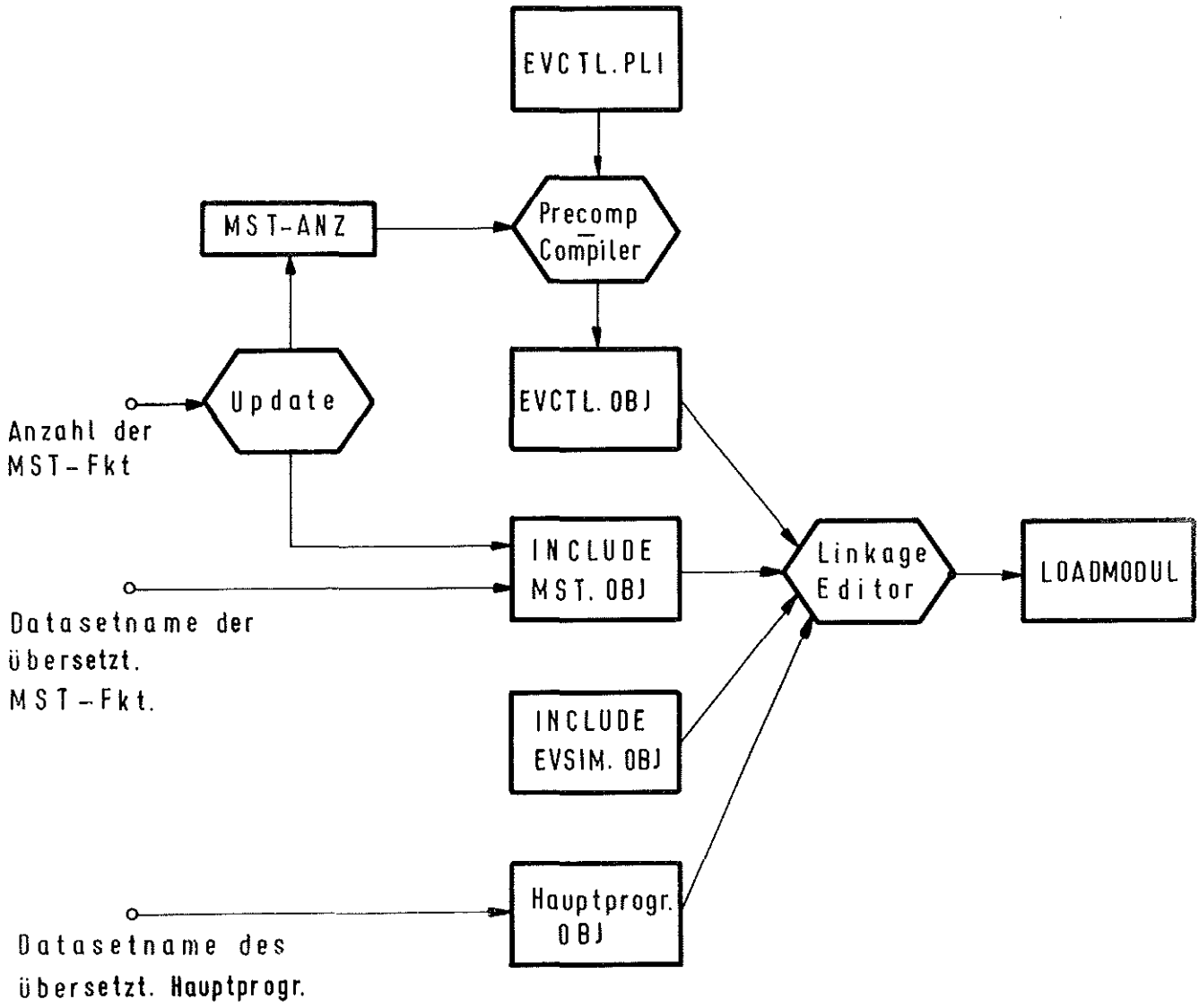

Bild 5.2.1

Ablaufschema der Kommandoprozedur

EVLINK