Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

Alfred Lupper
Universität Ulm
Abt. Verteilte Systeme
D-89061 Ulm, Germany
lupper@informatik.uni-ulm.de

1. Einleitung

Distributed Shared Memory (DSM) ist neben Message Passing und Tuple Spaces eine mögliche Methode für verteiltes Rechnen auf Arbeitsplatzrechnern. DSM erlaubt einzelnen Rechnern ihren verfügbaren Speicherbereich transparent auf den Speicher weiterer Rechner auszudehnen und über den gemeinsamen Speicher miteinander zu kommunizieren. Der Vorteil von DSM gegenüber anderen Methoden liegt in der einfachen Programmierbarkeit eines solchen Systems. DSM ist für den Entwickler verteilter Anwendungen wesentlich einfacher zu handhaben als das Message Passing-Paradigma, das ihm die explizite Partitionierung der Daten und die Verwaltung der Kommunikation abverlangt.

Bisher wurde verteilter gemeinsamer Speicher vor allem in enggekoppelten Multiprozessorsystemen verwendet. Mit zunehmender Zuverlässigkeit und Geschwindigkeit der Netzwerke – besonders im lokalen Bereich – wird diese Methode der Kommunikation auch für lose gekoppelte Systeme interessant.

Der folgende Artikel beschäftigt sich mit Fragen der Namensverwaltung und Adreßabbildung für Distributed Heap Storage [Traub, 94], einem flexiblen DSM-System für lose gekoppelte Rechner, das einen gemeinsamen Heap-Speicher realisiert. Es wird gezeigt, wie eine Namensverwaltung für DSM-Systeme zu gestalten ist, um das verteilte Memory Management zu unterstützen und die positiven Eigenschaften von Distributed Heap Storage für die Realisierung eines Naming-Systems selbst zu nutzen. Die Plazierung der Namenstabellen im logischen Adreßraum ermöglicht es, das Naming-System ohne direkten Zugriff auf Netzwerkdienste und ohne explizite Kenntnis der Verteiltheit zu realisieren.

2. Das DHS-Modell und seine Abstraktionsebenen

Das vorliegende DHS-Modell gliedert sich in drei Abstraktionsebenen mit unterschiedlichen Sichten des Systems: Auf der Objektebene befinden sich Objekte, die syntaktische Namen tragen und denen der physikalische Speicherort nicht bewußt ist. Objekte dieser Ebene und anonyme Objekte ohne Namen werden auf kohärente Speicherblöcke des virtuellen verteilten Speichers projiziert. Der ein Objekt repräsentierende Speicherbereich dieser Ebene ist durch eine logische Adresse, die innerhalb einer Gruppe von Knoten (Cluster) eindeutig ist, und durch die Größe des Speicherblocks bestimmt. Speicherbereiche dieser Ebene werden transparent auf den physikalischen Speicher eines oder mehrerer Knoten abgebildet, d.h. ein Objekt ist physikalisch u.U. auf mehreren Knoten existent (vgl. Abbildung 2.1). Große Objekte können sich dabei auch über den physikalischen Speicher mehrerer Knoten erstrecken, falls möglich sollte das aber vermieden werden, da dadurch für das Memory Management ein hoher Verwaltungsaufwand entsteht. Objekte können somit auf Knoten partitioniert und auch repliziert sein. Diese Zuordnung von Speicher zu Objekt und den schematischen Aufbau der Schichten des DHS-Systems zeigt Abbildung 2.1:

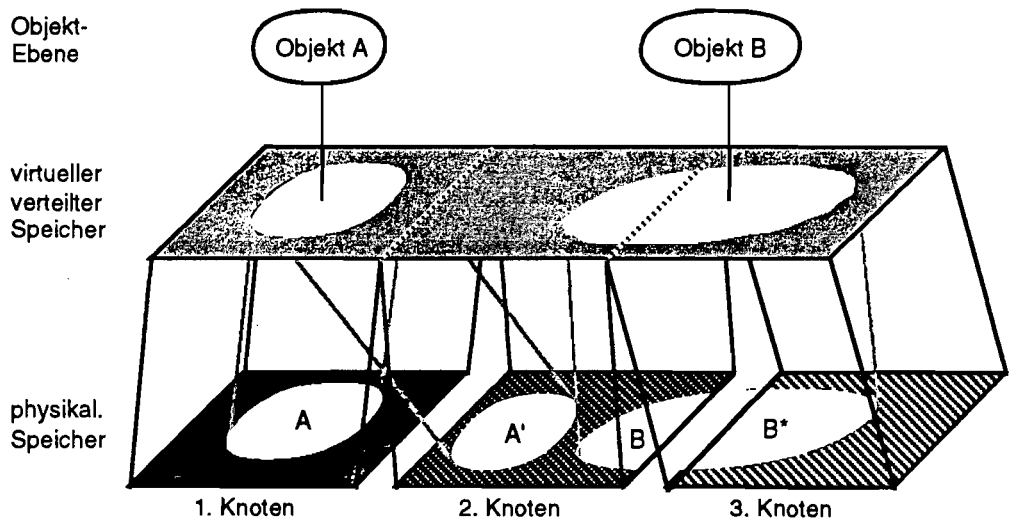

Abbildung 2.1: Die Ebenen des DHS-Modells

Abbildung 2.1 läßt zudem erkennen, daß zwischen den einzelnen Schichten und ihren Abstraktionsebenen jeweils Abbildungen notwendig sind, um ausgehend vom Namen eines Objekts den physikalischen Speicherort zu ermitteln. Die Abbildungen zwischen den Objekten und den logischen Adressen werden vom Naming-System übernommen. Die physikalische Repräsentation eines Objekts ermittelt das Memory Management. Wie obige Abbildung ebenso verdeutlicht, können Objekte im physikalischen Speicher auf Knoten repliziert (A, A') und über Knoten verteilt (B, B*) gespeichert sein, so daß die Abbildung zwischen logischen und physikalischen Adressen nicht notwendigerweise eindeutig ist.

3. Namen und Objekte in DHS

Nachdem das Modell des DHS vorgestellt wurde, sollen im folgenden Kapitel die im Modell auftretenden Begriffe Name, logische und physikalische Adresse, Knoten und Cluster definiert und deren Eigenschaften in DHS beschrieben werden.

3.1. Eigenschaften von Objekten und Objektbezeichnern

Alle Aktionen eines Benutzers in DHS beziehen sich auf Objekte. Für DHS sind Objekte nichts weiter als logisch zusammenhängende Speicherblöcke. Die Gesamtheit der in DHS vorkommenden Objekte wird durch die Menge O beschrieben.

$$O = \{o | o \text{ ist Objekt in DHS}\}$$

Objekte des realen Lebens können durch unterschiedliche Kennzeichen identifiziert werden. In Rechnersystemen werden Objekte gewöhnlich durch Bezeichner[1] (object identifier) identifiziert, die eindeutig in Zeit und Raum sind und durch Kardinalzahlen (32 Bit, 64 Bit) realisiert werden. Die Menge der erzeugten Objektbezeichner I ist potentiell unendlich und stets gleich mächtig wie die Menge der existierenden Objekte O[2].

$$I = \{i \in \mathrm{N} | \exists o \in O : i \text{ ist } \textit{Identifier} \text{ von } \textit{Objekt } o\}$$

In realen Systemen ist die Anzahl der möglichen Bezeichner in der Regel durch die Wortlänge des Systems beschränkt, wodurch nur endlich viele Objekte verwaltet werden können.

3.2. Eigenschaften von Namen

Da Objektbezeichner für den Benutzer wenig einprägsam sind, tragen Objekte in DHS *Namen*, die vom Benutzer vergeben werden[3]. Die vergebenen Namen können im einfachsten Fall flach oder primitiv sein. Sie bestehen aus einer Folge von Buchstaben oder Zahlen eines endlichen Alphabets Σ:

$$SimpleName = letter\{letter | digit\}_0^* \text{ mit } digit \in S,\ letter \in S$$

Der durch solche Namen erzeugte Namensraum N_f ist jedoch nicht sehr mächtig. Bei der Vergabe von Namen muß die Eindeutigkeit stets systemweit gewährleistet sein. Flache Namen enthalten jedoch keinerlei Ortsinformation, so daß die Verwaltung eines flachen verteilten Namensraums in verteilten Systemen einen hohen Kommunikationsaufwand erfordert.

$$N_f = \{n | n = SimpleName\}$$

---

1 Objektbezeichner werden auch als Systemnamen aufgefaßt.

2 Man kann Namen und Identifier wie ein Betriebsmittel verstehen, das erzeugt, belegt und wieder freigegeben wird. Vgl. dazu [Mattes, 91].

3 Protocols use Addresses, People prefer Names.

Flache Namen sind für Menschen auch nur dann einprägsam, wenn sie Wörtern einer natürlichen Sprache entsprechen. In flachen Namensräumen wird es daher mit steigender Anzahl von Namen immer schwieriger aussagekräftige Namen zu finden.

Aus diesem Grund werden Namen *strukturiert* . Strukturierte Namen bestehen aus dem primitiven Objektnamen selbst und der Bezeichnung eines Kontextes, die sich selbst wieder aus primitiven Namen zusammensetzt.

$$StructName = SimpleName\{"."\ SimpleName\}_0^*$$

Strukturierte Namen setzen sich bei der Generierung aus einer Folge einfacher Namen zusammen und werden bei der Auflösung wieder in solche zerlegt. Syntaktisch strukturierte Namen aus Objekt- und Kontextbezeichnung (z.B. pc.vs.info.uni-ulm) spannen einen hierarchischen Namensraum N_h auf.

$$N_h = \{n | n = StructName\}$$

Erst ein hierarchischer oder syntaktisch partitionierter Namensraum ermöglicht die effiziente Dezentralisierung von Namensoperationen und die Delegierung der Zuständigkeiten auf Verwalter.

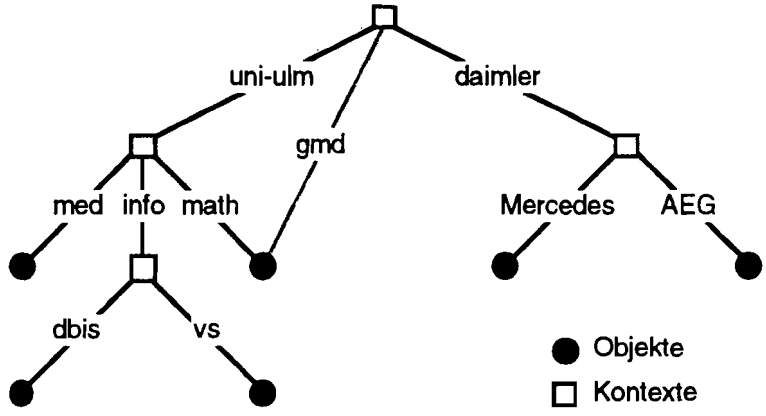

Abbildung 3.1: Hierarchischer Namensraum mit strukturierten Namen

Hierarchische Namensräume lassen sich als Graph darstellen. Knoten in diesem Graphen stellen Objekte oder Kontexte dar. Die Namen der Objekte werden an den Kanten des Graphen angetragen. Benannt werden also nicht die Objekte selbst, sondern die Übergänge zwischen den einzelnen Kontexten und Objekten. Zu beachten ist, daß ein Knoten im Namensgraphen nicht notwendiger Weise nur einen Vorgänger besitzt[4]. Somit kann dasselbe Objekt in unterschiedlichen Kontexten unterschiedliche Namen tragen. In jedem Fall müssen jedoch Zyklen im Graphen vermieden werden, um ein Terminieren der Operationen im Namensgraphen zu gewährleisten. Ein Namensbaum ist gerichtet, daher ist die Reihenfolge der Komponenten eines strukturierten Namens signifikant. Eine Beschreibung des Graphen aus Abbildung 3.1 kann wie folgt aussehen:

---

[4] In Unix werden z.B. sog. Links Verweise in andere Directories möglich.

Wurzel = {(uni-ulm{(med), (info{(dbis, vs)}), (bio)}), (daimler{(Merzedes), (AEG)}), (gmd)}

In einer höheren Programmiersprache wäre die Beschreibung dieses Namensgraphen auch als auch als verschachtelte Record-Struktur darstellbar:

```
Wurzel =  RECORD
            uni-ulm:  RECORD
                        med: Objekt;
                        info: RECORD
                                dbis: Objekt;
                                vs: Objekt;
                              END;
                        math: Objekt;
                      END;
            gmd: Objekt;
            daimler:  RECORD
                        Mercedes: Objekt;
                        AEG: Objekt;
                      END;
          END;
```

Abbildung 3.2: Darstellung eines hierarchischen Namensraums als Record-Struktur

In der Oberon-basierten Umgebung des DHS können die strukturierten Namen von Objekten aus einer Konkatenierung von Modul-, Command- und Objekt- und Feldnamen bestehen (vgl. [Wirth, 92]).

3.3. Eigenschaften von Adressen in DHS

Objekte in DHS liegen im logischen Adreßraum und besitzen innerhalb einer bestimmten Gruppe von Knoten eindeutige logische Adressen[5]. Daher bietet es sich im DHS-System an, als Objektbezeichner die *logische Adresse* l eines Objekts im gemeinsamen Speicher zu wählen. Das hat den Vorteil, daß bei der Abbildung vom Objekt auf den das Objekt repräsentierenden Speicher eine Indirektionsstufe entfallen kann.

$$L = \left\{ l \in \mathrm{N}_0 \mid l \text{ ist logische Adresse} \right\}$$

Eine Gruppe von Knoten, die eindeutige logische Adressen gewährleistet, wird zu einem Cluster zusammengefaßt. Somit läßt sich die Menge der in einem Cluster eindeutigen logischen Adressen wie folgt definieren:

$$L_C = \left\{ l \in L \mid l \text{ ist eindeutige logische Adresse im Cluster } c \right\}$$

Logische Adressen als Objektbezeichner haben allerdings den Nachteil, daß sie einem Objekt nur temporär inhärent sind. Sie können sich in DHS mit der Zeit ändern wodurch eine Anpassung notwendig wird.

Neben den bereits erwähnten logischen Adressen existieren in DHS physikalische Adressen, die den tatsächlich vorhandenen Haupt- oder Hintergrundspeicher repräsentieren. Physikalische Speicheradressen sind nicht im Gesamtsystem oder im Cluster eindeutig, sondern

---

5 Eine ausführliche Diskussion relativer Objektbezeichner wird in [Fujinami, 92] gegeben.

lediglich innerhalb eines einzelnen Knotens. Die Menge P bezeichnet alle im System existenten physikalischen Adressen[6].

$$P = \{p \in \mathrm{N} | \exists k \in K, p \text{ ist physikalische Adresse in } k\}$$

Betrachtet man nur die physikalischen Adressen innerhalb eines Knotens P_k, so ist die Eindeutigkeit durch die physikalische Speicherstruktur gegeben.

$$P_k = \{p \in P | p \text{ ist physikalische Adresse im Knoten } k \in K\}$$

3.4. Eigenschaften von Clustern und Knoten

Die Menge der physikalischen Adressen ist endlich. Physikalische Adressen sind daher nur innerhalb eines bestimmten Bereichs eindeutig. Dieser Bereich wird durch einen Knoten repräsentiert. Die Menge aller Knoten bezeichnen wir mit K.

$$K = \{k | k \text{ ist Knoten}\}$$

Knoten werden durch eine eindeutige Knotennummer[7] identifiziert. Knoten und Knotennummer werden daher im folgenden äquivalent verwendet. Um außerhalb eines Knotens die Eindeutigkeit physikalischer Adressen zu erreichen, muß beim Zugriff auf physikalische Adressen stets noch die Knoten-Nummer ergänzt werden.

Ebenso endlich ist die Menge der in realen Systemen verfügbaren logischen Adressen. Diese Menge reicht daher nicht aus, um die potentiell unendliche Menge von Objekten zu adressieren. Die Menge der Knoten wird daher in Teilmengen, Cluster, zerlegt, in denen die logischen Adressen eindeutig sind. Bei Zugriffen über Cluster-Grenzen hinweg muß daher zusätzlich eine eindeutige Cluster-Nummer angegeben werden[8].

$$c = \left\{ k \in K \middle| l_i \neq l_j,\ l_i, l_j \in L \text{ ist logische Adresse in } k, \forall\, i \neq j \right\}$$

Die Menge aller Cluster konfirmiert sich aus disjunkten Mengen von Knoten, d.h. es gibt stets eine eindeutige Zuordnung von Knoten zu Cluster. Zwischen den Clustern wird nur mit Cluster-Nummern und logischen Adressen operiert.

$$C = \left\{ c \middle| c \subseteq K,\ c_i \cap c_j = \{\ \},\ \forall\, i \neq j \right\}$$

Die Existenz von Clustern ist lediglich dem Memory Management und dem Naming-System bekannt. Applikationen sehen die Cluster-Grenzen nicht. Sie hantieren nur mit logischen Adressen, die bei cluster-übergreifenden Operationen vom Memory Management in clusterexterne Adressen transformiert werden.

---

6 Adressen können als physikalische Namen aufgefaßt werden, die einen flachen Namensraum bilden.

7 Als eindeutige Knotennummer kann z.B. die Ethernet-Adresse dienen, oder es wird durch ein Voting-Protokoll innerhalb eines Clusters eine eindeutige Knotennummer bestimmt.

8 und/oder die Adresse in eine andere Adresse transformiert werden

4. Naming-Abbildungen in DHS

Ausgehend von der Definition der relevanten Mengen werden in diesem Kapitel die Beziehungen zwischen Objekten, Namen, logischen bzw. physikalischen Adressen, Knoten und Clustern als Abbildungen zwischen den jeweiligen Mengen beschrieben.

Ganz allgemein kann ein *Naming-System* als eine Funktion Φ verstanden werden, die die Menge der Namen N auf die Menge der Objekte O abbildet[9].

$$\Phi: N \rightarrow O \; mit \; \Phi(n) = o, \; o \in O, \; n \in N$$

Ist die Funktion Φ bijektiv, so handelt es sich um eine eins-zu-eins-Abbildung und $n \in N$ bezeichnet eindeutig ein Objekt $o \in O$. Das ist jedoch nicht immer der Fall. So kann ein Objekt durch mehrere Namen bezeichnet werden, und Namen können auch eine ganze Gruppe von Objekten identifizieren. Ebenso kann es Objekte geben, die keinen Namen tragen und somit für den Benutzer nicht präsent sind.

Elemente aus der Menge der Namen können ihrerseits wieder als Objekte verstanden werden und durch ein anderes Naming-System benannt werden. Somit entstehen verschiedene Ebenen der Abstraktion. Naming-Systeme können also mehrstufig sein. Als Beispiel sei hier nur die Abbildung von Dateinamen auf I-Nodes und Dateien oder die Abbildung von Domain Namen auf IP-Adressen und Ethernet-Adressen genannt.

Bevor wir die Beziehungen zwischen den Mengen weiter formal beschreiben, sollen diese zunächst an Hand von Tabellen anschaulich dargestellt werden.

4.1. Beziehungen zwischen Objekten und Objektbezeichnern

Objekte werden in modernen Rechnersystemen auf der Benutzerebene in der Regel durch grafische Symbole dargestellt. Mit diesen Symbolen kann zwar der Anwender sehr bequem arbeiten, nicht jedoch das System selbst. Protokolle verwenden für die Indentifizierung von Objekten sogenannte Objektbezeichner, in der Regel ganze positive Zahlen mit ausreichender Kardinalität.

Objekt	*Objekt-Bezeichner*
	#1
	#2
	#3

Abbildung 4.1: Tabelle zur Zuordnung von Objektbezeichnern zu Objekten

---

9 Es gilt zu beachten, daß die Namen nicht notwendigerweise Benutzernamen sein müssen und die Objekte wiederum Namen repräsentieren können.

Bei der Erzeugung von Objekten wird das System daher für jedes Objekt einen Bezeichner generieren, der das Objekt für die gesamte Lebensdauer eindeutig in Zeit und Raum von anderen Objekten unterscheidet. Diese Zuordnung verdeutlicht die Tabelle in Abbildung 4.1. Formal läßt sich die Zuordnung wie folgt als Funktion angeben:

Definition:

> Die Funktion $\Omega: O \rightarrow I$ heißt *Naming-System von Objekt zu Bezeichner* und ordnet dem Objekt $o \in O$ den Bezeichner $i \in I$ eindeutig zu.

Die Zuordnung eines gültigen Objektbezeichners zu einem Objekt ist, wie bereits bemerkt, eineindeutig (bijektiv). Ein Objekt kann nicht mehrere Objektbezeichner haben und jedem Objektbezeichner ist genau ein Objekt eindeutig zugeordnet. Die Abbildung Ω ist also bijektiv und somit umkehrbar. Da zwischen Objekten und Bezeichnern eine Bijektion besteht, werden wir im folgenden Objekte und Bezeichner äquivalent verwenden.

4.2. Beziehungen zwischen Objekten und Namen

Für den Anwender ist die Möglichkeit zur Manipulation von Objekten anhand von Symbolen zwar nützlich, zur genaueren Differenzierung reicht eine grafische Repräsentation jedoch i.a. nicht aus[10]. Auch die Unterscheidung von Objekten anhand von Objektbezeichnern ist für den Anwender nicht praktikabel, da Zahlen aufgrund der fehlenden Struktur wenig einprägsam sind und der optischen Verarbeitungsweise des menschlichen Verstandes nicht entgegenkommen. Namen hingegen eignen sich viel besser für Menschen. Dies ist der Grund, warum Objekten zusätzlich zu Objektbezeichner und eventueller grafischer Repräsentation Namen zugeordnet sind, die sie für den Anwender identifizierbar machen.

Objekt-Name	*Objekt-Bezeichner*
{ Phone.AT&T, Telephone }	#1
{ MyMac, PC.info.ulm }	#2
{ Ball }	#3

Abbildung 4.2: Tabelle zur Abbildung von Namen auf Objektbezeichner

Eine Zuordnung von Namen zu Objektbezeichnern zeigt Abbildung 4.2. Hier läßt sich bereits erkennen, daß ein Objekt(bezeichner) $o \in O$ mehrere Namen $\{n\} \subset N$ tragen kann. Umgekehrt ist es durchaus denkbar, daß einem Namen $n \in N$ eine ganze Gruppe von Objekten $\{o\} \in O$ zugeordnet ist. In unserem Fall wollen wir das jedoch ausschließen, da Gruppen von Objekten selbst wieder als Objekte behandelt werden.

---

[10] Ähnliche Objekte besitzen dieselben Symbole. Ferner ist es schwierig für jedes Objekt ein aussagekräftiges grafisches Symbol zu finden.

Definition:

Die Funktion $\Psi: N \rightarrow O$ heißt *Naming-System von Name zu Objekt* und ordnet dem Namen $n \in N$ das Objekt $o \in O$ zu.

Diese Funktion Ψ ist weder injektiv, noch surjektiv, da ein Objekt mehrere Namen tragen kann und nicht jedes Objekt zwangsläufig einen Namen trägt. Die Umkehrabbildung Ψ^{-1} von Objekt zu Namen hingegen ist surjektiv, jedoch nicht injektiv, da ein Name eine ganze Gruppe von Objekten bezeichnen kann.

4.3. Abbildung von Namen auf logische Adressen in DHS

Bisher haben wir die Beziehungen zwischen Namen, Objekten und Objektbezeichnern betrachtet. Wie bereits angedeutet läßt sich in DHS eine Beziehung zwischen Objekten, logischen Adressen und Clustern herstellen. Die Bijektivität zwischen Objekten und Objektbezeichnern haben wir bereits festgestellt. Wir betrachten daher im folgenden zunächst den Zusammenhang zwischen Objektbezeichnern, logischen Adressen und Clustern.

Objekt-Bezeichner	*Cluster-Nummer*	*logische Adresse*
#1	1	$FB013425
#2	7	$5E6F627A
#3	2	$A1E67980

Abbildung 4.3: Tabelle zur Abbildung von Namen auf Cluster und logische Adressen

Die einzelnen Abbildungen zwischen Objektbezeichnern und Clustern und Objektbezeichnern und logischen Adressen sind eindeutig, jedoch nicht deren Umkehrabbildungen. Einem Cluster oder einer logischen Adresse können mehrere Objektbezeichner zugeordnet sein. Im folgenden wird aus diesen beiden Teilabbildung eine neue Abbildung definiert:

Definition:

Die Funktion $\Theta: O \rightarrow C \times L$ heißt *Naming-System von Objekt zu Cluster und logischer Adresse* und ordnet dem Objekt $o \in O$ das Tupel $(c,l) \in C \times L$ zu.

Da die Funktion Θ bijektiv ist, kann nun eine Funktion aufgestellt werden, die die Zuordnung von Namen nach Cluster und logischer Adresse übernimmt.

Um ausgehend vom Namen eines Objekts den physikalischen Speicher eines Objekts zu bestimmen, muß zunächst die Nummer des Clusters ermittelt werden, in dem sich das Objekt befindet. Ein Objekt existiert in DHS zu einem Zeitpunkt stets nur in einem Cluster, d.h. die Abbildung von Objekt auf Cluster-Nummer ist stets eindeutig.

Definition:

Die Funktion $\alpha: N \rightarrow C$ heißt *globales Naming-System von Name nach Cluster* und bildet den Namen $n \in N$ auf den zugehörigen Cluster $c \subset C$ ab.

Innerhalb des Clusters kann dann aus dem Namen die logische Adresse eines Objekts bestimmt werden.

Definition:

Die Funktion $\beta: C \times N \rightarrow L$ heißt *cluster-internes Naming-System von Name und Cluster nach logischer Adresse* und bildet den Namen $n \in N$ im Cluster $c \subset C$ auf die zugehörige logische Adresse $l_C \subset L_C$ ab.

Diese beiden Teilfunktionen lassen sich zu einer einzigen Funktion zusammenfassen. Abbildung 4.4 zeigt die Zuordnung von Objektnamen zu Cluster-Nummern und logischen Adressen.

Objekt-Name	*Cluster-Nummer*	*logische Adresse*
Telephone	1	$FB013425
PC.info.ulm	7	$5E6F627A
MyMac	7	$5E6F627A

Abbildung 4.4: Tabelle zur Abbildung von Namen auf Cluster und logische Adressen

Die zusammengesetzte Funktion ist nicht injektiv, da mehrere Namen auf dasselbe Tupel aus Cluster-Nummer und logischer Adresse (c, l) abgebildet werden können.

Definition:

Die Funktion $\Lambda: N \rightarrow C \times L$ heißt *Naming-System von Name nach Cluster und logischer Adresse* und bildet einen gegebenen Namen $n \in N$ auf das Tupel $(c,l) \in C \times L$ ab.

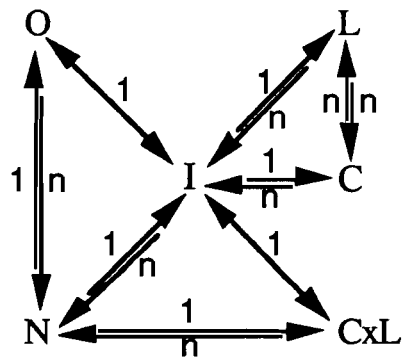

Abbildung 4.5: Beziehungen zwischen Namen, Objekten und Adressen

Die Beziehungen zwischen den einzelnen Mengen (Namensräumen) werden noch einmal in Abbildung 4.5 verdeutlicht. Hier ist zu erkennen, daß die Bijektionen zwischen den Objekten und Bezeichnern sowie zwischen Bezeichnern und Clustern und logischen Adressen zur Vereinfachung der Abbildungen von Namen auf logische Adressen führen.

4.4. Abbildung von logischen auf physikalische Adressen

Sind Cluster-Nummern und logische Adressen eines Objekts durch das Naming-System ermittelt, so muß vom Memory Management die logische Adresse auf *Knotennummern* und *physikalische Adressen* abgebildet werden. Das Bestimmen der physikalischen Adressen zur logischen Adresse geschieht nicht explizit durch die Namensverwaltung, sondern implizit durch das Memory Management, indem es logische auf physikalische Speicherseiten abbildet.

Cluster-Nummer	*logische Adresse*	*Knoten-Nummer*
1	$FB013425	{ 22, 25 }
7	$5E6F627A	{ 45 }
2	$A1E67980	{ 27 }

Abbildung 4.6: Tabelle zur Abbildung von Cluster-Nummer und logischer Adresse auf Knoten

Der zu einem Objekt gehörende physikalische Speicher kann repliziert und/oder über mehrere Knoten partitioniert sein. Daher existieren zu einer logischen Adresse u.U. in einem Cluster mehrere zugehörige Knoten(nummern), auf denen sich ein Teil des durch die logische Adresse identifizierten Objekts befindet.

Definition:

> Die Relation $\Sigma: C \times L \rightarrow K$ heißt *cluster-internes Mapping von logischer Adresse und Cluster auf Knoten* und bildet logische Adressen $l \in L$ im Cluster $c \in C$ auf die Nummern derjenigen Knoten $\{k\} \in K$ ab, die einen Mapping-Eintrag mit der logischen Adresse l enthalten, d.h.:

$$\Sigma(l,c) = \{k \in c | \exists p \in P:\ p \text{ ist physikalische Adresse zur logischen Adresse } l \text{ im Knoten } k\}$$

Knotennummern und physikalische Adressen sind bei der Kommunikation zwischen Clustern nicht sichtbar. Die Kommunikation zwischen den Knoten eines Clusters verwendet nur logische Adressen von Objekten und Knotenadressen, auf denen sich die zugehörigen Mapping-Einträge des Memory Managements befinden. Die Abbildung von logischen auf physikalische Adressen findet stets lokal innerhalb eines Knotens statt, so daß physikalische Adressen nicht außerhalb der Knotengrenzen, sondern nur lokal innerhalb eines Knotens auftreten.

logische Adresse	*Knoten-Nummer*	*physikalische Adresse*
$FB013425	22	$12F3AF87
$FB013425	25	$E2F6AF87
$5E6F627A	45	$2F3A4E71
$A1E67980	27	$02F3AE59

Abbildung 4.7: Abbildung logischer Adressen auf physikalische Adressen innerhalb eines Knotens

Definition:

> Die Relation $\Pi: L \times K \rightarrow P$ heißt *lokales Mapping von logischer Adresse und Knoten auf die physikalische Adresse* und bildet im Knoten $k \in K$ die logische Adresse $l \in L$ auf die physikalische Adresse $p \in P$ ab.

Die Verwaltung von Cluster- und Knotennummern ist nicht Aufgabe des Naming-Systems, sondern vielmehr Aufgabe des Netzwerk-Managers. Cluster-Nummern projiziert der Netzwerk-Manager auf Netzwerkbereiche, was für die Naming-Funktionen transparent ist. Die Abbildung der im Cluster eindeutigen Knotennummer auf eine Knotenadresse erledigt ebenso das Netzwerk-Management.

Abbildung 4.8: Abbildung logischer auf physikalische Adressen

Ist eine logische Adresse lokal nicht bekannt, so müssen vom Memory Manager diejenigen Knoten ermittelt werden, auf denen sich Mapping-Einträge zur gesuchten logischen Adresse befinden. Auch die lokale Mapping-Funktion wird nicht etwa vom Naming-System bereitgestellt, sondern vom Memory Manager des lokalen Knotens.

Für das Naming-System in DHS verbleiben daher vor allem folgende Aufgaben:

- Die globale Verwaltung und die Bestimmung der Cluster-Nummer, in dem das zu einem Namen gehörende Objekt verwaltet wird. Diese Problemstellung wollen wir in diesem Artikel nicht behandeln. Der Leser sei hierzu auf [Lupper, 95] verwiesen.
- Die Verwaltung strukturierter Namen zu logischen Adressen innerhalb eines Clusters. In den folgenden Kapitel werden wir uns eingehend mit dieser Thematik beschäftigen.

5. Naming bei Operationen auf Speicherobjekten

Der folgende Abschnitt befaßt sich mit den Aufgaben des Naming-Systems beim Generieren, Zugreifen, Verschieben und Löschen von Speicherobjekten im logischen Adreßraum. Der Naming-Service in DHS kooperiert eng mit dem Memory Management, wie Abbildung 5.1 verdeutlicht.

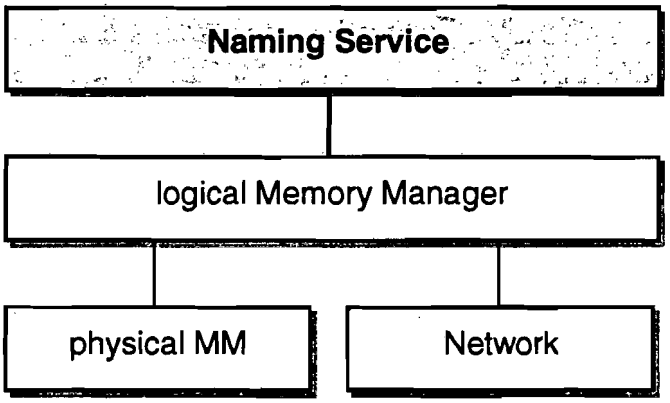

Abbildung 5.1: Zusammenwirken von Memory Manager und Naming-Service

Das Zusammenwirken besteht konkret darin, daß einerseits der Naming-Service den logischen Memory Manager für die Verwaltung der Namenstabellen verwendet, andererseits das Naming-System vom logischen Memory Manager gerufen wird, um bei Änderungen der Speicherbelegungen im logischen Adreßraum die Abbildungsfunktion ResolveName zu aktualisieren. Wie aus Abbildung 5.1 zu erkennen ist, greift das Naming-System nicht direkt auf den physikalischen Speicher und das Netzwerk zu, sondern nur über den Netzwerk-Manager und den Memory Manager. Der Speicherort von Namenseinträgen und Objekten ist für das Naming-System somit in keiner Weise sichtbar. Die folgenden Abschnitte behandeln im Detail die Interaktionen zwischen Memory Manager und Naming-System. Die Aufgaben der einzelnen Funktionen des Naming-Systems werden in Abschnitt 6 beschrieben.

5.1. Die Registrierung von Objekten im Naming-System

In Programmen treten zum einen globale Objekte, wie Variablen, Typen, Konstanten und Prozeduren auf, die an den Programmcode gebunden sind, zum anderen lokale Objekte, wie lokale Variablen und Übergabeparameter, die auf dem Stack angelegt werden. Neben diesen statischen Objekten werden zur Laufzeit dynamisch Datenblöcke auf dem Heap alloziert und wieder freigegeben.

Art des Objekts	Speicherort	Scope
globale Objekte (globale Variablen, Typen, Prozeduren und Konstanten)	befinden sich als Datenblock beim Modulcode	werden durch ein Symbol exportiert und damit beim Linken registriert
lokale Objekte (Parameter, Variablen, Typen, Prozeduren und Konstanten)	liegen auf dem Stack des Programms	werden nicht exportiert
dynamisch erzeugte Objekte (dynamische Datenblöcke)	befinden sich auf dem Heap	werden explizit durch RegisterName beim Naming-System registriert

Abbildung 5.2: Registrierung verschiedener Objektarten

Vom Naming-System in DHS können nur globale Programmobjekte und dynamische Objekte auf dem Heap explizit exportiert werden. Lokale Variablen auf dem Stack beeinflussen das Laufzeitverhalten von Programmen sehr stark und sind meist nur von kurzer Lebensdauer, so daß ein Export außerhalb eines Programmes wenig sinnvoll erscheint. Die folgenden beiden Abschnitte beschäftigen sich daher ausschließlich mit der Registrierung von globalen und dynamischen Programmobjekten.

5.1.1. Registrierung beim Übersetzen von Programm-Modulen

Über das Naming-System registriert werden sollen nur globale Programmobjekte, die in Anlehnung an die Oberon-Syntax [Wirth, 91] explizit entweder durch einen Asterix „*“ oder durch ein Nummern-Zeichen „#“ gekennzeichnet sind. Die Bedeutung der einzelnen Symbole zur Angabe des Gültigkeitsbereichs beschreibt folgende Tabelle:

Postfix	Scope eines globalen Programmobjekts
keines	gültig innerhalb des Moduls
*	gültig innerhalb des lokalen Clusters
#	gültig für alle Cluster

Abbildung 5.3: Postfixe zur Angabe des Scopes von Objekten im Programm

Globale Programmobjekte werden vom Übersetzer (modifiziert für DHS) dem Naming-System durch den Aufruf von RegisterName bekannt gemacht. Ein zusätzlicher Lade- und Link-Vorgang für Programme in DHS entfällt, da Programme sich nach dem Übersetzen bereits im Speicher befinden und somit ausführbar sind.

Importierte globale oder dynamische Objekte aus externen Modulen werden zugegriffen, indem anhand der Namenstabelle beim Übersetzen des Programms die Einträge der Symboltabelle mit Name und logischer Adresse angepaßt werden. Die logischen Adressen zu Namen

modul-externer, aber cluster-interner Objekte werden durch den Aufruf von ResolveName direkt durch den Übersetzer ermittelt und eingesetzt.

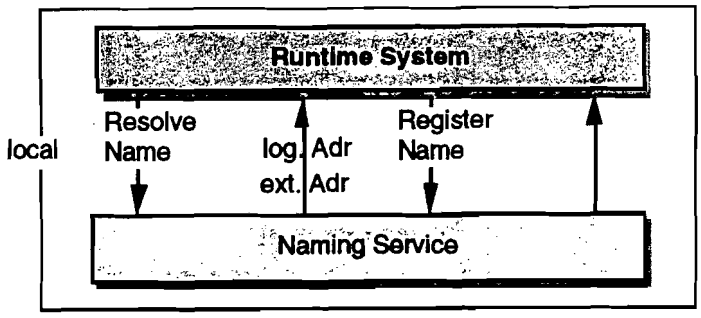

Abbildung 5.4: Aufruf des Naming-Systems beim Übersetzen von Modulen

Für Objekte, die zum Zeitpunkt des Übersetzens dem Naming-System (noch) nicht im Cluster bekannt sind, und für cluster-externe Objekte werden zunächst ungültige, externe Adressen zurückgegeben. Diese werden anstelle der logischen Adressen eingesetzt, die dann nicht beim Übersetzen des Programms, sondern erst beim Zugriff auf die Adresse aufgelöst werden.

5.1.2. Registrierung von Heap-Objekten im logischen Adreßraum zur Laufzeit

Um ein neues dynamisches Objekt im logischen Adreßraum zu erzeugen, wird der Memory Manager mit p := DHS.New(Type) gerufen. Dieser liefert die logische Adresse eines Speicherblocks zurück, der nach Möglichkeit im lokalen Bereich des logischen Adreßraums residiert.

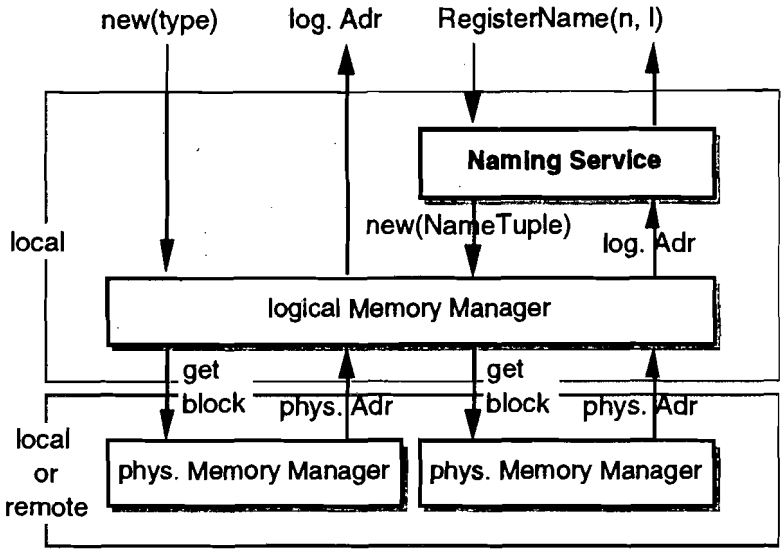

Abbildung 5.5: Aufrufe bei der Registrierung von Namen

Beim Aufruf der Funktion DHS.New wird vom Memory Manager die Mapping-Tabelle des Knotens, auf dem das Objekt erzeugt wurde (lokal), mit den für die MMU notwendigen Mapping-Einträgen versehen. Damit das Speicherobjekt mit einem Namen angesprochen werden kann, muß die Registrierung eines Namens für das neue Objekt durch einen expliziten

Aufruf von RegisterName erfolgen. Falls im lokalen Knoten, in dem das Objekt erzeugt wurde, genügend Speicher vorhanden ist, wird das Memory Management den Namenseintrag auf demselben Knoten erzeugen.

5.2. Naming beim Zugriff von Objekten

Der Zugriff auf modulinterne Objekte geschieht über die Adresse, die beim Übersetzen eingesetzt wird. Externe Objekte oder Objekte, die zur Übersetzungszeit noch nicht bekannt sind, besitzen ungültige Adressen. Auf diese Objekte wird zugegriffen, indem ausgehend vom Namen ihre logische Adresse ermittelt wird. Der Zugriff auf externe Objekte über Namen geschieht entweder direkt durch Aufruf von ResolveName vor der Verwendung der Adresse im Programm, oder indirekt durch das Laufzeitsystem bei Verwendung ungültiger Adressen.

5.2.1. Direkter Zugriff auf Objekte über Namen

Der direkte Zugriff auf benannte Objekte erfolgt, indem das Anwendungsprogramm vor dem Dereferenzieren einer Adresse explizit die Funktion ResolveName ruft, um die logische Adresse des Objekts zum gegebenen Namen zu ermitteln. Das Naming-System beginnt seine internen Tabellen nach der Adresse zum gegebenen Namen zu durchsuchen. Dabei können Teile der Namenstabelle auf externen Knoten plaziert sein.

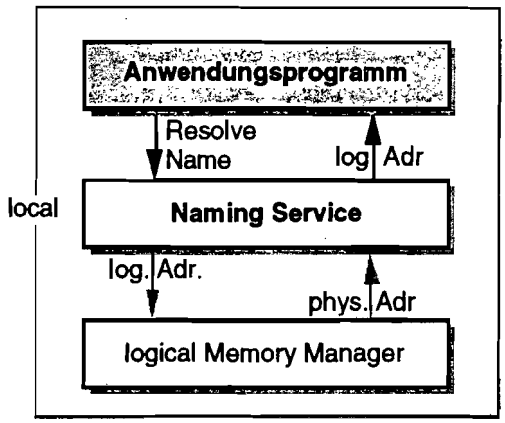

Abbildung 5.6: Direkter Zugriff auf externe Objekte über Namen

Externe Seiten werden beim Zugriff durch das Naming-System auf den lokalen Knoten transportiert. Dies kann als Replikation oder im Original erfolgen. Bei einem erneuten Lesezugriff ist die Speicherseite mit dem gewünschten Tabelleneintrag dann bereits lokal vorhanden. Ein explizites Cachen der Seiten oder der Namenseinträge durch das Naming-System ist nicht notwendig. Das Propagieren von Änderungen in Speicherseiten des Naming-Systems ist allein Aufgabe des Memory Managements.

Der Aufruf ResolveName liefert die logische Adresse, die anschließend für den Zugriff auf das Objekt verwendet wird. Abbildung 5.6 demonstriert diesen Vorgang. Diese Vorgehensweise ist jedoch weniger transparent als der indirekte Zugriff auf Objekte über Namen. Hier muß dem Programmierer bewußt sein, daß auf externe Objekte zugegriffen wird. Dies will man in DHS jedoch vermeiden.

5.2.2. Indirekter Zugriff auf Objekte über Namen

In der Regel wird für die Ermittlung der logischen Adresse zu einem Namen das Naming-System nicht direkt vom Programm gerufen, sondern vom Memory Manager beim Zugriff auf ungültige Adressen. Dabei wird das Memory Management aktiviert, um über die Naming-Funktion NameofAddress den zugehörigen Namen zur ungültigen Adresse zu ermitteln. Der Aufruf von ResolveName liefert dann zum Namen die logische Adresse des zugegriffenen Objekts. Diese wird vom Memory Manager anschließend an der Stelle der ungültigen Adresse eingesetzt und das Anwendungsprogramm mit der Ausführung fortgefahren.

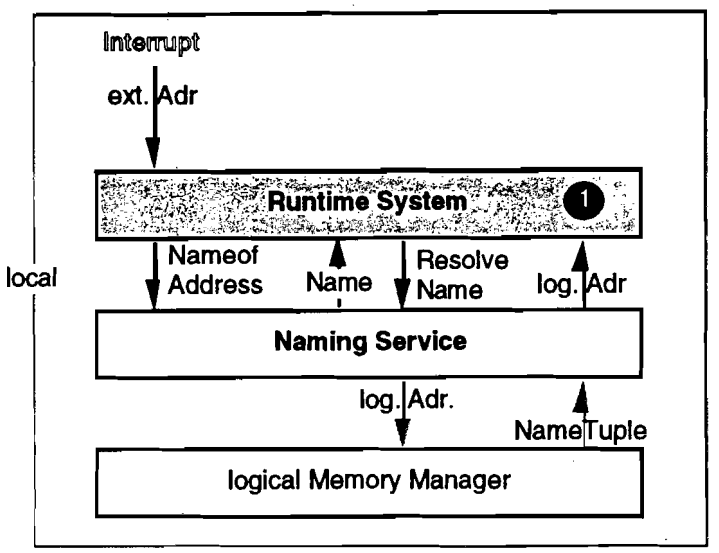

Abbildung 5.7: Indirekter Zugriff auf externe Objekte über Namen

Nachdem die logische Adresse zu einem Namen einmal ermittelt wurde, können alle weiteren Zugriffe direkt über die logische Adresse erfolgen. Das Memory Management verwaltet Referenzen auf Objekte so, daß auch nach einer Änderung der logischen Adresse des Objekts der Zugriff weiterhin über die ursprüngliche Adresse stattfinden kann, solange bis die Garbage Collection die Referenz aktualisiert hat.

5.2.3. Zugriff von Objekten über logische Adressen

Greift ein Modul auf Knoten A über die logische Adresse auf ein Objekt zu und ist das Objekt lokal nicht präsent, d.h. für dessen logische Adresse existiert auf Knoten A kein Mapping-Eintrag im Memory Manager, so löst die MMU beim Zugriff auf das Objekt einen Interrupt aus. Der Interrupt aktiviert den logischen Memory Manager und übergibt die gewünschte logische Adresse. Der Memory Manager hat nun die Aufgabe durch einen Multicast im Cluster den Knoten zu ermitteln, auf dem die gesuchte Adresse vorliegt, und die zugehörige Speicherseite auf den lokalen Knoten zu transportieren[11].

---

[11] Diese beiden Schritte sind zur besseren Veranschaulichung hier getrennt dargestellt.

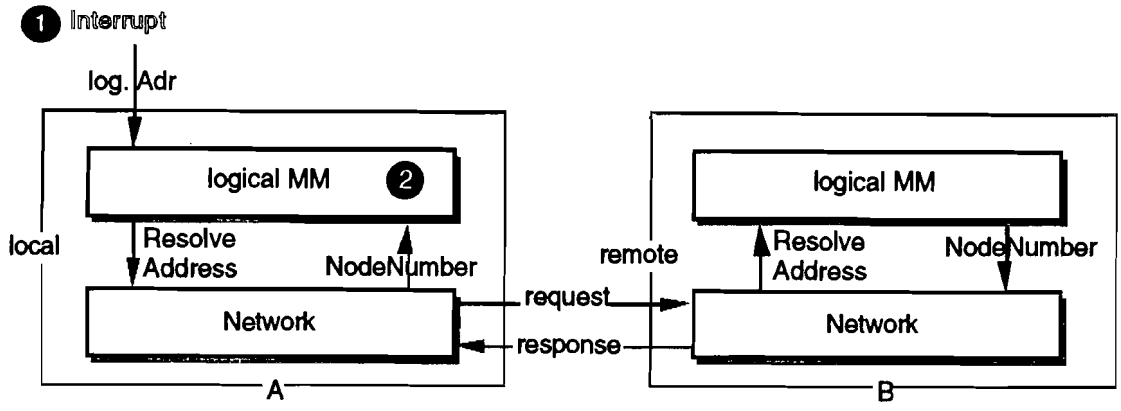

Abbildung 5.8: Ermitteln der Knotennummer eines Speicherobjekts

Nachdem der Memory Manager eines entfernten Knotens B die gesuchte Adresse in seinen Mapping-Tabellen lokalisiert hat, gibt er über das Netzwerk die Knotennummer B an den Knoten A zurück. Der Memory Manager in Knoten A wendet sich nun mit der logischen Adresse direkt an den ermittelten Knoten B, um den gewünschten Speicherblock zu erhalten. Der Speicherblock migriert darauf von Knoten B zum anfordernden Knoten A.

5.3. Naming bei der Migration von Objekten

Der Zugriff auf ein externes Objekt impliziert die Migration des Objektes zum anfordernden Knoten A. Bei der Relokation eines Objektes von Knoten B zu Knoten A innerhalb eines Clusters ist keine Änderung der logischen Adresse l des Eintrags notwendig. Eine Adaption erfordert allerdings die physikalische Adresse p und implizit auch die Knotennummer k. Die vom Memory Manager verwaltete Mapping-Tabelle des Zielknotens A muß bei erfolgreicher Übertragung um das Tupel (l, p) ergänzt und der alte Eintrag im Ursprungsknoten B eliminiert werden. Während der Anpassung der Tabellen ist die Antwort auf eine Anfrage nach der Knotennummer k solange zu verzögern, bis die Einträge wieder konsistent sind.

Nach der Migration eines Objekts sollte bei Gelegenheit auch der Namenseintrag von Knoten B zu Knoten A migrieren. Das ist insofern sinnvoll, da bei Ausfall eines Knotens keine Objekte vom Zugriff über Namen isoliert werden dürfen, nur weil ihre Namenseinträge auf ausgefallenen Knoten liegen und nicht mehr zugängig sind. Die Namenseinträge müssen jedoch nicht unmittelbar zusammen mit den Objekten migrieren. Daher ist die Migration der Namenstabellen von der eigentlichen Migration der Objekte entkoppelt.

Die Migration der Namenstabelleneinträge zum Zielknoten findet in der Regel nicht explizit auf Veranlassung des Naming-Systems oder des Memory Managements statt. Beim Zugriff auf Objekte über den Namen wandert vielmehr bereits vorher der Teil der Namenstabelle, der den gesuchten Namenseintrag eines migrierenden Objektes enthält, wenigstens als Kopie zum Zielknoten, da die Seiten immer lokal zugegriffen werden. Somit ist die oben geforderte Koppelung des Speicherorts des Objekts und des Namenseintrags in der Regel gegeben. Eine explizite Migration der Namenseinträge zu den Knoten auf denen sich die Objekte befinden wird nur dann notwendig, wenn ein Knoten abgeschaltet wird.

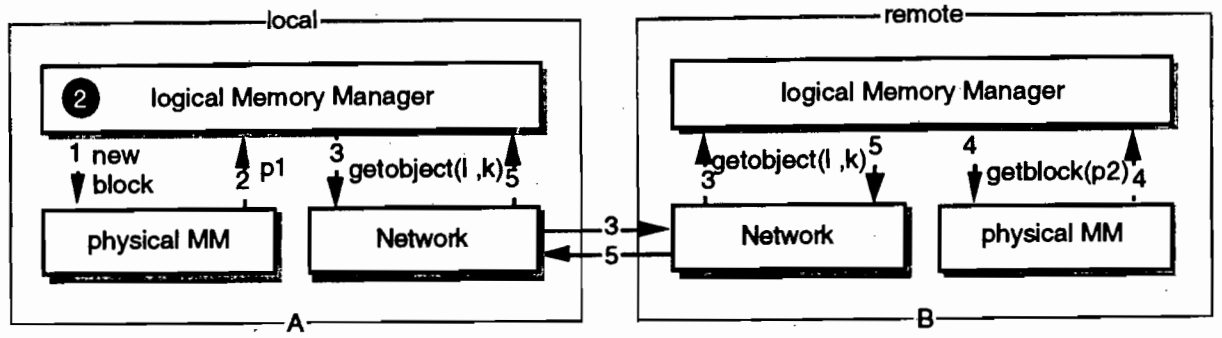

Abbildung 5.9: Migration des Speicherobjekts und Anpassung der Adressen

Wird bei der Migration ein Objekt über die logische Adresse direkt zugegriffen, so ist die Namenstabelle von der Migration nicht betroffen, der Eintrag wandert auch nicht zum Zielknoten des Objekts. Zur Migration eines Namenstabelleneintrags wäre notwendig, zuerst mit dem Aufruf NameofAddress beim Quellknoten den Namen zur logischen Adresse des migrierten Objekts in Erfahrung zu bringen. Daraufhin würde der Namenseintrag in einer Speicherseite zum Zielknoten transportiert. Dies würde jedoch bedeuten, daß bei jedem Zugriff auf ein Objekt, das nicht lokal ist, auch das Naming-System bemüht werden müßte. Da jedoch nach dem Zugriff über logische Adressen kein Zugriff über den Namen mehr zu erwarten ist, erscheint es nicht notwendig, den Namenseintrag beim Zugriff über Adressen lokal zu halten. Der Aufruf des Naming-Systems kann entfallen.

5.4. Verschieben von Objekten im logischen Adreßraum

Als Objektbezeichner in DHS werden logische Adressen benutzt. Adressen können sich im Gegensatz zu Bezeichnern jedoch im Laufe der Zeit ändern. Eine logische Adresse kann beispielsweise zu verschiedenen Zeitpunkten unterschiedlichen Objekten zugeordnet sein:

- Z.B., wenn das vorangegangene Objekt gelöscht wurde, und an derselben Adresse nun ein neues Objekt zu finden ist,
- oder wenn ein Objekt an einen anderen Ort im logischen Adreßraum verschoben wurde.

Referenzen auf gelöschte Objekte können in DHS nicht auftreten, da Objekte nicht explizit, sondern von der Garbage Collection[12] gelöscht werden, wenn die Objekte nicht mehr referenziert werden können. Beim Verschieben von Objekten im logischen Adreßraum hingegen ändert sich die logische Adresse. Objekte müssen im logischen Adreßraum reloziert werden,

- wenn durch die Garbage Collection Löcher im logischen Adreßraum gefüllt werden, um den Speicher zu kompaktifizieren.
- wenn die Größe eines Objektes verändert wird und nach der Änderung nicht mehr an der ursprünglichen Stelle im logischen Adreßraum Platz findet.

---

[12] Das vorgestellte DSM-System wird regelmäßig von einer Garbage Collection durchforstet.

- wenn sich bei der Migration eines Objekts auf einen anderen Knoten in einer logischen Speicherseite mehrere Objekte befinden. In diesem Fall wird das „Ausräumen" von Seiten notwendig, damit die übrigen Objekte in der betroffenen Speicherseite nicht ebenfalls migrieren.

In den hier angeführten Situationen ändert sich die logische Adresse nicht jedoch notwendigerweise die physikalische Adresse eines Objekts (vgl. [Traub, 94]) Bei Verschiebungen müssen die Namens- und die Mapping-Tabellen der involvierten Knoten angepaßt werden. Das Naming-System stellt für die Adaption der Naming-Tabellen *keine* eigene Funktion UpdateAddress zur Verfügung. Anpassungen von logischen Adressen erledigt die Garbage Collection des Memory Managements. Die Einträge in den Mapping-Tabellen der Knoten werden bei Änderung der physikalischen Adresse ebenfalls vom Memory Management angepaßt.

Bei der Verschiebung von Objekten im logischen Adreßraum genügt es aber nicht, den Update logischer Adressen im Naming-System zu propagieren. Auch alle Referenzen in anderen Objekten auf die ursprüngliche logische Adresse müssen geändert werden. Diese Aufgabe übernimmt die Garbage Collection, die alle Strukturen im Speicher eines jeden Knotens zyklisch durchläuft.

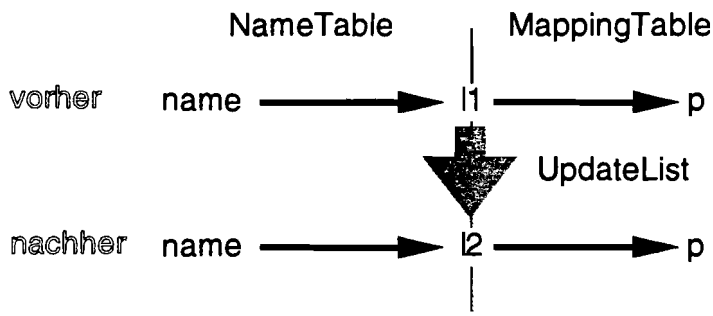

Abbildung 5.10: Anpassung der Tabelleneinträge beim Verschieben von Objekten im logischen Adreßraum

Bis die Änderung bei allen Knoten propagiert wurde, besteht die Gefahr, daß ein Objekt über eine noch nicht angepaßte Referenz mit der alten Adresse auf ein Objekt zugreift. Um dies zu vermeiden, wird vom Memory Management die geänderte logische Adresse für ungültig proklamiert und zusammen mit der neuen Adresse in eine interne Tabelle UpdateList aufgenommen. Erst wenn das Propagieren des geänderten Eintrags abgeschlossen ist und von der Garbage Collection mit Hilfe dieser Tabelle alle Referenzen angepaßt wurden, wird der Eintrag aus UpdateList gelöscht.

5.5. Naming beim Löschen von Objekten

Ein Objekt in DHS wird nicht explizit durch einen Systemaufruf, sondern implizit durch die Garbage Collection gelöscht. Diese entfernt ein Objekt genau dann, wenn

- keine Referenz auf das Objekt mehr besteht und
- somit als Referenz auch der Namenseintrag für das Objekt gelöscht wurde.

Solange ein Name existiert, wird das Objekt nicht gelöscht, auch wenn keine weiteren Referenzen mehr vorhanden sind. Der Aufruf für das Löschen eines Namens zu einem Objekt erfolgt durch den Aufruf von DeleteName.

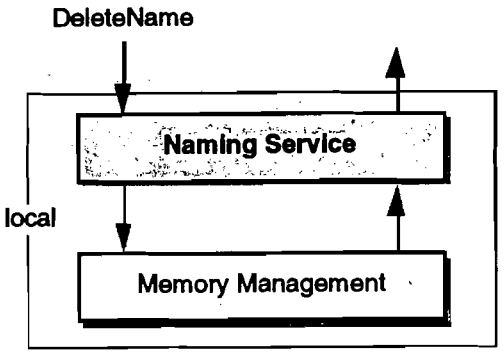

Abbildung 5.11: Löschen eines Namenseintrags

Findet der Garbage Collector auf seiner Suche nach Referenzen für ein Objekt auch in der Namestabelle keine Referenz mehr, so wird das Objekt entfernt. Beim Löschen von Einträgen wird eine Speicherseite der Namenstabelle verändert. Das Memory Management muß also alle eventuell vorhandenen Replikationen dieser Seite als ungültig markieren. Dieser Vorgang bleibt jedoch für das Naming transparent.

5.6. Naming bei cluster-übergreifenden Speicheroperationen

Die Verwaltung globaler Namen erfordert den Zugriff auf Namenseinträge in externen Clustern. Dieser kann erfolgen, durch

- die lokale Bearbeitung der Strukturen der Namensverwaltung, wobei die zugehörigen Speicherseiten vom externen Cluster zum lokalen Cluster übertragen werden.
- das Übertragen der Ausführung zu dem Cluster, in dem die zu bearbeitenden Namenseinträge vorhanden sind. Hierzu müssen über den DSM-Mechanismus RPC-Aufrufe realisiert werden.

Die erste der beiden aufgezeigten Möglichkeiten ist in DHS die natürlichste. Sie hat allerdings den Nachteil, daß über die Cluster-Grenzen hinweg große Mengen von Speicher zu bewegen sind. Außerdem müssen alle Referenzen, die in den migrierten Speicherblöcken vorhanden sind, angepaßt werden. Daher ist es in der Regel sinnvoller Namensanfragen auf entfernte Cluster zu übertragen.

Die Mechanismen zur globalen Namensverwaltung und insbesondere die Verteilung und Lokation globaler Namenseinträge auf Cluster sollen in diesem Artikel nicht weiter behandelt werden. Der interessierte Leser sei an dieser Stelle auf [Lupper, 95] verwiesen.

6. Realisierung der Namensoperationen in DHS

Die Namensverwaltung übernimmt alle Aufgaben, die für die Abbildung von Namen auf logische Adressen notwendig sind. Die lokalen Funktionen des Naming-Systems für DHS lassen sich grob in primitive und makroskopische Namensfunktionen untergliedern. Die primitiven Funktionen behandeln nur einfache Namen und sind im Modul Simple Names gruppiert. Zu den Makrofunktionen, d.h. zu den Naming-Funktionen, die sich ihrerseits aus primitiven Funktionen zusammensetzen, gehören die Funktionen zur Handhabung strukturierter Namen. Diese Funktionen sind im Modul Structured Names vereint. Die Funktionen dieser beiden Schichten sind völlig ortstransparent; das Naming-System ist sich der Verteilung der Einträge innerhalb des Clusters nicht bewußt.

Die Auflösung von Namen über Cluster-Grenzen hinweg kann jedoch für das Naming nicht mehr ortstransparent geschehen. Daher gibt es ein zusätzliches Modul Global Names, das die Verwaltung globaler Namen übernimmt und dazu die Funktionen des Moduls Structured Names verwendet [Lupper, 95]. Für den Anwender muß die Namensvergabe dennoch weitgehend ortstransparent ablaufen. Die Anwendungsschnittstelle des Moduls Global Names wird lediglich mit Angaben über den Gültigkeitsbereich der Namen versorgt (vgl. Abbildung 5.3). Die Unterteilung des Naming-Systems in Module zeigt Abbildung 6.1.

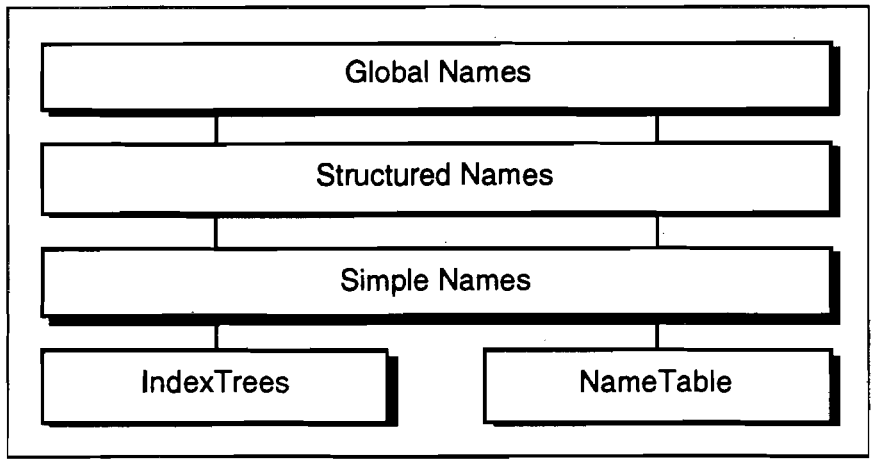

Abbildung 6.1: Schichten des Naming-Systems

Um das Netzwerk von Zugriffen zu entlasten und die Zeit für die Auflösung von Namen zu reduzieren, ist es notwendig, Namenseinträge zu replizieren und zwischenzuspeichern. Innerhalb eines Clusters sind die Naming-Funktionen jedoch völlig ortstransparent. Daher müssen keine Funktionen zur passiven Replikation (Caching) in den unteren Schichten des Naming-Systems enthalten sein. Das Cachen von Namenseinträgen geschieht im Cluster auf der Basis von Speicherseiten, in denen sich die zu einem Knoten gehörigen Einträge gruppieren. Speicherseiten von Objekten und Namenseinträgen, bzw. deren Replikationen werden sich in den Knoten ansammeln, auf denen sie zugegriffen werden. Insbesondere bei

Wurzeleinträgen ist das Replizieren vorteilhaft, da sie sehr häufig und von verschiedenen Knoten zugegriffen werden.

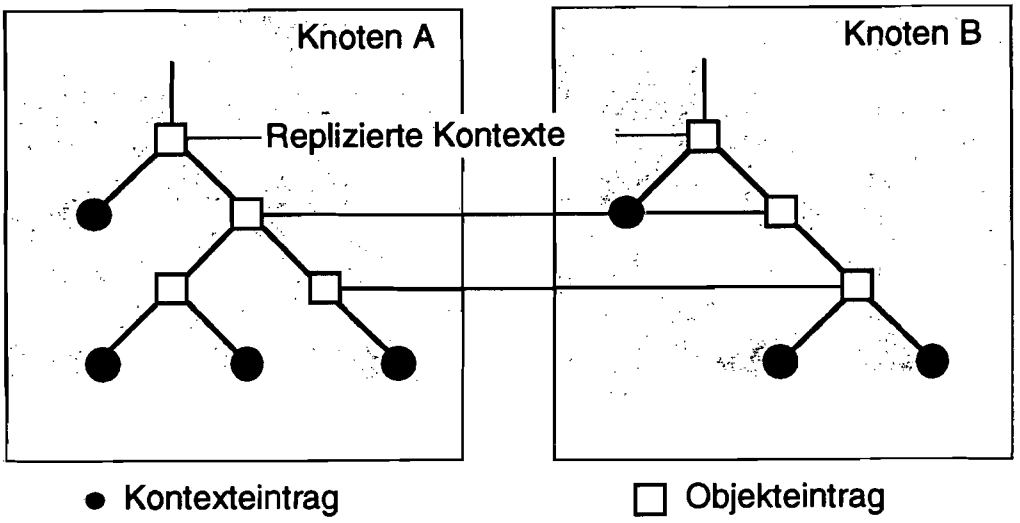

Abbildung 6.2: Replikation von Namenseinträgen auf den zugreifenden Knoten

Ohne Replikation müßten die Seiten mit den Wurzeleinträgen bei jedem Zugriff eines anderen Knotens wandern. Das Zulassen von Replikationen verschärft die Problematik der Konsistenz von Speicherseiten für den Memory Manager jedoch erheblich. Auf der anderen Seite ändern sich gerade die häufig gelesenen Wurzeleinträge sehr selten, so daß sie für die Replikation besonders geeignet sind. Die Replikation von Namenseinträgen zwischen Clustern muß durch das Naming-System übernommen werden. Die dazu notwendigen Funktionen sind im Modul Global Names vorhanden.

Für die Realisierung der cluster-internen Naming-Funktionen ist es ferner wichtig, die Anzahl der verschiedenen zugegriffenen Speicherseiten zu reduzieren. Das ist besonders von Bedeutung, wenn die Seiten über mehrere Knoten verteilt sind.

Zu beachten ist ferner, daß auf Namensanfragen (z.B. de.*.informatik.*) unter Umständen mehrere Einträge zutreffen können. Die hier dargestellten Naming-Funktionen liefern jedoch der Einfachheit wegen immer nur einen Eintrag zurück. In der konkreten Implementierung wurden die Schnittstellen des Naming-Systems so erweitert, daß durch einen speziellen Funktionsaufruf weitere Einträge zu einer Anfrage zurückgeliefert werden können.

Für den Zugriff auf Objekte in DHS ist es neben der Namensauflösung notwendig, logische Adressen im Cluster auf Knotennummern und physikalische Adressen abzubilden. Die Abbildung von logischen auf physikalische Adressen übernimmt das Memory Management und muß nicht vom Naming-System durchgeführt werden. Realisiert wird diese Abbildung durch Einträge in den Mapping-Tabellen der MMUs in den Knoten, deren Verwaltung vom Memory Manager übernommen wird.

6.1. Definition der grundlegenden Strukturen für das Naming-System

Bevor wir auf die einzelnen Funktionen des Naming-Systems eingehen, wollen wir die verwendeten Typen und Datenstrukturen vorstellen. Diese sind recht einfach ausgelegt und sehen wie folgt aus:

```
ClusterNumber = LONGINT;
LogicalAddress = POINTER TO Objekt;
ResultType = (error, no match, full match, partial match, cluster not alive, cluster alive);
LocationType = (local, external, replicated, cached);
EntryType = (object, context);
NameTupleSet = SET OF NameTuple;
SimpleName = ARRAY OF CHAR;
StructName = ARRAY OF CHAR;
```

Abbildung 6.3: Die einfachen Typen des Naming-Systems

Die Typen ClusterNumber und LogicalAddress sind jeweils skalare Datentypen mit 32 Bit Länge. Die Kardinalität der logischen Adressen sollte später auf 2^{64} (64 Bit) erhöht werden.

$$StructName = SimpleName\{"."SimpleName\}^{*}.$$

Ein einfacher Name SimpleName ist lediglich eine Zeichenkette. Die Datenstruktur für einen strukturierten Namen bildet bei der Übergabe zunächst nur ein Feld von Zeichen. Die einzelnen Namensbestandteile SimpleName des strukturierten Namens StructName sind lediglich durch einen Punkt „." separiert. Intern wird vom Naming-System ein derartiger Name komponentenweise als Folge von Einträgen des Typs SimpleName abgelegt.

```
NameTupleAddress = POINTER TO NameTuple;
NameTuple =   RECORD
                 Name: SimpleName;
                 FAddress: NameTupleAddress ;
                 OAddress: LogicalAddress;
                 Entry: EntryType;
              END;
GlobalNameTuple =  RECORD
                      (NameTuple)
                      Cluster: ClusterNumber;
                      Location: LocationType;
                   END;
```

Abbildung 6.4: Struktur eines Namenstuples

Namenseinträge werden intern als NameTuple mit der logischen Adresse des Ablageorts des Objekts als Objektbezeichner OAddress, dem Bezeichner des übergeordneten Eintrags FAddress und dem Namensbestandteil Name verwaltet. Die einzelnen Einträge werden über das Feld FAddress miteinander verbunden. Das Feld EntryType gibt an, ob der Eintrag ein Objekt oder einen Kontext bezeichnet.

Für die cluster-übergreifende (globale) Verwaltung von Namenseinträgen wird der Typ NameTuple zum Typ GlobalNameTuple erweitert. Das Feld Cluster gibt dann die Nummer des Clusters an, in dem der gefundene Eintrag verwaltet wird. In Location wird abgelegt, ob es sich um einen lokalen, externen, replizierten oder zwischengespeicherten Eintrag handelt.

LAddress	Name	OAddress	FAddress
$1	uni-ulm	NIL	$0
$2	info	NIL	$1
$3	vs	$23	$2
$4	Daimler	NIL	$0
$5	Mercedes	$87	$4
$6	AEG	$45	$4
$7	math	$57	$1
$8	med	$78	$1
$9	dbis	$C4	$2
$7	gmd	$A6	$0

Abbildung 6.5: Interne Repräsentation der hierarchischen Namensstruktur aus Abbildung 3.1

Strukturierte Namen werden als verkettete Folge von Namenstupeln abgelegt. Die Verkettung wird über FAddress hergestellt. Abbildung 7.5 zeigt, daß z.B. der Namen uni-ulm.info.vs durch die Tupel [$1, (uni-ulm, NIL, $0)], [$2, (info, NIL, $1) und [$3, (vs, $23, $2) repräsentiert wird. Die Relation wird über FAddress und dem Speicherort der Namenstupels hergestellt. Synonyme für Objekte können durch mehrfache Einträge in dieser Tabelle erzeugt werden. Außerdem sind auf diese Art leicht Querverweise (Links) herzustellen (gmd, $0, $7).

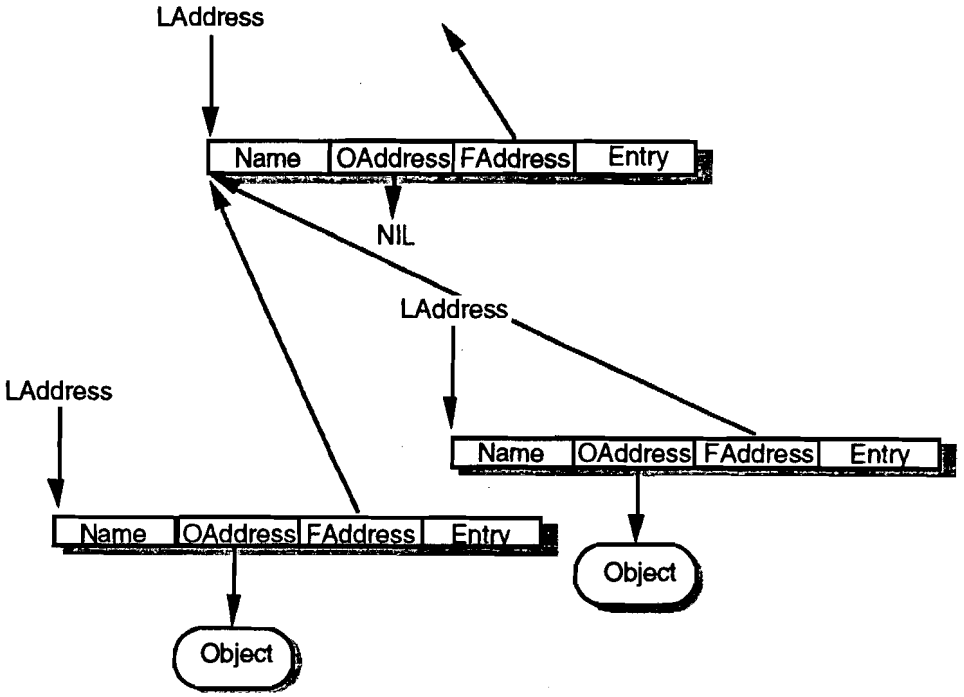

Abbildung 6.6: Die Verkettung der Namenstupel auf dem Heap über das Feld FAddress

Die Namenstupel werden dynamisch auf dem Heap angelegt und über Indexbäume, die nach den jeweiligen Komponenten der Namenstupel geordnet sind, zugegriffen (vgl. Abbildung 6.6). Die Indexbäume erlauben die flexible und effiziente Speicherung der Namenstupel. Namen werden dabei stets nur einmal im Heap des DHS abgelegt, auch wenn sie unterschiedliche Kontexte oder Objekte bezeichnen. Die Suchbäume erlauben außerdem das schnelle Durchsuchen von Tupeln nach verschiedenen Kriterien. Dabei müssen nur wenige Speicherseiten angefaßt werden. Die gewählte Verwaltungsstruktur für Namenstupel bewirkt, daß die Struktur der Namen durch den Algorithmus zum Durchlaufen der Namenstupel und nicht durch die Datenstruktur zur Aufnahme der Tupel festgelegt wird.

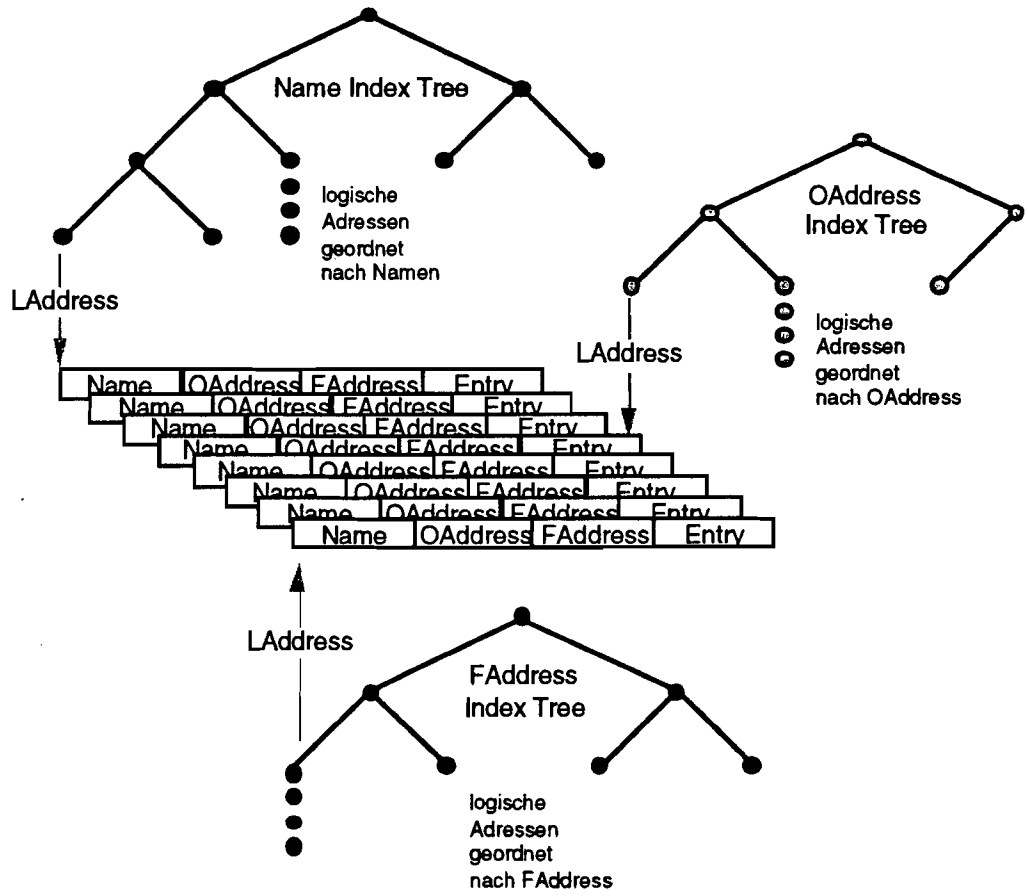

Abbildung 6.7: Indexbäume für den Zugriff auf die Namenstupel

Durch die feine Granularität der Namenseinträge können sich mehrere Namenstupel in denselben Speicherseiten gruppieren, die auf den Knoten wandern, der sie zugreift. Somit ist zu erwarten, daß sich die Seitenzugriffe mit der Zeit reduzieren. Eine weitere Beschleunigung des Zugriffs kann durch die Umordnung der Indexbäume unter Berücksichtigung der Zugriffshäufigkeit der Namenseinträge geschehen. Als Nachteil der feingranularen Namenstupel ist eine starke Fragmentierung des DHS-Speichers zu nennen, die durch die Speicherkompaktifizierung des DHS behoben werden muß.

6.2. Die primitiven Namensfunktionen

Aufgabe dieser Ebene des Naming-Systems ist die Verwaltung einfacher Namen zu logischen Adressen von Objekten. Die Registrierung geschieht durch die primitiven Funktionen immer lokal innerhalb des Clusters, in dem das Objekt angelegt wurde. Der Name ist innerhalb eines Clusters und eines Kontexts eindeutig.

Beim Ermitteln der logischen Adresse zu einem gegebenen Namen innerhalb des Clusters ist dem Naming-System nicht bekannt, welche Namenseinträge in lokalen Speicherseiten vorliegen und welche nicht. Greift das Naming-System auf einen Eintrag zu, der nicht lokal vorhanden ist, so sorgt das Memory Management dafür, daß die entsprechenden Speicherseiten von den anderen Knoten des Clusters herbeigeschafft werden. Das Naming-System besitzt keine Kenntnis über die Verteilung der Namenseinträge.

Sollen andere Knoten außerhalb dieses Cluster auf Objekte zugreifen, so müssen die zuvor Einträge exportiert werden. Für das Exportieren eines Namens zu einem Objekt außerhalb des Clusters ist es notwendig, daß vom System zusätzlich eindeutige Cluster-Nummern vergeben und verwaltet werden. Die Cluster-Grenzen sind dann für das Naming-System nicht mehr transparent.

Die primitiven Funktionen für die Namensverwaltung können wie folgt angegeben werden:

PROCEDURE RegisterSimpleName(NameEntry: NameTuple): Result;

Dieser Funktion wird ein NameTuple übergeben, das den Namen des Objekts, die logische Adresse und die Kennung des übergeordneten Kontexts enthält. Um die Eindeutigkeit zu gewährleisten, muß anhand der Funktion ResolveSimpleName zuvor nach dem neuen Eintrag gesucht werden. Ist die Liste der gefundenen Namenseinträge leer, so wird der neue Namenseintrag angelegt und in die Indexbäume aufgenommen. Tritt ein Konflikt durch doppelte Namen oder belegte Adressen auf, so muß das Objekt unter einem anderen Namen registriert werden. Der Name wird nur innerhalb des Clusters registriert. Eine nachträgliche Ausdehnung des Gültigkeitsbereichs eines Namen über Cluster-Grenzen hinaus ist nur möglich, sofern keine Konflikte mit existierenden globalen Namen auftreten.

```
PROCEDURE  RegisterSimpleName(NameEntry: NameTuple): Result;
VAR I: Index;
BEGIN
   RegisterSimpleName := error;
   IF ResolveSimpleName(NameEntry) = {} THEN
      WITH NameEntry DO
         I := InsertNameTable(NameEntry);
         RegisterSimpleName:= InsertNameIndex(Name, I);
         RegisterSimpleName:= InsertObjectIndex(OAddress, I);
         RegisterSimpleName:= InsertFatherIndex(FAddress, I);
      END
   END
END RegisterSimpleName;
```

Abbildung 6.8: Registrierung eines einfachen Namens

PROCEDURE ResolveSimpleName(VAR NameEntry:NameTuple):NameTupleSet;

Die Funktion ermittelt die logische Adresse zu einem gegebenen flachen Namen. ResolveSimpleName durchsucht dazu den Namensindexbaum und die Adreßindexbäume nach dem gesuchten Eintrag. Werden in den Bäumen mehrere passende Einträge ermittelt, so wird die Schnittmenge gebildet und als Ergebnis zurückgegeben. Solange bei der Suche nach Namen keine Wildcards zugelassen sind, ist das Ergebnis immer eindeutig. Die Suche terminiert beim ersten Match, da die abgebildete Funktion ResolveSimpleName weder Replikate von Namenseinträgen noch Synonyme vorsieht.

```
PROCEDURE ResolveSimpleName(NameEntry: NameTuple): NameTupleSet;
BEGIN
   I1, I2, I3: NameTupleSet;
   WITH NameEntry DO
      I1 := GetNameIndex(Name);
      I2 := GetObjectIndex(LAddress);
      I3 := GetFatherIndex(FAddress);
   END;
   ResolveSimpleName := I1 AND I2 AND I3;
END ResolveSimpleName;
```

Abbildung 6.9: Auflösung eines einfachen Namens

Zu beachten ist, daß bei der Suche nach einfachen Namen nicht alle Felder des Namenstupels voll spezifiziert sein müssen. FAddress 0 bedeutet z.B eine beliebige Vorgängeradresse, der leere Name "" symbolisiert einen beliebigen Namen. Die Indexfunktionen liefern darauf einen Joker zurück, der beim Schnitt der Indexmengen berücksichtigt wird.

PROCEDURE DeleteSimpleName(NameEntry: NameTuple): Result;

Mit Hilfe von DeleteSimpleName wird ein Namenseintrag gelöscht. Das Löschen geschieht, indem alle Einträge aus den Indexbäumen entfernt werden, die auf Namenseinträge verweisen, die das übergebene Tupel enthalten.

```
PROCEDURE DeleteSimpleName(NameEntry: NameTuple): Result;
VAR I: NameTupleSet;
BEGIN
   I := ResolveSimpleName(NameEntry);
   WITH NameEntry DO
      DeleteSimpleName := DeleteNameIndex(Name,I);
      DeleteSimpleName := DeleteObjectIndex(LAddress, I);
      DeleteSimpleName := DeleteFatherIndex(FAddress,I);
   END;
   { garbage collection !! }
END DeleteSimpleName;
```

Abbildung 6.10: Löschen eines einfachen Namens

Da in DHS eine Garbage Collection enthalten ist, genügt das Löschen aus den Indexbäumen, um den Eintrag zu entfernen. Er kann dann vom der Namensverwaltung nicht mehr zugegriffen werden und wird bei Gelegenheit von der Garbage Collection aus dem Speicher entfernt, da keine Referenzen mehr auf den Eintrag existieren.

6.3. Makroskopische Operationen auf dem Namens- und Adreßraum

Aus den primitiven Operationen auf dem Namensraum lassen sich komplexere Operationen zusammensetzen. Diese Operationen müssen in der Regel atomar ablaufen. Durch passende Kombination der primitiven Operationen kann in vielen Fällen auch ohne eine atomare Ausführung bereits die Konsistenz gewährleistet werden. Dort wo Wettstreitsituationen auftreten können, sorgt die Transaktionsverwaltung in DHS mit dem Aufruf Transaction für die atomare Ausführung des Oberon-Commands. Tritt bei der Ausführung eines atomaren Kommandos ein Konflikt auf, so wird es zurückgesetzt.

PROCEDURE RegisterName(NameEntry: NameTuple): Result;

Dieser Funktion wird ein Namenstupel übergeben, das den strukturierten Namen und die logische Adresse des Objekts enthält. Falls keine logische Adresse angegeben ist, ermittelt die Funktion eine freie Adresse für das Objekt und registriert den zugehörigen strukturierten Namen im Cluster unter dieser Adresse.

```
PROCEDURE RegisterName(NameEntry: NameTuple): Result;
VAR N: NameTuple;
      I: Index;
BEGIN
   found := done;
   RegisterName := done;
   N.FAddress := GetContext(NameEntry.Name);
   WHILE NameEntry.Name = structured_Name DO
      N.Name := LEFTPART(NameEntry.Name);
      NameEntry.Name := NameEntry.Name - N.Name;
      I := ResolveSimpleName(N);
      IF found AND I <> {} THEN
         N.FAddress := I^.LAddress;
      ELSE
         N.LAddress := NewOID;
         found := RegisterSimpleName(N);
         N.FAddress := N.LAddress ;
         found := FALSE;
      END
   END;
   N.Name := NameEntry.Name;
   N.LAddress := LAddress;
   RegisterName:= RegisterSimpleName(N);
END RegisterName;
```

Abbildung 6.11: Registrierung eines strukturierten Namens

Um die Eindeutigkeit der Namenskomponenten zu prüfen, wird die Funktion ResolveSimpleName verwendet. Tritt ein Konflikt durch einen doppelten Namen, so muß das Objekt unter einem anderen Namen registriert werden. Eine globale Ausdeh-

nung des Gültigkeitsbereichs ist nachträglich möglich. Tritt hierbei jedoch ein Konflikt auf, so kann die globale Registrierung nur unter einem Synonym erfolgen.

PROCEDURE ResolveName(Name: StructName): NameTuple;

Die Funktion ermittelt die logische Adresse zu einem gegebenen Namen. ResolveName durchsucht dazu die cluster-internen Verzeichnisse nach dem strukturierten Namen. Wurde im Cluster kein lokaler Eintrag für einen Namen gefunden, so kann im Anschluß die Suche über den Cluster hinaus ausgedehnt werden.

```
PROCEDURE ResolveName(Name: StructName): NameTuple;
VAR N: NameTuple;
      I: NameTupleSet;
BEGIN
   N.FAddress := GetContext(Name);
   WHILE Name = structured_Name DO
      N.Name := LEFTPART(Name);
      Name := Name - N.Name;
      I := ResolveSimpleName(N);
      IF I <> {} THEN
         N.FAddress := I^.LAddress;
      ELSE
         Exit;
      END
   END;
   N.Name := Name;
   ResolveName:= ResolveSimpleName(N);
END ResolveName;
```

Abbildung 6.12: Auflösung eines strukturierten Namens

Zu beachten ist, die hier abgebildete Funktion ResolveName keine alternativen Pfade (Links), Synonyme und Wildcards bei der Suche berücksichtigt. Zurückgegeben wird von ResolveName entweder eine gültige oder die ungültige Adresse 0;

PROCEDURE DeleteName(Name: StructName): Result;

Mit Hilfe von DeleteName wird ein Namenseintrag mit dem strukturierten Namen Name gelöscht. Bezeichnet der Namen ein Blatt, so wird es gelöscht. Handelt es sich um einen Kontext, so wird er nur dann entfernt, wenn keine Nachfolger mehr vorhanden sind. Werden durch das Löschen übergeordnete Kontexte ebenso überflüssig, so werden diese rekursiv entfernt, bis noch weitere Einträge vorhanden sind.

PROCEDURE NameofAddress(Object: LogicalAddress): NameTuple;

Diese Prozedur liefert zu einer gegebenen Objektadresse den zugehörigen strukturierten Namen, der die Kanten des Pfades bezeichnet, die zum Objektbezeichner führen. Alternative Pfade sind in der ersten Version nicht vorgesehen.

```
PROCEDURE NameofAddress(Object: LogicalAddress): NameTuple;
VAR N: NameTuple;
      I: NameTupleSet;
BEGIN
   N.LAddress := Object; N.Name := ""; N.FAddress := 0;
   I := ResolveSimpleName(N);
   NameofAddress := I^.Name;
   N.LAddress := I^.FAddress
   WHILE I^.FAddress <> Root DO
      I := ResolveSimpleName(N) ;
      IF I <> {} THEN
         N.LAddress := I^.FAddress;
         NameofAddress := I^.Name + NameofAddress ;
      ELSE
         Exit;
      END
   END;
END NameofAddress;
```

Abbildung 6.13: Ermitteln des Namens zu einem gegebenen Bezeichner

Ausgehend von Object werden der Objektbezeichner des übergeordneten Kontexts ermittelt und die Namensteile sukzessive zusammengebaut.

PROCEDURE UpdateName(OldName, NewName: StructName): Result;

Die Funktion weist einem Namenseintrag einen anderen Namen zu. Die zugehörigen primitiven Funktionen sind ResolveName, DeleteName, RegisterName. Diese aus atomaren Funktionen zusammengesetzte Funktion muß als Transaktion ablaufen.

```
PROCEDURE UpdateName:(OldName, NewName: StructName): Result;
VAR HelpTuple: NameTuple;
BEGIN
   UpdateName:= noErr;
   HelpTuple:= ResolveName(OldName);
   UpdateName:= DeleteName(OldName);
   HelpTuple.Name := NewName;
   UpdateName:= RegisterName(HelpTuple);
END UpdateName;
```

Abbildung 6.14: Anpassung eines strukturierten Namens

Abschließend sei nochmals darauf hingewiesen, daß die abgebildeten Funktionen stark vereinfacht dargestellt wurden, um dem Leser nicht den Blick auf das Wesentliche durch Details zu trüben. Dieser Vereinfachung unterliegt auch die Darstellung der Behandlung von Wildcards und Synonymen, auf die an dieser Stelle verzichtet wurde, auch wenn sie ein integraler Bestandteil der implementierten Namensverwaltung sind.

Die dargestellten Systemaufrufe erlauben lediglich die Rückgabe eines einzigen Namenstupels auf eine Anfrage. In vielen Fällen ist es jedoch wünschenswert, bei der Suche nach Objekten Wildcards verwenden zu können, z.B. um dem Anwender eine Auswahl von vorhandenen Objekten mit bestimmten Eigenschaften anzubieten. Der aktuelle Entwurf ermöglicht Synonyme für Namen nur in der Form, daß für ein Objekt ein alternativer Name vergeben werden kann. Weiterreichende Funktionen z.B. zum Auffinden aller Synonyme eines Objekts oder zum Löschen aller Synonyme sind zur Zeit noch nicht vorgesehen.

7. Zusammenfassung und Ausblick

Die gegenwärtige Version des Naming-Systems stellt die grundlegenden Funktionen zur Verwaltung von Namen in DSM-Systemen zur Verfügung. Dabei wurde besonders Wert darauf gelegt, die positiven Eigenschaften des DHS zu nutzen. Das vorliegende Naming-System ist völlig ortstransparent und verwaltet somit keine expliziten Replikationen von Namenseinträgen, wie sie zur Steigerung der Zugriffsgeschwindigkeit notwendig sind. Replikation findet transparent auf Basis von Speicherseiten statt und wird von DHS verwaltet. Das Naming-System ist in keiner Weise mit der Konsistenz replizierter Namenseinträge befaßt.

Da jeder Zugriff auf eine Speicherseite Netzwerk-Transaktionen nach sich ziehen kann, ist darauf zu achten, die Anzahl der zugegriffenen Speicherseiten zu reduzieren. Dabei wird ausgenützt, daß lokal verwendete Namenseinträge mit der Zeit vom Memory Manager zusammen mit anderen lokal benötigten Einträgen in denselben Speicherseiten plaziert werden.

Im Rahmen des Plurix Projekts wurde ein typensicherer Compiler (Oberon) so modifiziert, daß er die für das Übersetzen notwendigen Informationen der Symboltabellen über das Naming-System bezieht. Aus Modulen exportierte Konstanten, Typen, Variablen und Prozeduren werden vom Compiler über die Namensverwaltung bekannt gemacht, und können somit von anderen Modulen zugegriffen werden.

Die in Kapitel 6 beschriebenen Funktionen laufen alle innerhalb eines Clusters ab. Die Verwaltung cluster-übergreifender Nameseinträge wurde in diesem Artikel nicht behandelt, obgleich diese einen Schwerpunkt der implementierten Namensverwaltung darstellt. Für die globale Namensverwaltung wurde ein dynamisches Verfahren entwickelt, das die automatische Generierung und Plazierung globaler Kontexte übernimmt. Die globalen Operationen der Namensverwaltung erlauben zudem die transparente Migration von Objekten von Cluster zu Cluster. Der Leser sei hierzu nochmals auf [Lupper, 95] verwiesen.

Ein wesentlicher Aspekt in verteilten Systemen ist der Schutz von Objekten vor unberechtigtem Zugriff. Das gegenwärtig implementierte Naming-System enthält noch keine Zugangskontrolle. In der nächsten Versionen wird die Namensverwaltung jedoch einen Authentisierungsmechanismus enthalten. Die Entwicklung des vorliegenden Naming-Systems wird basierend auf Erfahrungen aus der ersten Version fortgeführt und soll letztendlich attributierte Namen cluster-übergreifend verwalten.

8. Literatur

[Comer, 89] *Comer, Douglas E. and Peterson, Larry L.:* „Understanding Naming in Distributed Systems". Department of Computer Science, University of Arizona, Tucson, AZ, Technical Report 85-27a, February 1987.

[Fujinami , 92] *Fujinami, Nobuhisa and Yasuhiko Yokote*: „Naming and Addressing of Objects without Unique Identifiers", Sony Computer Science Laboratory Inc., IEEE Computer Society, Proceedings of the 12th ICDCS, Yokohama, Japan, June 9–12, 1992, pp. 581–588.

[Goscinsky, 91] *Goscinsky, A.:* „Distributed Operating Systems, The Logical Design", Addison-Weseley, 1991.

[Lupper, 95] *Lupper, Alfred:* „Dynamische globale Namensverwaltung in verteilten Betriebssystemen", Universität Ulm, Dissertation, 1995, in Vorbereitung.

[Mattes, 91] *Mattes, B.:* „Namens- und Objektverwaltung in verteilten Systemen", Dissertation, Uni Karlsruhe 1991.

[Mockapetris, 87] *Mockapetris, P. V:* „Domain Names – Concept and Facilities", RFC 1034, USC/Information Sciences Institute, November 1987.

[Neuman, 92] *Neuman, Barry Clifford*: „The Virtual System Model: A Scalable Approach to Organizing Large Systems.", Ph. D. Thesis, University of Washington, USA, 1992.

[Schwartz, 89] *Schwartz, Michael F.*: „Resource discovery and related projects at the University of Colorado.", Technical Report CU-CS-508-91, Department of Computer Science, University of Colorado, Boulder, January 1991.

[Sechrest, 92] *Sechrest, Stuart and McClennen, Michael:* „Blending Hierarchical and Attribute-Based File Naming", Dept. of Electrical Engineering and Computer Science, University of Michigan, IEEE Computer Society, Proceedings of the 12th ICDCS, Yokohama, Japan, June 9–12, 1992, pp. 572–580.

[Stroud, 87] *Stroud, Robert James:* „Naming issues in the design of transparently distributed operating systems", Ph.D. Thesis, Newcastle upon Tyne: University of Newcastle upon Tyne, Computing Laboratory, 1987.

[Traub, 94] *Traub, Stefan:* „The Design of a Distributed Oberon System", In the Proceedings of the Joint Modular Languages Conference, University of Ulm, Sept. 1994.

[Wirth, 91] *Wirth, Niklaus and Reiser, Martin:* „Programming in Oberon. Steps beyond Pascal and Modula-2.", ISBN 0-201-56543-9.

[Wirth, 92] *Wirth, Niklaus und Gutknecht, Jürg:* „Project Oberon – The Design of an Operating System and Compiler", Addison-Wesely, 1992.

[Younger, 92] *Younger, E.J. and Bennett, K.H.:* „Functional approach to mathematical model of naming", Information and Software Technologie, 34, No. 3, March 1992, pp. 185–191.

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03 *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06 *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07 *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluationof Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04 *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05 *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06 *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07 *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08 *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09 *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen