Concurrently Executable Modules

Ein einheitlicher Ansatz der Systementwicklung

Dr. W.K. Epple *, H. Spandl
Universität Karlsruhe

Zusammenfassung

Dieser Beitrag stellt einen einheitlichen Ansatz zur Software Entwicklung für verteilbare Systeme vor. Dabei wird versucht, die Konzepte der Datenabstraktion zu erweitern und für die Spezifikation, den Entwurf und die Implementierung von verteilbaren Systemen nutzbar zu machen.

Zunächst wird das sogenannte Π – **Paradigm** (sprich: Pi – Paradigm) vorgestellt. Dieses besteht aus einer **Methode**, einer **Sprachfamille** mit den darauf abgestimmten **Werkzeugen** sowie einem gemeinsamen (Modellierungs–) **Konzept**: den *C*oncurrently *E*xecutable *M*odules, kurz *CEMs* genannt. Zum Schluß wird anhand eines kleinen Beispiels die Anwendung der Methode und des CEM Konzeptes aufgezeigt. Dieses Beispiel veranschaulicht, wie die Techniken der Datenabstraktion erfolgreich zur Entwicklung von verteilten Systemen eingesetzt werden können.

Schlüsselwörter: Datenabstraktion, Software Engineering, Modulares Programmieren, Parallelität, Verteilbare Systeme, Verteilte Systeme

Abstract

This paper presents the concept of a uniform approach to software development of distributed systems. The aim is to extend and apply what has been learned about the power of data abstraction to the specification, design and implementation of distributed systems.

First the socalled Π – **Paradigm** (pronounce: P – Paradigm) will be described. This paradigm consists of a **method**, a set of adopted **languages** and **tools**, and a common (modeling) **concept**: a Concurrently Executable Module (CEM). The paper finally shows the application of the method and the CEM concept to an example. This demonstrates how data abstraction techniques can be successfully used for the development of concurrent systems.

Key Words: data abstraction, software engineering, modular programming, concurrency, distributable systems, distributed systems

1. Einleitung

Über die letzten Jahre konnte ein ansteigender Bedarf für verteilte (Rechner–, Software–) Systeme beobachtet werden. Dieser Bedarf wurzelt in der Tatsache, daß der Mensch oftmals nicht mehr in der Lage ist, mit der zunehmenden Komplexität der heutigen Systeme Schritt zu halten. Hinzu kommt, daß in einer technischen Umgebung ein hohes Maß an

- Sicherheit
- Zuverlässigkeit und
- Flexibilität

gefordert wird. Zudem erfordert der Mangel an ausgebildetem und erfahrenem Personal in der Softwareindustrie die Entwicklung von wiederverwendbaren Software Komponeneten.

Aufgrund dieser Anforderungen entstand ein großer Forschungsschwerpunkt um den Software – Entwicklungsprozeß zu unterstützen und zu standarisieren. Das Ziel all dieser Forschungsaktivitäten ist es, das Produkt Software in kürzerer Zeit und mit höherer Qualität zur Verfügung zu stellen. Obwohl heutzutage bereits eine Vielzahl von Werkzeuge für die Entwicklung von kommerzieller Datenverarbeitungssoftware existiert, insbesondere für die Design–, die Implementierungs– und die Testphase, besteht immer noch ein großer Bedarf an Methoden und Werkzeugen, die alle Phasen der Entwicklung von integrierten und verteilbaren Computersystemen abdecken. In [11] wird berichtet, daß je weiter die Phase, die ein Werkzeug unterstützt, von der Kodierungs– und Testphase entfernt ist, desto unwahrscheinlicher wird es, daß es überhaupt Anwendung findet. Für die Phase der Anforderungsdefinition wird in der Praxis gegenwärtig quasi kein Hilfsmittel verwendet.

Dieser Beitrag stellt einen einheitlichen Ansatz der Softwareeinwicklung für verteilbare Systeme vor, indem er versucht, die Konzepte der Datenabstraktion zu erweitern und für die Spezifikation, den Entwurf und die Implementierung von verteilten Systemen nutzbar zu machen.

Grundlage ist das sogenannte Π – **Paradigm**, bestehend aus einer **Methode**, einer **Sprachfamille** und dazugehörigen **Werkzeugen** sowie einem gemeinsamen (Modellierungs–) **Konzept**: Gleichzeitig Ausführbaren Modulen (Concurrently Executable Modules, kurz *CEMs* genannt). Der Name Π – Paradigm steht für Peacock – Paradigm, wobei Peacock wiederum der Name des ESPRIT Forschungsprojektes ist, in dessen Rahmen der hier vorgestellte Ansatz näher erforscht wird.

2. Das Π – Paradigm

Ein heutzutage oft beschrittener Weg der Systementwicklung zerlegt ein System in einem ersten Schritt in disjunkte Einheiten, die eine bestimmte Funktion ausführen. In einem zweiten Schritt werden die Informationen identifiziert, die zwischen den

Einheiten ausgetauscht werden. Man bezeichnet diese Vorgehensweise als funktionalen Ansatz. Diese Technik wird weitgehend von den traditionellen Programmiersprachen unterstützt, die Konzepte wie Prozeduren oder parallele Prozesse zur Verfügung stellen. Der von uns verfolgte Ansatz stellt dagegen die Verwaltung einer Dateneinheit und die auf den Dateneinheiten vorgenommenen Operationen in den Vordergrund. Das allgemeine Ziel ist es, die Konzepte der Datenabstraktion auf die Spezifikation, den Entwurf und auf die Implementierung verteilter Systeme anzuwenden.

Um ein verteiltes System erfolgreich entwickeln zu können, genügt es nicht, lediglich eine gute Programmiersprache zur Verfügung zu stellen. Selbst das Vorhandensein einer kompletten Entwicklungsumgebung, die eine Spezifikationssprache, eine Implementierungssprache und Werkzeuge zur Manipulation oder Analyse einer Systembeschreibung enthält, genügt nicht, wenn nicht ein Paradigma der Systementwicklung zugrundeliegt.

Dieses Paradigma muß Antworten auf folgende Fragen bereithalten (siehe auch Bild 1):

- Wie sieht ein System auf den unterschiedlichen Abstraktionsebenen aus?
- Wie kann der Systementwickler ein System auf den verschiedenen Abstraktionsebenen beschreiben?
- Welche Hilfen werden dem Systementwickler geboten, damit er sich zwischen den verschiedenen Ebenen bewegen kann, um so das System aufzubauen oder zu verändern?

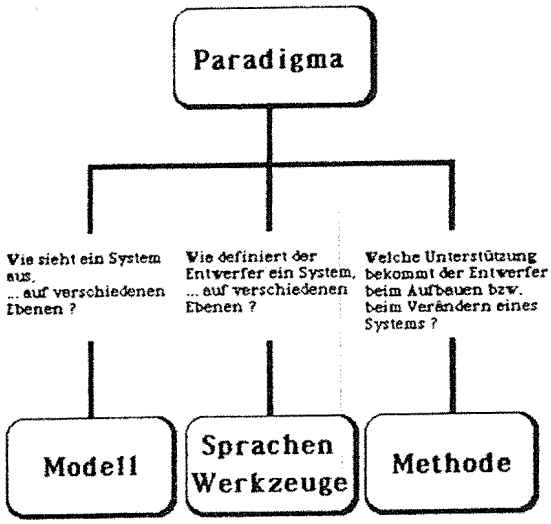

Bild 1: Bestandteile eines Paradigmas

Zur Beantwortung der ersten Frage wird ein **konzeptuelles Modell** der Struktur und Organisation eines Systems zur Verfügung gestellt, das definiert, mit welchen Komponenten der Entwickler arbeiten kann und wie diese Einheiten miteinander verknüpft bzw. wie sie erzeugt werden können. Das Modell umfaßt dabei alle Abstraktionsebenen der Systementwicklung, von der niedrigsten Stufe des lauffähigen Systems bis zur höchsten Spezifikationsstufe und in die Ebene der Anforderungsdefinition hinein. Die elementaren Komponenten des hier vorgestellten Ansatzes sind die gleichzeitig ausführbaren Module: CEMs.

Sprachen erlauben die Beschreibung von CEMs auf den verschiedenen Ebenen des Systementwurfs wie Anforderungserfassung, Design und Implementierung, wobei der Übergang von einer Beschreibung in die nächste von Werkzeugen zu unterstützen ist.

Die **Methode** basiert auf der Datenabstraktion [3], bei der die Details der Repräsentation einer Datenstruktur und die Details der auf dieser Datenstruktur operierenden Funktionen "versteckt" werden und damit beim Entwurf zunächst nicht berücksichtigt werden. Zunächst werden mögliche Datenmanipulationen nur benannt.

Ein gutes **Paradigma** ist charakterisiert durch seine Klarheit –es existiert nur **ein** konzeptuelles Modell, **eine** Entwicklungsmethode und **eine** Sprachfamilie. Damit ein Paradigma auch effektiv angewandt werden kann, muß es selbstverständlich durch entsprechende (rechnergestützte) Werkzeuge unterstützt werden.

3. Concurrently Executable Modules

Das dem Π – Paradigm zugrundeliegende Modell entstammt einer Synthese der Konzepte aus den Gebieten der Datenabstraktion und dem der parallelen Prozesse. Daraus resultiert eine einzige Einheit modularer Struktur: Gleichzeitig Ausführbare Module, oder Concurrently Executable Modules, kurz CEMs genannt.

Der Name spiegelt die Tatsache wieder, daß auf der untersten Ebene einer Systemdefinition ein System aus Modulen in dem Sinn besteht, daß jedes Modul eine Einheit der Datenabstraktion darstellt. Jede dieser Einheiten unterstützt die konkurrierende Ausführung der exportierten Operationen wo immer dies möglich ist. Dabei ist zu beachten, daß die **lokale Parallelität** innerhalb eines Moduls aus der Sicht untergeordneter Module zu einer **globalen Parallelität** wird. Auf der untersten Definitionsebene besteht ein System dann aus kommunizierenden sequentiellen Prozessen [4].

Die höheren Ebenen einer Systemdefinition abstrahieren von den verschiedenen Aspekten eines lauffähigen Systems. So hat zum Beispiel jedes Modul einen assoziierten Datentyp, da es ja eine Einheit der Datenabstraktion darstellt. Dies geschieht auf einer Ebene der Systemdefinition, auf der alle mit Speicher und Paralletität zusammenhängenden Details weggelassen werden. Andere Ebenen der Systemdefinition erhält man, indem lediglich von der möglichen Parallelität oder dem erforderlichen Speicherplatz eines lauffähigen Systems abstrahiert wird.

Der Systementwickler kann mit Hilfe von CEMs das System auf verschiedenen Abstraktionsebenen repräsentieren. Darüberhinaus kann die Darstellung eines Systems bei Verwendung von CEMs in den beiden zusätzlichen Dimensionen

- Formalität einer Beschreibung
  (So kann zum Beispiel eine erste Spezifikation lediglich die Systemkomponenten aufzählen, während ihre semantische Bedeutung informell dargestellt wird.)
- Vollständigkeit einer Beschreibung
  (In dem Sinne, daß auch eine unvollständige Spezifikation möglich ist und als von Wert betrachtet wird.)

beeinflußt werden. CEMs erlauben die Darstellung eines Systems in jeder "Ecke" des durch die drei Dimensionen

- Abstraktionsgrad,
- Formalitätsgrad und
- Vollständigkeitsgrad

aufgespannten Raumes (Bild 2).

Ein lauffähiges Programm hätte z.B. die "Koordinaten" 100% vollständig, 100% formal und 0% abstrakt. Eine Darstellung abstrakter Datentypen wie man sie gewöhnlich in der einschlägigen Literatur findet, hätte die Merkmale 50–70% vollständig, 100% formal und 100% abstrakt.

Durch das hier vorgestellte Modell wird der Systementwerfer gezwungen, ein System als eine aus CEMs gebildete Struktur zu betrachten. CEMs sind die Grundlage des Modells, sie sind die einzige Einheit zur modularen Stukturierung in der Sprachfamilie. Die Methode der Systementwicklung beinhaltet die Definition bzw. die Veränderung von Definitionen von CEM – Strukturen entlang der drei Freiheitsgrade.

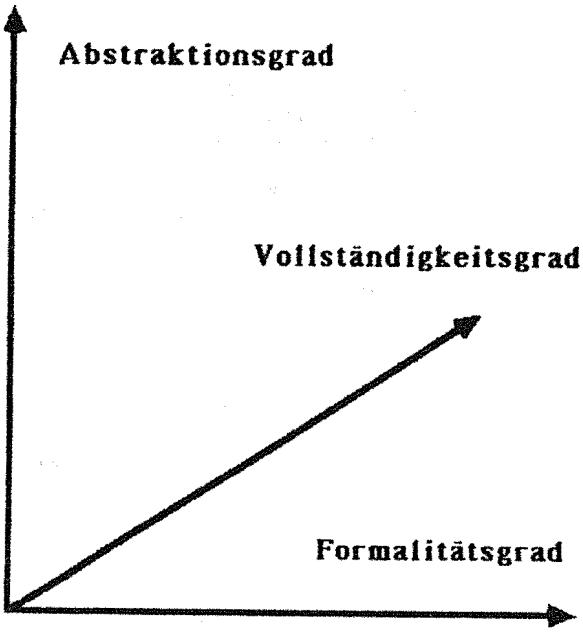

Bild 2: Freiheitsgrade des Entwurfs

4. Ein kleines Beispiel

Das vorgestellte Konzept wird anhand des Beispiels "Liftsystem" etwas konkretisiert.

4.1 Problemstellung

Gegeben sei ein Gebäude mit m Stockwerken und n Aufzügen. Jeder Lift besitzt innen für jedes Stockwerk einen Leuchttaster, der aufleuchtet, sobald er betätigt wird. Des weiteren existiert ein Alarmknopf, mit dessen Hilfe dem Hausmeister ein Notfall angezeigt werden kann. Auf jedem Stockwerk gibt es Tasten, mit denen man einen Mitnahmewunsch für eine Aufwärts– bzw. eine Abwärtsfahrt signalisieren kann. Auch hier leuchtet der Taster auf, sobald er betätigt wird. Wird ein bestimmter Fahrauftrag erledigt, so werden die entsprechenden beleuchteten Tasten wieder zurückgesetzt (Bild 3).

Es ist nun ein Steuerprogramm zu entwickeln, bei dem sichergestellt ist, daß alle Aufträge innerhalb einer endlichen Zeitspanne erledigt werden.

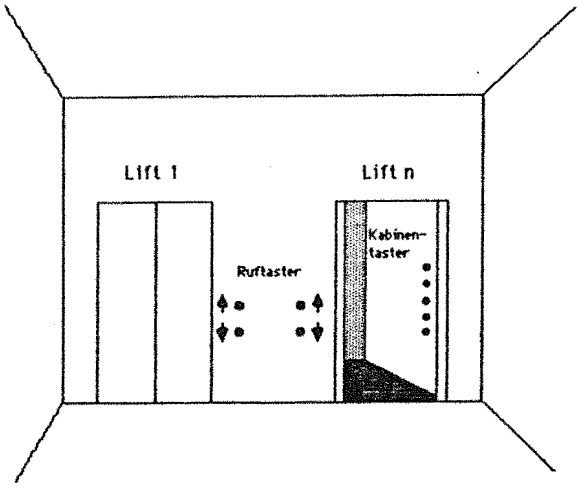

Bild 3: Das Liftsystem

4.2 Skizze einer Problemlösung

4.2.1 Systementwurf

Das Liftsystem besteht aus zwei Teileinheiten, den *Aufzügen* selbst und den *Stockwerken*. In dem System können mehrere Lifts gleichzeitig existieren, die Stockwerke sind jedoch allen Lifts gemeinsam.

Die gewählte Zerlegung spiegelt diese Struktur wieder, indem folgende Module (CEMs) zur Verfügung gestellt werden (siehe auch Bild 4):

- ein Modul zur Modellierung der Stockwerke (Floor–CEM). Die Hauptaufgabe ist die Verwaltung der Mitnahmeanforderungen.
- ein Modul zur Modellierung eines Aufzugs (Lift–CEM), welches wiederum aus folgenden Teilmodulen besteht:
  - einem Modul zur Verwaltung der innerhalb der Kabine getätigten Aufträge.
  - einem Kontrollmodul, das entscheidet, welche Aktion(en) auszuführen sind.
  - einem Modul, welches den Lift real bewegt.

Der Stockwerke CEM ist prinzipiell eine passive Einheit, die Anforderungen entgegennimmt und Informationen über noch zu erledigende Aufträge an die einzelnen Aufzüge weitergibt.

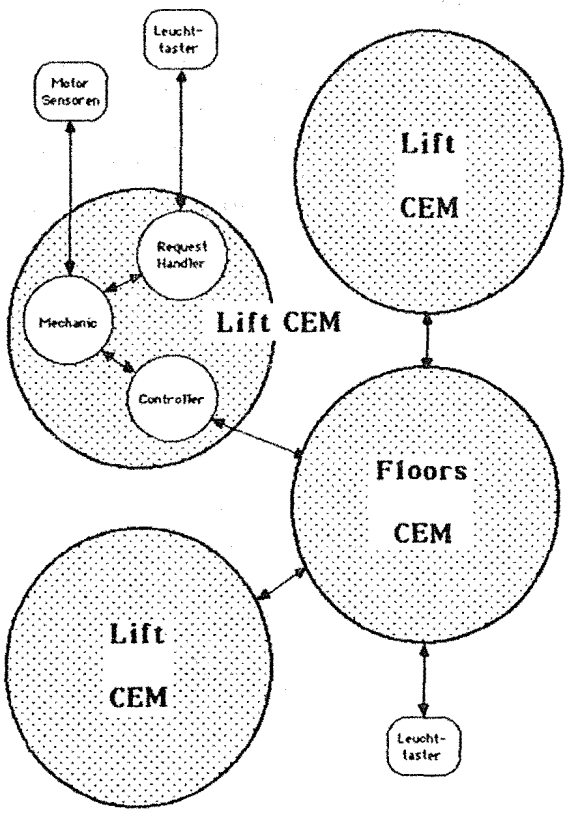

Bild 4: Struktur des Liftsteuerungs Systems

Jeder Lift–CEM kontrolliert seine Bewegung selbst. Die verwendete, relativ einfache Strategie versucht, möglichst in der momentanen Bewegungsrichtung weiterzufahren. Wenn ein Stockwerk erreicht wurde, wird überprüft, ob irgendwelche Aufträge vorliegen, die diese Etage betreffen. Ist dies der Fall, so werden sie bedient. (Dabei wird vorausgesetzt, daß die Überprüfung in Realzeit ausführbar ist.) Falls noch unerledigte Aufträge in der aktuellen Bewegungsrichtung vorliegen, so wird diese beibehalten. Im anderen Fall wird versucht, die Richtung umzudrehen, indem überprüft wird, ob Aufträge für die Gegenrichtung vorliegen. Wenn ja, so setzt sich der Aufzug dorthin in Bewegung. Sollte jedoch kein Auftrag vorliegen, so verbleibt der Lift auf dem aktuellen Stockwerk und "pollt" solange die Etagen über bzw. unter sich (inklusive dem momentanen Flur), bis ein neuer Fahrwunsch signalisiert wird.

Die hier vorgestellte Strategie ist sicherlich nicht die optimalste, sie erfüllt aber die Anforderungen und ist zur Veranschaulichung des CEM Konzeptes ausreichend.

4.2.2 Design

Als nächstes werden die abstrakten Datentypen definiert. Dazu wird hier exemplarisch der Lift–CEM behandelt. Dieser besteht aus mehreren Untermodulen, die alle getrennte Aufgabenbereiche wahrnehmen. Die verwendete Schreibweise lehnt sich an die Programmiersprache Pascal an.

Ein Lift setzt sich aus folgenden Teilen zusammen:

```
MODULE request_handler;
Sorts:      floors
Operations:
   request_floor                  : floor
   any_request_for_current_floor  :       ⇒ BOOLEAN
   any_request_above              : floor ⇒ BOOLEAN
   any_request_below              : floor ⇒ BOOLEAN
Semantics:      ...
```

Der Request_handler ist zuständig für Fahrwünsche, die innerhalb der Aufzugskabine getätigt werden. Er hat auch dafür zu sorgen, daß die entsprechenden Schalter beleuchtet werden. Die einzelnen Funktionen haben folgende Aufgabe:

- request_floor
  Nimmt eine Anforderung für Etage <floor> entgegen.
- any_request_for_current_floor
  Gibt an, ob das aktuelle Stockwerk besucht werden muß.
- any_request_above und
  any_request_below
  Wahr, falls der Lift noch Aufträge über bzw. unter der Etage <floor> erledigen muß. Die Sonderfälle oberstes bzw. unterstes Stockwerk müssen dabei natürlich entsprechend berücksichtigt werden.

Ein wichtiger Teil ist auch der "controller". Er ist ein aktiver Modul (CEM), der die im vorherigen Kapitel beschriebene Bewegungsstrategie verwirklicht. Hinzu kommt ferner der "mechanic"–CEM, der für die Motorsteuerung des Lifts zuständig ist und der außerdem die verschiedenen Sensoren, Taster etc des Aufzugs überwacht.

4.2.3 Eine mögliche Implementierung

Das CEM Konzept stellt eine Reihe von Anforderungen an die potentielle Implementierungsprache. Im einzelnen muß es möglich sein

- Datentypen zu definieren,
- Objekte eines bestimmten Typs zu generieren,
- die erlaubte Parallelität zwischen den Operationen eines Objekts zu definieren,
- parallele Objekte zu verknüpfen und ihre Kommunikation zu beschreiben,
- die (mögliche) Verteilung der Module auf verschiedene Knoten zu beschreiben, sowie
- (parallele) Objekte dynamisch zur Laufzeit zu erzeugen bzw. zu vernichten.

Eine optimal auf das CEM Konzept abgestimmte Implementierungssprache existiert zur Zeit noch nicht. Unter den existierenden Sprachen gibt es jedoch einige, die Teile der oben aufgeführten Anforderungen erfüllen und so gegebenenfalls als Zielsprache doch in Frage kämen. Es sind dies unter anderen (in alphabetischer Reihenfolge):

- Ada [1]
- Conic [6]
- Modula–2 [10]
- Nil [7]
- Occam [5]
- Pascal–Plus [8]

- Pearl [9]
- Smalltalk [2]

In einer (Pearl ähnlichen) Pseudo Programmiersprache könnte die Aufzugsteuerung folgendermaßen implementiert werden:

```
PROGRAM lift_system;
:
:
CEM floors;
   MONITOR CEM request_handler;
   PROCESS CEM io_handler;
     :
   BEGIN
      (* Initialisierungscode ... *)
      ACTIVATE io_handler;
   END;
CEM lift;
   MONITOR CEM request_handler;
   MONITOR CEM mechanic;
   PROCESS CEM controller;
     :
   BEGIN
      (* Initialisierungscode ... *)
      ACTIVATE controller;
   END;
:
:
INSTANCE lifts : ARRAY [1..n] of lift];
         house : floors;
BEGIN
   (* ''Hauptprogramm'' *)
END
```

Dabei wird hier davon ausgegangen, daß zu jeder CEM Beschreibung ein Anweisungsblock definiert ist, der bei der Generierung eines Objekts des entsprechenden Type durchlaufen wird. Dieser Block kann der Initialisierung dienen, oder auch z.B. im Falle der Prozeß CEMs die Prozeßanweisungen in Form einer Endlosschleife enthalten.

Neben der oben aufgeführten Implementierungsskizze sind sicherlich noch andere Lösungen vorstellbar, man denke nur an Objekte, die über Nachrichtenaustausch kommunizieren und nicht über gemeinsamen Speicher wie es beim Monitoransatz der Fall ist. Auch wurde nichts über eine mögliche Verteilung der CEMs ausgesagt (wie z.B., ein Lift–CEM residiert in einem lokalen Steuerrechner im jeweiligen Aufzug etc). Zur Illustration sollte dies jedoch genügen.

5. Zusammenfassung

Der Beitrag zeigt, wie unter Verwendung des einheitlichen Ansatzes des II – Paradigms auf der Basis von Datenabstraktion verteilbare Systeme entwickelt werden können. Es ist zu erkennen, daß diese Vorgehensweise weitgehend unabhängig von der endgültigen Realisierungssprache angewandt werden kann. Dazu muß lediglich gewährleistet sein, daß die Implementierungsprache eine Reihe von Mindestanforderungen erfüllt.

Der Vorteil der hier vorgeschlagenen Methode liegt darin, daß der Systementwerfer ein System in allen Phasen der Entwicklung als eine aus CEMs gebildete Struktur sieht. CEMs sind die Grundlage des Modells und sie sind die einzige Einheit zur modularen Strukturierung in der Sprachfamilie. Die Entwicklungsmethode beinhaltet lediglich die Definition bzw. die Veränderung von Definitionen von CEMs entlang der drei Freiheitsgrade: Abstraktionsgrad, Formalitätsgrad und Vollständigkeitsgrad.

Anerkennung

Der hier vorgestellte Ansatz entstand im Rahmen des Projektes *PEACOCK*, das von der Europäischen Gemeinschaft im Rahmen des ESPRIT Programms gefördert wird. An diesem Projekt sind folgende Institutionen beteiligt: Plessey Electronics Systems Research (Großbritanien), Eurosoft Systems (Frankreich), The Hatfield Polytechnic (Großbritanien), sowie die Universitäten Dortmund und Karlsruhe (Deutschland). Die dargestellten Ideen und Konzepte entstanden in enger Zusammenarbeit aller Partner. Die Arbeiten wurden am Institut für Informatik III, Forschungsgruppe Prozeßrechentechnik, Prof. Dr.–Ing. U. Rembold durchgeführt.

Literatur

1 Ada
Reference Manual for the Ada Programming Language
ANSI / MIL–STD 1815A, 1983

2 A. Goldberg, D. Robson
Smalltalk–80 – The Language and its Implementation
Addison–Wesley, 1983

3 J. Guttag, J.J. Horning
Formal Specification as a Design Tool
Xerox PARC CSL-80-1

4 C.A.R. Hoare
Communicating Sequential Processes
Communications of the ACM, August 1978

5 Inmos Ltd
Occam Reference Manual
Prentice Hall 1984

6 J. Kramer, J. Magee, M. Sloman, K. Twidle, N. Dulay
The Conic Programming Language Version 2.4
Research Report DOC 84/19
Imperial College, London, 1984

7 R.E. Strom, S. Yemini
NIL: An intergrated Language and System for Distributed Programming
Sigplan 83 Symposium on Programming Languages Issues in Software Engineering

8 J. Welsh, D.W. Bustard
Pascal–Plus – Another Language for modular Multiprogramming
Software – Practice & Experience, Vol 9, 1979

9 H. Werum, H. Windauer
Pearl
Vieweg Verlag, 1978

10 N. Wirth
Programming in Modula–2
Springer 1983, 2nd Ed.

11 M. Zelkowitz, R.T. Yeh, R. Hamlet, J. Cannon, V. Basili
State-of-Practice Survey of Software Industry
IEEE Computer, 1983

Anschrift der Autoren

Dr. Wolfgang K. Epple
BMW AG, Abt. EG–26
Knorr – Straße 146
Postfach 400240
8000 München 40, F.R.G.

Horst Spandl
Institut für Informatik III
Prof. Dr.–Ing. U. Rembold
Universität Karlsruhe
Postfach 6380
7500 Karlsruhe, F.R.G.

UNIX Mail: spandl@uka.UUCP
(alte Form: ...!unido!uka!spandl)