Konzeptioneller Ansatz zur Bildung einer Theorie des verteilten objektorientierten Programmierens

N. Schmeißer*
Forschungszentrum Rossendorf e.V.,
Abt. Kommunikation und Datenverarbeitung

7. Mai 1997

Inhaltsverzeichnis

*Telefon: (+49 351) 260/2207, E-Mail: nils@schmeisser.com,
URL: http://www.fz-rossendorf.de/FVTK/MITARB/schmei

1 Abstrakt

Objektorientiertes und paralleles Programmieren sind sind wesentliche Aspekte bei der Erstellung virtueller Verteilter Umgebungen. Es gibt bisher eine Reihe von Lösungsstrategien, die hierzu verfolgt werden (DIS, Java/Beans). Eine Theorie, die insbesondere die Ableitung von Verteilungs- und Umverteilungsstartegien gestattet ist bisher in der Literatur allerdings nicht zu finden.
Wir legen in diesem Artikel unsere Überlegungen zur Erstellung einer solchen Theorie dar. Wir beginnen mit einer kurzen Einführung in die Theorie des objektorientierten Programmierens und leiten davon für unsere weiteren Betrachtungen wichtige Konzepte ab.

2 Theorie des objektorientierten Programmierens

Grundlegendes Konzept für die theoretische Beschreibung des objektorientierten Programmierens ist der Begriff des *abstrakten Datentyps* (ADT). Wir geben hier ohne Ableitung die Definition des abstrakten Datentyps nach [1] wieder:

Definition : Gegeben sei eine Menge Ω von Sorten[1], eine Menge Δ von Operatoren über Ω und deren Spezifikation Γ zusammen mit einer homomorphen Abbildung $B : \Omega \rightarrow S$ in eine Menge semantischer Ausdrücke, dann heißt die mehrsortige $\Omega\Delta\Gamma$-Algebra $\mathcal{A}$ *abstrakter Datentyp.*

Wir identifizieren $(\{(\omega_{i_1}, ..., \omega i_k) = \omega_{i_1} + ... + \omega i_k : \omega_i \in \Omega, k \in \mathbf{N}\}, +)$ als die Permutationsgruppe über Ω (das ist gleich dem Träger von Δ). Im Folgenden soll Ω diese Gruppe bezeichnen (es ist dann $B(\omega) = \times_{i=1}^{l(\omega)} B(\omega_i)$, $\omega = \omega_{i_1} + ... + \omega_{i_k}$, $l(\omega) = k$).
Wir betrachten jetzt zwei Gruppen $\Omega \subseteq \Omega'$ und definieren eine homomorphe Abbildung $\kappa : \Omega' \rightarrow \Omega$ (Sortenkonvertierung). Der Homomorphiesatz für Gruppen liefert sofort

$$\Omega' | kern(\kappa) \cong \Omega \tag{1}$$

und damit

$$\delta : \Omega \rightarrow \Omega = \Omega' | kern(\kappa) \rightarrow \Omega' | kern(\kappa) \tag{2}$$

Ein Operator δ kann damit auch durch einen Operator über Ω' gemäß $\delta = \kappa \circ \delta'|_{\Omega'|kern(\kappa)}$ ausgedrückt werden. Das bedeutet natürlich auch

$$B((\kappa \circ \delta')(\omega)) = B(\delta(\omega)) \tag{3}$$

[1]Mit Sorte bezeichnet [1] eine Struktur $\omega = i_1 : \tau_1 + ... + i_n : \tau_n$, die aus Komponentennamen i_k mit zugehörigem Typbezeichner τ_k gebildet wird.

Definition : Seien $\mathcal{A}= \Omega\Delta\Gamma - Algebra$ und $\mathcal{A}' = \Omega'\Delta'\Gamma' - Algebra$ mit $\Omega \subseteq \Omega'$ und existiere ein Homomorphismus $\kappa : \Omega' \to \Omega$, dann heißen zwei Operatoren $\delta \in \Delta$ und $\delta' \in \Delta'$ *polymorphe Operatoren über* Ω *und* Ω', wenn gilt

$$\kappa \circ \delta'|_{\Omega'|kern(\kappa)} \equiv \delta \tag{4}$$

$\mathcal{A}'$ heißt *von* $\mathcal{A}$ *abgeleiteter abstrakter Datentyp* ($\mathcal{A}' \geq \mathcal{A}$), wenn außerdem gilt

$$\forall \delta \in \Delta \Rightarrow \exists \delta' \in \Delta' : \kappa \circ \delta'|_{\Omega'|kern(\kappa)} \equiv \delta \tag{5}$$

Wegen (3) folgern wir für polymorphe Operatoren $B(\delta'(\Omega)) = B(\delta(\Omega))$. Wir nennen diese Art des Polymorphismus den *universellen Polymorphismus.*
Betrachten wir zusätzlich eine *implizite Typkonversion* $\leq$, so gelangen wir zum Bergiff des *Ad-hoc Polymorphismus.*

Definition : Existiert für zwei Sorten ω und ω' einer $\Omega\Delta\Gamma$-Algebra ein Homomorphismus $\leq : \omega \leq \omega'$ so, daß

$$\begin{array}{l} B(\omega \leq \omega') \in B(\omega) \to B(\omega'); \\ B(\omega \leq \omega) = id; \\ \omega \leq \omega' \wedge \omega' \leq \omega'' \Rightarrow B(\omega \leq \omega'') = B(B(\omega \leq \omega') \leq \omega'') \end{array} \tag{6}$$

so heißt ω ein *Subtyp* von ω'.

$\leq$ heißt *implizite Typkonversion.* Damit wird jede Abbildung, die über ω' arbeitet zu einer polymorphen Abbildung über ω. Aufgrund der Definition folgt sofort

$$B(\Gamma_\delta(..., \omega, ...)) \to B(\Gamma_\delta(..., \omega', ...)) \tag{7}$$

Mit diesen Hilfsmitteln können wir jetzt eine Definition für den Begriff *Klasse* angeben.

Definition : Ein abstrakter Datentyp $\mathcal{A}= (\Omega\Delta\Gamma)$ zusammen mit einer Methode c zur Erzeugung und einer Methode d zum Zerstören einer $\to$ *Instanz* und einer Eingabe- und Ausgabemenge heißt *Klasse.*

$$\mathcal{C} = (a_0, P, M, I, O) \tag{8}$$

$P = \omega_0 = \Pi_0(\Omega)$ heißen *Attribute* von $\mathcal{C}$. P ist die Realisierung einer fixen Permutation von Sorten aus Ω, d.h. P ist eine Menge von Variablen $p_1 : \omega_1, ..., p_n : \omega_n$. $M = \Delta \cup \{f | f : P \times I \to P \times O\}$ heißen die *Methoden* von $\mathcal{C}$(I und O heißen die Eingabe- bzw. Ausgabemenge von $\mathcal{C}$). a_0 ist der, durch c erzeugte Anfangszustand von $\mathcal{C}$($a_0 = B(P)$ zum Zeitpunkt der Konstruktion).

Wir schreiben auch kurz $\mathcal{C} = (a_0, P, M)$. Mit $ADT(\mathcal{C})$ bezeichnen wir den zughörigen abstrakten Datentyp. Damit gelangen wir zu einer relativ simplen Beschreibung des Konzeptes der Vererbung und erklären:

Definition :

Eine Klasse B, für die

$$ADT(A) \leq ADT(B) \wedge A.M \subseteq B.M \wedge A.P \subseteq B.P \tag{9}$$

gilt, heißt *von A abgeleitete Klasse*, und wird mit $B :: A$ bezeichnet. B heißt *direkt von A abgeleitet* ($B ::_d A$), wenn es keine Klasse C gibt mit $B :: C$ und $C :: A$.

"::" wollen wir auch *is-a Relation* nennen und mit "$\rightarrow$" bezeichnen. Wir führen jetzt noch eine Reihe weiterer Begriffe ein, die wir für unsere folgenden Überlegungen benötigen.

1. Die geordnete Struktur $ic(C_0, C_n) = C_0 :: ... :: C_n$ heißt *Ableitungskette von C_0 bis C_n*. $ic(C_0, C_n)$ heißt vollständig, wenn $\forall i < j \wedge C_i, C_j \in ic(C_0, C_n) \Rightarrow \nexists C : C_i :: C :: C_j$.

2. $l(ic(C_0, C_n)) = n$ heißt *Länge der Kette*.

3. Sei $IC = \{ic(C_0, C_n)\}$ die Menge aller vollständigen Ketten, dann heißt

$$d(C_0, C_n) = min\{l(ic(C_0, C_n)), ic(C_0, C_n) \in IC\} \tag{10}$$

der *Abstand zwischen C_0 und C_n*.

4.

$$FCP(A, B) = C : \begin{array}{c} \exists ic_A(A, C) \wedge \exists ic_B(B, C) \\ \wedge d(A, B) = l(ic(A, C)) \\ \wedge d(B, C) = l(ic(B, C)) \end{array} \tag{11}$$

heißt **ein** *erster gemeinsamer Vorfahr von A und B*.

5.

$$grad(A, B) = d(A, FCP(A, B)) + d(B, FCP(A, B)) \tag{12}$$

heißt *Abstand zwischen A und B* oder *Grad der Verwandtschaft zwischen A und B*,
$B ::_d A \Rightarrow grad(A, B) = 1$.

6.

$$IG(C_0) = (\{C_i : C_i :: C_0\}, \{(C_i, C_j) : C_i ::_d C_j :: C_0\}) \tag{13}$$

heißt *Ableitungsgraph* oder *Ableitungshierarchie* von C_0.

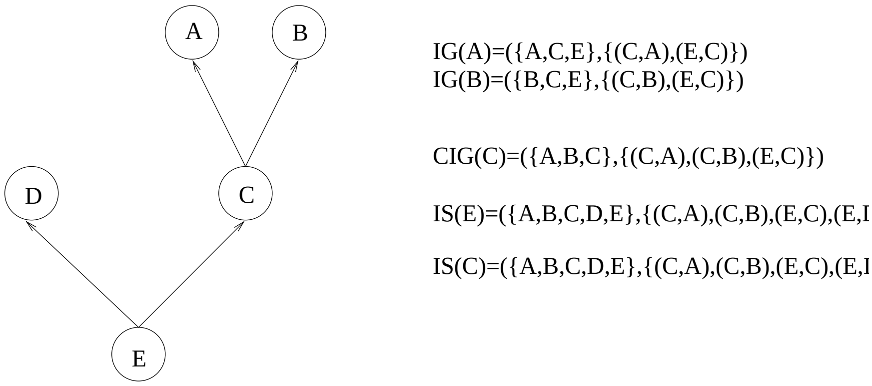

Abbildung 1: Ableitungsstrukturen

3 Abgeleitete Konzepte

Ausgehend von den genannten Konzepten führen wir jetzt den *kompletten Ableitungsgraphen einer Klasse C* gemäß

$$CIG(C) = \bigcup_{A_i | C \in IG(A_i)} IG(A_i) \tag{14}$$

ein und leiten davon die *Ableitungsstruktur um eine Klasse C* ab.

$$IS(C) = CIG(C) \bigcup \{IS(A_i) | A_i \in CIG(C)\}. \tag{15}$$

Wir verdeutlichen dies an Abbildung 1.
Untersuchen wir die Attribute einer Klasse, so stellen wir fest, das es eine weitere Beziehung zwischen Klassen geben kann, die wir *has-a Relation* nennen.

Definition : Seien A und B Klassen und existiert ein $x = x_i \in A.P | \omega_i = B$, dann heißt B *Komponentenklasse* von A oder *zwischen A und B besteht eine has-a Relation* ($A \bullet\!\!- B$).

Ausgehend von einem *has-a Graphen* einer Klasse C

$$HA(C) = (\{A_i | C \bullet\!\!- A_i\}, \{(C, A_i)\}) \tag{16}$$

gelangen wir zum Begriff der *has-a Struktur*

$$HS(C) = HA(C) \cup \bigcup_{A_i \in HA(C)} HA(A_i) \cup \bigcup_{C \in HA(A_i)} HA(A_i). \tag{17}$$

Wir können die bis jetzt erklärten Beziehungen zwischen Klassen ausnutzen um einen *IH-Graphen* zu erklären.

$$IH(C) = IS(C) \cup \bigcup_{A_i \in HS(B_j) | B_j \in IS(C)} IS(A_i). \tag{18}$$

Für diese Struktur geben wir zum besseren Verständnis eine formale Beschreibung zur Bildung an:

bilde $IS(C)$
für alle Knoten $I_i \in IS(C)$
  bestimme $HS(I_i)$
  für alle Knoten H_j von $HS(I_i)$
    bestimme $IS(H_j)$ und füge $IS(H_j)$ zu $IH(C)$

Wir werden die IH Struktur später zu einer IHU Struktur ausbauen, die zusätzlich alle $use - a$ Relationen betrachtet.

4 Ausblick

Auf der Grundlage dieser Konzepte, insbesondere des IHU-Graphen können wir ein Schema für die Verteilung von Instanzen angeben. Wir führen für die Kanten des IHU-Graphen eine geeignete Bewertung ein und können mit einem Minimalschnitt des Graphen eine suboptimale Verteilung der Instanzen bezüglich der Kriterien Kommunikationsaufwand und Knotenlast angeben.
Diese Verteilung ist a priori ausführbar. Um für eine optimale Verteilung während der Laufzeit zu sorgen, müssen wir das "Snap-shot" Verfahren einführen. Sobald eine Umverteilung nötig wird, muß jede Instanz ihren Zustand speichern (Snap-shot), danach wird der aktuelle IHU-Graph aufgebaut und die Umverteilung findet statt. Nach der Umverteilung rekonstruiert jede Instanz ihren Zustand und die Arbeit kann fortgesetzt werden. Wir fordern also für alle Objekte in der verteilten Umgebung die Eigenschaft der *Persistenz.*
Unsere nächsten Überlegungen werden sich mit der Notwendigkeit einer Synchronisation zur Snap-shot-Zeit auseinandersetzen.

Literatur

[1] Carl A. Gunter, John C. Mitchell:
"Theoretical Apsects of Object-Oriented Programming";
The MIT Press, 1994

[2] Richard Wiener:
"An Object-Oriented Introduction to Computer Science Using Eiffel";
Prentice Hall, 1996

[3] Andreas Heuer:
"Objektorientierte Datenbanken";
Addison-Wesley; 1992

[4] Nils Schmeißer:
"Seminar Objektorientiertes Programmieren: Theorie, Methodik und Anwendungen";
Seminar-Script, FVTK, Forschungszentrum Rossendorf; 1997