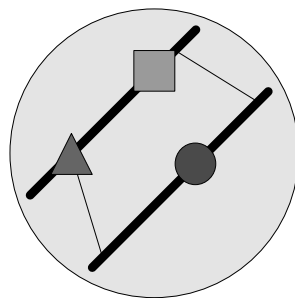


# Paralleles Modell einer Produktionsplanung und -steuerung für kleine und mittelständische Auftragsfertiger (PAMOPPS)

Anlage zum Abschlußbericht

## Referenzhandbuch zur Skriptsprache



# DREM

Markus Rehfeldt, Klaus Turowski

Institut für Wirtschaftsinformatik, Prof. Dr. Jörg Becker  
Westfälische Wilhelms-Universität Münster

Stand: 01.03.1997

## Vorwort

Dezentrale betriebliche Organisationsformen, die von Autonomie, Kooperation und indirekter Führung geprägt sind, lösen vermehrt tief gestaffelte Unternehmenshierarchien mit überwiegend bürokratischem Führungsstil ab. Eine dezentrale Organisation erfordert Anpassungen in der informationstechnischen Struktur des Unternehmens, um eine möglichst hohe Durchgängigkeit und Flexibilität bei der Gestaltung von betrieblichen Anwendungssystemen sicherzustellen. Diese Forderung legt es nahe, im Sinne einer hohen Kongruenz zwischen Organisations- und Softwaresystemgestaltung die organisatorische Dezentralisierung durch die Verteilung betrieblicher Anwendungssysteme zu unterstützen.

Ein Konzept, um - ausgehend von der Gestaltung der Organisation - die Softwaresystemgestaltung zu vereinfachen, stellt der *Planungsobjektansatz* dar. Mit der Identifikation von Planungsobjekten, die als betriebliche Aufgabe definiert sind, und Informationsobjekten, die unternehmensinterne oder -externe Daten repräsentieren, ist es möglich, die gleichen Beschreibungsmittel sowohl für die Organisationsgestaltung als auch für die Softwaresystemgestaltung zu verwenden. Planungs- und Informationsobjekte werden jeweils als eigene Softwarebausteine implementiert.

Die Implementierung eines Planungsobjekt-basierten Anwendungssystems wird durch die skalierbare Entwicklungs- und Laufzeitumgebung DREM (*Development and Runtime Environment Münster*) unterstützt, die sowohl für den Anwender als auch den Softwareentwickler Verteilungstransparenz herstellt und die parallele Ausführung von Planungsobjekten erlaubt.

Das Handbuch richtet sich an Entwickler von auf DREM basierenden betrieblichen Anwendungssystemen und beschreibt die DREM-Skriptsprache. An dieser Stelle sei jedoch darauf hingewiesen, daß es sich bei diesem Handbuch nur um einen Zwischenstand handeln kann, da das DREM stetig fortentwickelt wird.

DREM wurde im Rahmen unserer Tätigkeit am Institut für Wirtschaftsinformatik der Westfälischen Wilhelms-Universität Münster entwickelt. Großer Dank gebührt darum Herrn Prof. Dr. Jörg Becker, der uns die Entwicklung von DREM erst ermöglichte. Zu besonderem Dank sind wir unserem Kollegen, Herrn Dipl.-Wirtschaftsinformatiker Oliver Vering, unserer studentischen Hilfskraft, Herrn Helge Borchard, sowie unseren engagierten Diplomanden und Teilnehmern des Projektseminars verpflichtet, die uns bei der Entwicklung und Implementierung unterstützten. Ohne ihre Unterstützung wäre die Implementierung des DREM nicht möglich gewesen.

*Markus Rehfeldt*

*Klaus Turowski*

1	<b>Inhaltsverzeichnis</b>	
2	<b>1 Einleitung</b>	<b>1</b>
3	<b>2 Komponenten der Skriptsprache</b>	<b>2</b>
4	2.1 Allgemeines	2
5	2.2 Aufbau einer Methode	2
6	2.3 Variablendeklaration	3
7	2.4 Datentypen	3
8	2.4.1 Einfache Datentypen	3
9	2.4.2 Strukturierte Datentypen	5
10	2.4.3 Benutzerdefinierte Datentypen	6
11	2.5 Typkonvertierungsfunktionen	6
12	2.5.1 STRING	7
13	2.5.2 INTEGER	7
14	2.5.3 REAL	7
15	2.5.4 FUZZYSET	8
16	2.5.5 FUZZYNUMBER	8
17	2.5.6 FUZZIFY	8
18	2.5.7 DEFUZZIFY	8
19	2.5.8 TIMESTAMP	9
20	2.5.9 TIMESPAN	9
21	2.6 Eigenschaften	9
22	2.7 Ein- und Ausgabefunktionen	10
23	2.7.1 EDIT	10
24	2.7.2 SHOW	10
25	2.7.3 DEBUGPRINT	10
26	2.7.4 CLEAR DEBUGWINDOW	11
27	2.7.5 DEBUGALARM	11
28	2.8 Entscheidungsstrukturen	11
29	2.8.1 IF-Anweisung	11
30	2.8.2 SWITCH-Anweisung	12
31	2.9 Operatoren	13
32	2.9.1 Logische Operatoren	13
33	2.9.2 Relationale Operatoren	14
34	2.9.3 Arithmetische Operatoren	15
35	2.10 Schleifen	15
36	2.10.1 WHILE-Schleife	15
37	2.10.2 FOR-Schleife	16
38	<b>3 Befehle für einfache Datentypen</b>	<b>18</b>
39	3.1 Mathematische Funktionen	18
40	3.1.1 INDIKATOR	18
41	3.1.2 EXP	18
42	3.1.3 RNDSET	18
43	3.1.4 RND	18
44	3.1.5 NRND	19
45	3.1.6 SRND	19
46	3.1.7 ERND	19
47	3.1.8 CRND	19
48	3.2 Befehle für die Datentypen TIMESTAMP und TIMESPAN	20
49	3.2.1 CURRENT_TIME	20
50	3.2.2 DAY_OF_WEEK	20
51	3.2.3 YEAR	20
52	3.2.4 MONTH	20

1	3.2.5 DAY	21
2	3.2.6 HOUR	21
3	3.2.7 MINUTE	21
4	3.2.8 SEC	21
5	<b>3.3 Befehle für den Datentyp STRING</b>	<b>22</b>
6	3.3.1 LENGTH	22
7	3.3.2 CONCAT	22
8	3.3.3 SUBSTRING	22
9	3.3.4 LEFT	23
10	3.3.5 RIGHT	23
11	3.3.6 UPCASE	23
12	3.3.7 DOWNCASE	23
13	3.3.8 MAKE STRING	23
14	<b>4 Befehle für strukturierte Datentypen</b>	<b>25</b>
15	<b>4.1 Befehle für den Datentyp ARRAY</b>	<b>25</b>
16	4.1.1 Zugriff auf ARRAY-Elemente	25
17	4.1.2 LENGTH	25
18	4.1.3 INSERT	25
19	4.1.4 GET	26
20	4.1.5 EMPTY	26
21	<b>4.2 Befehle für den Datentyp SET</b>	<b>26</b>
22	4.2.1 Zugriff auf SET-Elemente	26
23	4.2.2 INSERT	26
24	4.2.3 GET	27
25	4.2.4 MEMBERSHIP	27
26	4.2.5 CARD	27
27	4.2.6 UNION	28
28	4.2.7 INTERSECTION	28
29	4.2.8 DIFFERENCE	28
30	4.2.9 IS_EMPTY	29
31	4.2.10 EMPTY	29
32	<b>4.3 Befehle für den Datentyp QUEUE</b>	<b>30</b>
33	4.3.1 INSERT	30
34	4.3.2 GET	30
35	4.3.3 FIRST	30
36	4.3.4 LAST	30
37	4.3.5 IS_EMPTY	31
38	4.3.6 EMPTY	31
39	<b>4.4 Befehle für den Datentyp STACK</b>	<b>31</b>
40	4.4.1 INSERT	31
41	4.4.2 GET	31
42	4.4.3 TOP	31
43	4.4.4 IS_EMPTY	32
44	4.4.5 EMPTY	32
45	<b>5 Sonstige Befehle</b>	<b>33</b>
46	5.1 CONTINUE	33
47	5.2 WAIT	33
48	5.3 BREAK	33
49	<b>6 Objekte</b>	<b>34</b>
50	6.1 Aufbau und Handhabung eines Objektes	34
51	6.2 Erzeugen und Löschen von Objekten	34
52	6.2.1 CREATE	34
53	6.2.2 DELETE OBJECT	34

1	<b>6.3 Methodenaufrufe</b>	<b>35</b>
2	6.3.1 CALL	35
3	6.3.2 LOCALCALL	35
4	6.3.3 CALLMETHOD-Variablen	36
5	<b>6.4 Selektieren existierender Objekte</b>	<b>37</b>
6	6.4.1 SELECT	37
7	6.4.2 FIRSTSELECT	38
8	<b>6.5 Synchronisation paralleler Abläufe</b>	<b>38</b>
9	6.5.1 SYNC	38
10	6.5.2 SYNCSELECT	39
11	<b>6.6 Handhabung von Sync-Nummern</b>	<b>40</b>
12	<b>6.7 Sonstige Objekt-Anweisungen</b>	<b>40</b>
13	6.7.1 ASSUME OBJECT	40
14	6.7.2 GET OBJECT	41
15	6.7.3 CHOOSE	42
16	6.7.4 MY_OID	42
17	6.7.5 CALLER_OID	42
18	6.7.6 ISVALID	42
19	6.7.7 SET_NAME	42
20	6.7.8 GET_NAME	43
21	<b>7 Fuzzy-Variablen</b>	<b>44</b>
22	<b>7.1 Befehle für den Datentyp DISCRETEFUZZYSET</b>	<b>44</b>
23	7.1.1 Zugriff auf DISCRETEFUZZYSET-Elemente	44
24	7.1.2 INSERT	45
25	7.1.3 GET	45
26	7.1.4 MEMBERSHIP	45
27	7.1.5 HEIGHT	45
28	7.1.6 GET_ALPHA_CUT	45
29	7.1.7 GET_STRICT_ALPHA_CUT	46
30	7.1.8 EMPTY	46
31	7.1.9 LENGTH	46
32	7.1.10 COMPLEMENT	46
33	7.1.11 Modifikatoren für diskrete Fuzzy-Mengen	46
34	7.1.11.1 CONCENTRATION	47
35	7.1.11.2 DILATION	47
36	7.1.11.3 CONTRAST_ENHANCE	47
37	7.1.12 COMBINE	47
38	<b>7.2 Befehle für die Datentypen FUZZYSET und FUZZYNUMBER</b>	<b>48</b>
39	7.2.1 CREATE_FUZZYSET	48
40	7.2.2 CREATE_FUZZYNUMBER	49
41	7.2.3 ALTER_FUZZYSET	49
42	7.2.4 DEGREE_OF_MEMBERSHIP	50
43	7.2.5 POSITIVE	50
44	7.2.6 NEGATIVE	51
45	7.2.7 HEIGHT	51
46	7.2.8 NORMALIZE	51
47	7.2.9 RECIPROCE	51
48	7.2.10 ISVALIDFUZZYNUMBER	52
49	7.2.11 MODIFY	52
50	7.2.12 COMBINE	52
51	<b>7.3 Fuzzy-Aggregationsoperatoren</b>	<b>53</b>
52	<b>7.4 Linguistische Variablen und Fakten</b>	<b>55</b>
53	7.4.1 FUZZYSET_OF_TERM	55
54	7.4.2 LINGVAR	56
55	7.4.3 VALUE	57
56	7.4.4 EVALUATE	57
57	<b>7.5 Befehle für den Datentyp FACTBASE</b>	<b>57</b>
58	7.5.1 INSERT FACT	57

1	7.5.2 GET_FACT	57
2	7.5.3 CLEAR	58
3	7.5.4 APPLY	58
4	<b>7.6 Regeln und unscharfes Schließen</b>	<b>58</b>
5	7.6.1 Approximatives Schließen	58
6	7.6.2 Plausibles Schließen	59
7	7.6.2.1 Fuzzy-Implikationsoperatoren	60
8	<b>7.7 Befehle für den Datentyp RULEBASE</b>	<b>61</b>
9	7.7.1 RULEBASE	61
10	7.7.2 INSERT RULE	62
11	7.7.3 DELETE RULE	63
12	7.7.4 CLEAR	63
13	7.7.5 APPLY	64
14	<b>7.8 Implementierung eines Fuzzy-Controllers</b>	<b>64</b>
15	7.8.1 Aufbau eines Fuzzy-Controllers nach E. Mamdani	64
16	7.8.2 Eine Beispielanwendung	65
17	<b>8 InOut-Manager</b>	<b>69</b>
18	8.1 Befehle für den Datentyp INOUTHANDLE	69
19	8.1.1 INOUT_OPEN	69
20	8.1.2 INOUT_READ	69
21	8.1.3 INOUT_READ_OBJECT	70
22	8.1.4 INOUT_READ_TYPEINFO	71
23	8.1.5 INOUT_READ_TYPEINFO_ALL	71
24	8.1.6 INOUT_WRITE	71
25	8.1.7 INOUT_WRITE_OBJECT	72
26	8.1.8 INOUT_WRITE_TYPEINFO	72
27	8.1.9 INOUT_SYNCABLE	73
28	8.1.10 INOUT_SYNC	74
29	8.1.11 INOUT_CLOSE	74
30	<b>9 Fehlermeldungen</b>	<b>76</b>
31	9.1 Allgemeine Laufzeit-Fehler	76
32	9.2 Laufzeit-Fehler des InOut-Managers	79
33	9.3 Laufzeit-Fehler des DDE-Managers	79
34	<b>10 Fehlerbehandlung</b>	<b>81</b>
35	10.1 Fehlerbehandlungsblock	81
36	10.1.1 TRY	81
37	10.1.2 RAISE	81
38	10.2 Werkzeuge zur Fehlersuche	82
39	10.2.1 SOURCELINE	82
40	10.2.2 SOURCEMETHOD	82
41	10.2.3 SOURCEOBJECT	82
42	<b>11 Index</b>	<b>84</b>
43		

# 1 Einleitung

Das Programm „Development and Runtime Environment Muenster“ (kurz: DREM) ist ein objektorientiertes Datenbank-System, das parallele Verarbeitung in einer verteilten Umgebung unterstützt. Das Paradigma der Objektorientierung besagt, daß Objekte Eigenschaften besitzen und ein Verhalten haben. Objekte, die gleiche Eigenschaften und gleiches Verhalten haben, werden in Klassen zusammengefaßt. Das Verhalten von Objekten einer Klasse wird in Methoden spezifiziert. Methoden können über den Typ-Browser von DREM erzeugt, verändert und gelöscht werden.

Eine Skriptsprache wird benötigt, um das Verhalten von Objekten formal auszudrücken. Bevor das neu definierte Verhalten für Objekte dieser Klasse zur Verfügung steht, muß die Methode kompiliert werden. Bei der Kompilierung wird das eingegebene Skript auf seine syntaktische Richtigkeit überprüft. Direkt im Anschluß der Kompilierung wird automatisch der für Menschen recht gut lesbare Code der Skriptsprache in einen Code übersetzt, der vom DREM-System wesentlich schneller ausgeführt werden kann.

In ihrer Syntax lehnt sich die Skriptsprache an diverse Programmiersprachen wie zum Beispiel Ada, Pascal oder C an. Die Skriptsprache ist allerdings um Elemente erweitert worden, um ihre Funktionalität an die Forderungen, die an das System gestellt werden, anzupassen. Deshalb kann man sagen, daß die zur Verfügung stehende Skriptsprache hinsichtlich ihrer Leistungsfähigkeit herkömmliche Programmiersprachen übertrifft.

In diesem Handbuch werden alle Aspekte besprochen, die für die Verwendung der Skriptsprache notwendig sind. In erster Linie handelt es sich dabei um die Syntax und die Bedeutung der verschiedenen Anweisungen. Hervorstechende Eigenschaften der Skriptsprache sind insbesondere der Umgang mit Objekten, das Verwenden von unscharfen Informationen und der Datenaustausch mit fremden Programmen. Sie werden in eigenen Kapiteln ausführlich behandelt.

## 2 Komponenten der Skriptsprache

### 2.1 Allgemeines

Zur Übersichtlichkeit und besseren Lesbarkeit dieses Handbuchs werden die reservierten Schlüsselwörter in großen Buchstaben und die Variablen-Bezeichner in kleinen Buchstaben dargestellt.

Zum Deklarieren eines Bezeichners benutzt man den Doppelpunkt (in Zeichen `:`). Wird dem Bezeichner ein Wert zugewiesen, so benutzt man einen Doppelpunkt und ein Gleichheitszeichen (in Zeichen `:=`). Jede Programmzeile, in der eine Deklaration, Zuweisung, Befehlsfolge oder ein Aufruf erfolgt, muß durch ein Semikolon (in Zeichen `;`) beendet werden. Kommentare beginnen mit zwei Gedankenstrichen (in Zeichen `--`) und enden am Zeilenende.

Bei der Definition der Syntax wird die erweiterte Backus-Naur-Form (EBNF) verwendet:

- [ a ] Die Benutzung von a ist optional.
- { a }\* a kann einmal oder beliebig oft hintereinander eingesetzt werden.
- { a }+ a kann einmal oder beliebig oft hintereinander eingesetzt werden.
- a | b Es kann a oder b benutzt werden.

### 2.2 Aufbau einer Methode

Eine Methode besteht aus zwei Teilen. Der erste Teil heißt **Deklarationsteil**. In diesem Deklarationsteil werden sogenannte „Variablen“ definiert. Diesen Variablen, die auch Bezeichner oder Attribute heißen, können Werte zugewiesen werden, um Zustände zwischenspeichern. Der Deklarationsteil wird durch das reservierte Schlüsselwort `DECLARE` eingeleitet und ist optional.

Der zweite Teil einer Methode wird als **Anweisungsteil** bezeichnet. Er dient dazu, Anweisungen zu definieren, die sequentiell abgearbeitet werden. Diese Anweisungen können dafür verwendet werden, Variablen zu manipulieren oder Methoden anderer Objekte aufzurufen. Im letzteren Fall kommunizieren die Objekte miteinander. Der Anweisungsteil wird durch die reservierten Schlüsselwörter `BEGIN` und `END` ummantelt.

Somit läßt sich der Aufbau einer Methode wie folgt darstellen:

Die allgemeine Syntax lautet:

```
[ DECLARE
  Variablendeklaration ]
BEGIN
  {Anweisung;}+
END
```

Als Beispiel soll das vielfach zitierte „Hello, world!“ Programm dienen. Die Funktionalität beschränkt sich darauf, im Deklarationsteil eine Variable *zeichenkette* vom Typ `STRING` zu deklarieren. Im Anweisungsteil wird dieser Variable der Wert „Hello, world!“ zugewiesen und mit der Anweisung `DEBUGPRINT` im Debug-Out-Fenster ausgegeben.



**Listing Methodenaufbau**

```

DECLARE
    zeichenkette : STRING;

BEGIN
    zeichenkette := "Hello, world!";
    DEBUGPRINT(zeichenkette);
END

```

## 2.3 Variablendeklaration

Eine Variable besitzt einen eindeutigen Namen und einen Typen, die im Deklarationsteil für die gesamte Ausführungszeit der Methode festgelegt werden. Mit dieser Erkenntnis läßt sich eine Variablendeklaration folgendermaßen formal ausdrücken:

Die Syntax einer Variablendeklaration lautet:

```

identifizierer : TYPESPEZIFIKATION;

```

Dabei ist es durchaus möglich, mehrere Variablen eines Typs in einer Befehlszeile zu deklarieren, indem die einzelnen Bezeichner durch Kommata abgetrennt werden.

Der Variablenname darf aus folgenden Zeichen bestehen:

- Buchstaben-Bereich: a-z, A-Z
- Zahlen-Bereich: 0-9
- Sonderzeichen: \_ ! ?

Von Bedeutung ist allerdings, daß der Name mit einem Buchstaben beginnt.

Die Datentypen, die eine Variable annehmen darf, werden im nächsten Kapitel vorgestellt.

## 2.4 Datentypen

Für die Variablendeklaration stehen vordefinierte Datentypen zur Verfügung. Dabei wird zwischen *einfachen* Typen, *strukturierten* Typen und *benutzerdefinierten* Typen unterschieden. Letztere werden über den Typ-Browser deklariert und in der Variablendeklaration als Typ verwendet.

Die Befehle, die für die einzelnen Datentypen von der Skriptsprache zur Verfügung gestellt werden, werden explizit in weiteren Kapiteln beschrieben und erläutert.

### 2.4.1 Einfache Datentypen

Ein einfacher Datentyp ist ein Datentyp, der keine Struktur enthält. Eine Variable, die von einem einfachen Typen ist, darf nur Werte annehmen, die im Wertebereich des Typen liegen. Auf diesen Wert kann direkt zugegriffen werden. Die folgende Tabelle enthält alle einfachen Datentypen, die von DREM zur Verfügung gestellt werden.

Einfacher Datentyp	Bedeutung	Beispiel
STRING	Ein String ist eine dynamische Zeichenkette, die maximal eine Länge von 32 KB annehmen darf. Es dürfen Buchstaben, Zahlen und Sonderzeichen bei der Zuweisung benutzt werden. Bei der	<pre> text : STRING; text := "Hello, world!"; </pre>