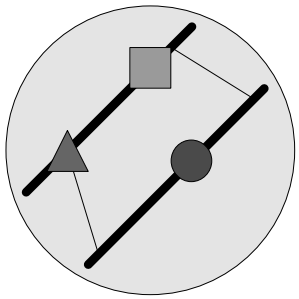
Paralleles Modell einer Produktionsplanung und -steuerung für kleine und mittelständische Auftragsfertiger (PAMOPPS)

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

Inhaltsverzeichnis

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 keinmal 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 zwischenzuspeichern. 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 : TYPSPEZIFIKATION;
```

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

Der Variablename 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 Skritpsprache 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	`text : STRING;` `text := "Hello, world!";`