— VS —

Namensdienste

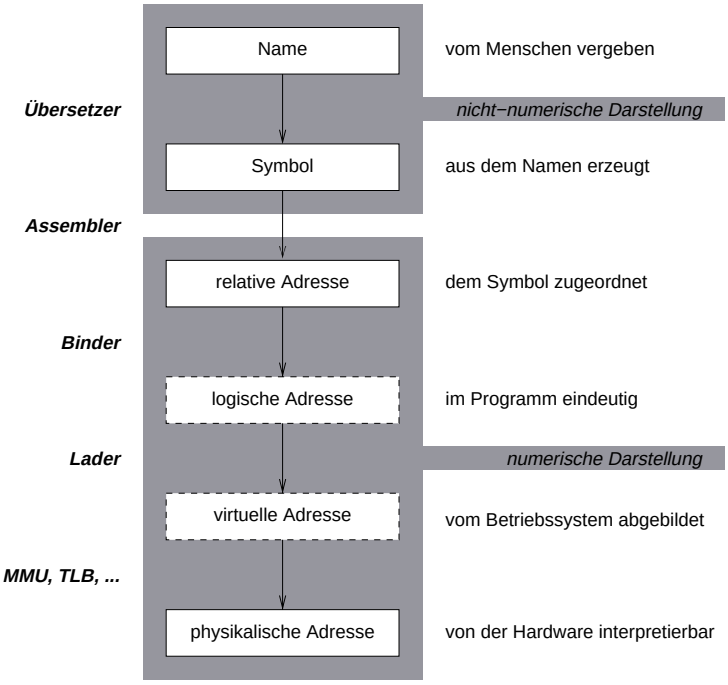
Verteilte Systeme, ©Wolfgang Schröder-Preikschat

Überblick

Sage mir Deinen Namen und ich sage Dir, wie Du heisst.

Name

Adresse



Name $\Longrightarrow$ Symbol

```
extern void foo () {}

namespace Foo {
    void foo () {}

    struct Bar { void foo (); };
    void Bar::foo () {}

    template <void (*bar)()>
    void foo () { bar(); }
};

extern "C" void bar () {
    Foo::foo<foo>();
}
```

.....g++ -S.....

```
.text
.globl __Z3foov
__Z3foov: ret

.globl __ZN3Foo3fooEv
__ZN3Foo3fooEv: ret

.globl __ZN3Foo3Bar3fooEv
__ZN3Foo3Bar3fooEv: ret

.globl __ZN3Foo3fooIXadL_Z3foovEEEEvv
__ZN3Foo3fooIXadL_Z3foovEEEEvv: ret

.globl _bar
_bar: ret
```


Symbol $\Longrightarrow$ Adresse

```
.text
.globl __Z3foov
__Z3foov: ret

.globl __ZN3Foo3fooEv
__ZN3Foo3fooEv: ret

.globl __ZN3Foo3Bar3fooEv
__ZN3Foo3Bar3fooEv: ret

.globl __ZN3Foo3fooIXadL_Z3foovEEEEvv
__ZN3Foo3fooIXadL_Z3foovEEEEvv: ret

.globl _bar
_bar: ret
```

..gcc -c..

```
00000000 b .bss
00000000 d .data
00000000 t .text
00000000 T __Z3foov
00000002 T __ZN3Foo3Bar3fooEv
00000001 T __ZN3Foo3fooEv
00000003 T __ZN3Foo3fooIXadL_Z3foovEEEEvv
00000004 T _bar
```

Symboltabelle

Name/Adresse in verteilten Systemen

- die *Abbildung* von Namen auf Adressen ist hier nur zur *Laufzeit* möglich
  - welche und wieviele Rechner den Verbund bilden werden, ist ungewiss
  - an welcher Adresse welcher Dienst zu finden ist, steht nicht statisch fest

  ⋮
  - ob und wohin Klienten Fernaufrufe absetzen, ist nicht im Voraus bekannt
  - welcher Anbieter welchen Fernaufruf zu erbringen vermag, ist offen

- „lose Kopplung“ der Rechner untereinander impliziert **dynamische Bindung**[1]

[1] Auch in konventionellen, nicht-verteilten Systemen ist dynamische Bindung bekannt und dort bereits seit geraumer Zeit ein Begriff (siehe Multics [5]). Diese wie auch die dynamische Bindung in verteilten Systemen ist jedoch keinesfalls mit dynamischer Bindung (*late Binding*) in objektorientierten Sprachen/Systemen zu verwechseln.

Nomen Est Omen

In einem verteilten System werden Namen verwendet, um auf die unterschiedlichsten Ressourcen zu verweisen wie beispielsweise Computer, Dienste, entfernte Objekte und Dateien sowie auch auf Benutzer. [1]

Namensverteilung ist grundlegend für verteilte Systeme

Namen. . .

- erleichtern die Kommunikation und die gemeinsame Nutzung von Betriebsmitteln
- werden benötigt, um Rechnersystemen eine Aktion für (ggf.) eins von vielen Betriebsmitteln abzuverlangen

Dazu muss eine *konsistente Namensgebung* gegeben sein[2]

[2]Benutzern fällt es beispielsweise recht schwer, per *Email* miteinander zu kommunizieren, wenn sie sich nicht gegenseitig beim Namen nennen können.

Name vs. Identifikation

Name ist vom Menschen interpretierbar, z.B.:

- Dateiname `index.html`
- Internet Domäne `informatik.uni-erlangen.de`
- URL `http://www4.informatik.uni-erlangen.de/~wosch`

Identifikation ein Name, der (nur) von Programmen interpretierbar ist, z.B.:

- Kommunikationsendpunkt zur IPC
- Dateideskriptor im NFS
- Verweise beim RPC

Jeder Prozess, der Zugriff auf ein Betriebsmittel benötigt, muss einen Namen oder eine Identifikation (ID) dafür kennen

„Namen sind Schall und Rauch“

reine Namen stehen einfach für *nicht interpretierte Bitmuster* [4]

- sie müssen immer (bei einem Namensdienst) nachgeschlagen werden
  - unaufgelöst sind sie nicht von irgendeinem Nutzen
- dagegen enthalten „unreine“ Namen zusätzliche Objektinformationen
  - z.B. Hinweise bzw. konkrete Angaben über den Ort des Objektes

Extremes Gegenbeispiel eines reinen Namens ist die *Adresse*: ein interpretiertes Bitmuster bzw. ein Wert, der die Position (d.h., den Ort) eines Objektes bezeichnet und nicht das eigentliche Objekt

→ S. 36

Adresse

- ermöglicht den *effizienten Zugriff* auf Objekte bzw. Betriebsmittel
  - werden Objekte (zur Laufzeit) verschoben, sind ihre alten Adressen ungültig
    - dies widerspricht einer möglichen Forderung nach Migrationstransparenz
  - **Relokation** wird ggf. erforderlich, ist aber leider nicht immer durchsetzbar
    - die alten Adressen müssten „eingezogen“ und aktualisiert werden
    - dazu müssten alle Adressen ihrer Verwendungsstellen bekannt sein
- garantiert *keinen dauerhaften Verweis* auf ein (fernes) Objekt[3]
  - ist damit nicht ausreichend für die Identifizierung in verteilten Systemen

[3] Auch Adressen auf die Objekte eines Programms sind nicht dauerhaft. Bereits der Einsatz eines anderen Übersetzers oder die Aktivierung des Optimierers kann zu anderen Adressen führen, ohne dass das Programm (von Hand) verändert worden ist.

Attribut

- ist allgemein der *Wert* einer einem Objekt zugeordneten *Eigenschaft*:

  Attribute sind für die Bezeichnung von Objekten sehr viel leistungsfähiger als Namen: es können Programme geschrieben werden, um Objekte gemäß genauer Attributspezifikationen auszuwählen, für die keine Namen bekannt sind bzw. für die Namen nicht bekannt sein sollen. (nach [1])

- ermöglicht, die interne Struktur eines Unternehmens nicht offenzulegen
  - im Gegensatz zu organisierten partitionierten Namen: `cs.uni-erlangen.de`

- ein besonders relevantes **Schlüsselattribut** eines Objektes ist seine *Adresse*

Bindung

- bezeichnet die Zuordnung zwischen einem Objekt und einem Namen
  - i.A. werden Namen zu *Attributen* des benannten Objektes gebunden und nicht zu der Implementierung der eigentlichen Objekte
  - die Bindung ist dynamisch d.h. erfolgt erst zur Laufzeit des Systems

- gestattet die spätere *Auflösung* eines Namens zu seinen Attributen
  - d.h. die Übersetzung in Daten über das benannte Objekt bzw. Betriebsmittel
  - häufig, um eine Aktion für das betreffende Objekt zu starten

- Fernaufrufe bedingen i.A. die Auflösung des Namens des passenden Anbieters

Name $\Longleftrightarrow$ Adresse

Eine „Adresse“ kann auch als zusätzlicher Name betrachtet werden, der nachgeschlagen werden muss, aber auch einen solchen „Namen“ enthalten kann:

IP-Adresse muss nachgeschlagen werden, um eine Netzwerkadresse zu erhalten

- die Netzwerkadresse kann z.B. eine Ethernet-Adresse darstellen

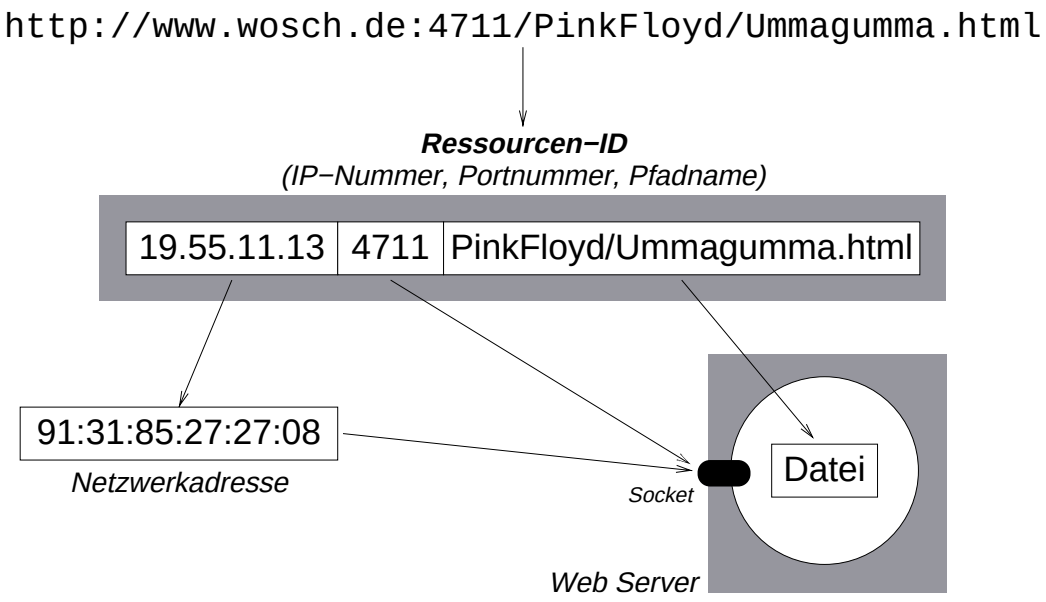
URL muss nachgeschlagen werden, um einen *Web Server* zu erhalten

- ein *Web Browser* interpretiert einen Domänennamen als URL

***Email*-Adresse** muss nachgeschlagen werden, um einen *Email Server* zu erhalten

- ein *Email Client* interpretiert einen Domänennamen als *Email*-Adresse

Interpretation eines Domänennamens



Universal Resource Identifier, URI (1)

URL *Uniform Resource Locator*, stellt die *Adresse* einer Web-Ressource dar

- ist effizient und für eine unbegrenzte Menge an Web-Ressourcen skalierbar
- wird die Ressource verschoben/gelöscht, entsteht eine „*hängende Referenz*“[4]

URN *Uniform Resource Name*, löst das Problem hängender Referenzen

- ist der (migrationstransparente) *dauerhafte Name* einer Web-Ressource
  - wird mit dem URL bei einem „URN-Nachschlagedienst“ hinterlegt
  - Auflösungsanfragen betreffen den URN, geliefert wird der zugehörige URL
- hat die Form `urn:`*Namensraum*`:`*Name*, z.B. `urn:ISBN:0-13-183369-3`

[4] Die Verwendung einer hängenden Referenz kann (a) zum Fehler führen oder (b) eine andere Ressource liefern, die jetzt an der durch den URL adressierten Stelle liegt.

Universal Resource Identifier, URI (2)

URC *Uniform Resource Characteristic*[5], die ausschließlich auf Attributen basierende Beschreibung einer Web-Ressource

- die (abstrakte) Beschreibung gilt schließlich als Suchschlüssel:
  „`author =` Harro Schlamm“
  „`title =` Schmonz und die Regenwürmer“
  „`keywords =` Regen, Schlamm, Würmer“
- stellt eine URN-Untermenge dar, ist eine Verfeinerung der Namensgebung
  - wird mit URL (ggf. URN) bei einem „URC-Nachschlagedienst“ hinterlegt
  - Auflösungsanfragen betreffen die URC, geliefert wird der zugehörige URL

[5] Auch als *Uniform Resource Citation* bezeichnet.

Kontext

- Namen besitzen nur in einem bestimmten Zusammenhang ihre *Bedeutung*
  - „siebenundvierzigelf“ bedeutet nicht immer gleich „Köllnisch Wasser“
  - gleiche Namen unterschiedlicher Kontexte meinen unterschiedliche Dinge
  - verteilte Systeme setzen sich aus beliebigen Kontexten zusammen
- ein Name ist einem (wohl definierten) **Gültigkeitsbereich** zugeordnet[6]
  - der den Kontext des jeweiligen Namens repräsentiert bzw. beschreibt
- der Namenskontext ist Semantik gebende Komponente eines *Namensraums*

[6] Ganz in Analogie z.B. zu blockorientierten Programmiersprachen.

Namensräume

Namensraum (*name space*) ist die Menge aller gültigen Namen, die ein bestimmter „Nachschlagedienst“ erkennt; ist syntaktisch definiert

- dabei kann die „Namensmenge“ hierarchisch oder flach organisiert sein
- ferner müssen gültige Namen nicht zwingend auf ein Objekt verweisen[7]

Namensdomäne (*name domain*) ist ein Namensraum, für den es eine einzige übergreifende administrative Autorität für die Zuordnung von Namen innerhalb der $\sim$ gibt; besitzt Kontrolle darüber, welche Namen zugeordnet werden dürfen

- die Kontrollaufgabe „Namenszuordnung“ kann aber auch delegiert werden

[7] Die Namen könnten überhaupt nicht oder z.B. an einem Attribut „*not present*“ gebunden sein.

Flacher Namensraum

- in dieser Organisationsform besitzen die Namen keine interne Struktur[8]
  - sie sind dann oftmals einfache numerische oder symbolische Bezeichner:
    42, 4711, foo, bar,
    siebenundvierzigelf, hortmötzivasarhelikutasipusta,
    hottentottentrottelmutterlattengitterwetterkotterbeutelrattenfangprämie
  - äußere Struktur schafft (ggf.) Lesbarkeit: sieben-und-vierzig-elf
- die Ausdehnung eines flachen Namensraums ist normalerweise endlich
  - letztlich bestimmt durch die ggf. maximal erlaubte Länge eines Namens

[8] Die Namen enthalten keine vom System (d.h. Nachschlagedienst) zu interpretierenden Strukturdaten.

Hierarchischer Namensraum

- die Namen haben eine interne Struktur, die ihre (relative) Position darstellt
  - sie entsprechen dann oftmals (komplex aufgebauten) *Pfadnamen*:
    sieben-und-vierzig-elf
    hort/mötzi/vasar/heli/kuta/si/pusta
    hottentotten.trottel.mutter.latten.gitter.wetter.kotter.beutel.ratten.fang.prämie
  - vom System zu interpretierende Strukturdaten trennen *Namenskomponenten*
- die Ausdehnung eines hierarchischen Namensraums ist potentiell unendlich
  - eine Namenskomponente bezeichnet einen Kontext: außer elf, pusta, prämie
  - jede einzelne Komponente wird relativ zu einem separaten Kontext aufgelöst
  - durch Einbindung von Kontexten ist eine beliebige Ausdehnung möglich

Heterogener Namensraum

- ein Name kann Bezeichner sein für mehr als nur einen Kontext oder ein Objekt
  - er kann { die „Kreuzung“ zweier Namensräume; die „Wurzel“ eines Dateibaums; das „Tor“ eines Sicherheitsprinzipals } darstellen
- er kann allg. das *Stammverzeichnis* eines anderen Namensraums benennen
  - der ggf. einer anderen syntaktischen Definition unterliegt
    - z.B. die Namenskomponenten durch ‘/’ anstatt ‘.’ voneinander trennt
  - der von einem „problemorientierten“ Nachschlagedienst implementiert wird
- der Name bezeichnet ggf. die Übergabestelle zu anderen Nachschlagediensten

Kombinierter Namensraum (1)

- ein Namensraum kann ganz oder teilweise in einen anderen eingebettet sein
  - analog zur in UNIX bekannten Einbettung von Dateisystemen (`mount(2)`)
- die Einbettung hat ggf. einen heterogenen Namensraum zur Folge
  - entsprechend können die Namen heterogen aufgebaut bzw. strukturiert sein:
    - z.B. sieben.und.vierzig.elf.4711/zwei/und/vierzig/42
    - dabei benennt 4711 die Einbettungs- bzw. Übergabestelle
    - /zwei/und/vierzig/42 wird einem anderen Nachschlagedienst zugeführt
    - um von dort die Attribute des mit 42 benannten Objekts abzurufen
- das **Alias**-Konzept hilft, komplexe Namenstrukturen zu vereinfachen

Kombinierter Namensraum (2)

Das Fazit ist, dass wir Namensräume immer kombinieren können, indem wir einen Stammverzeichniskontext auf einer darüber liegenden Ebene einführen, aber das kann zu Problemen mit der Abwärtskompatibilität führen. Die Korrektur des Kompatibilitätsproblems wiederum führt uns zu hybriden Namensräumen und der Unbequemlichkeit, alte Namen zwischen den Benutzern der beiden Computer übersetzen zu müssen. [1]

(a) Das Objekt `/etc/foobar` kann für Programme der Rechner `foo` und `bar` (auf beiden Rechnern) als `/foo/etc/foobar` bzw. `/bar/etc/foobar` erreichbar gemacht werden. Abwärtskompatibilität zu Programmen, die nur `/etc/foobar` verwenden, ist nicht gegeben.

(b) Das Objekt `/etc/foobar` bleibt lokal wie gehabt, d.h. abwärtskompatibel, erreichbar. Auf Rechner `foo` wird das `bar`-seitige `/etc/foobar` dann über `/bar/etc/foobar` erreicht (umgekehrt gilt entsprechendes). Der Namensraum ist dadurch nicht einheitlich, sondern hybrid.

Namensalias

- der von UNIX her bekannte „symbolische Link“ auf (komplexe) Namen
  - bei der Namensauflösung wird ein Alias zu seinem Wert „expandiert“
  - z.B. können Domänennamen durch andere Domänennamen ersetzt werden
- die Verwendung eines Alias ist u.a. ein Mittel zur Sicherstellung von Transparenz
  - der Name des Rechners, der z.B. den *Web Server* ausführt, bleibt verborgen: www4.informatik.uni-erlangen.de → faui42y.informatik.uni-erlangen.de
  - Klienten greifen so über einen **generischen Namen** auf die Objekte zu
  - wandern die so benannten Objekte, ist nur der Alias zu aktualisieren
- Aliase sind vom Nachschlagedienst zu erkennen und entsprechend aufzulösen

Namensdomänen

- die Domäne als Begriff für die administrative Autorität eines Namensraums
  - legt die Verantwortlichkeit für eine Namensdomäne fest
  - geht einher mit der Verwaltung/Aktualisierung einer *Namensdatenbank*
  - jede Namensdomäne besitzt normalerweise ihre eigene (verteilte) Datenbank
- die Verwaltung ist übergeordnet, sie kann auf *Subdomänen* erweitert werden
  - die Subdomäne `informatik.uni-erlangen.de` ist relativ eigenständig
    - sie kann beliebige Namen enthalten und autonom verwalten
  - ihr Name war mit der Autorität für `uni-erlangen.de` abzustimmen
- Namensdaten unterschiedlicher Domänen werden i.A. nicht zusammen abgelegt

Ora Et Labora

Im Allgemeinen ist die Auflösung ein *iterativer Prozess*, wobei der Name wiederholt Namenskontexten präsentiert wird.

- Ein Namenskontext bildet entweder einen gegebenen Namen direkt auf eine Menge elementarer Attribute ab oder auf einen weiteren Namenskontext und einen abgeleiteten Namen, der diesem Kontext präsentiert wird.
- Um einen Namen aufzulösen, wird er zunächst einem ersten Namenskontext präsentiert; die Auflösung iteriert, solange weitere Kontexte und abgeleitete Namen zurückgegeben werden.

Ein weiteres Beispiel der iterativen Natur der Auflösung ist die Verwendung von Aliasen. Die Verwendung von Aliasen lässt zu, dass im Namensraum Zyklen auftreten — weshalb die Auflösung ggf. niemals beendet wird. (nach [1])

Namensauflösung (1)

Die Suche nach Namensdaten auf mehreren Namensanbietern (*name server*) zur Auflösung eines Namens wird auch als **Navigation** bezeichnet:

iterative $\sim$ der Klient kontaktiert nacheinander die Namensanbieter, um einen Namen aufzulösen

- der erste Anbieter, der den Namen auflösen kann, beendet den Suchprozess

Multicast $\sim$ der Klient sendet die Aufforderung zur Auflösung an eine Gruppe von Namensanbietern

- problematisch ist der Fall, wenn mehr als ein Anbieter eine Auflösung hat

Namensauflösung (2)

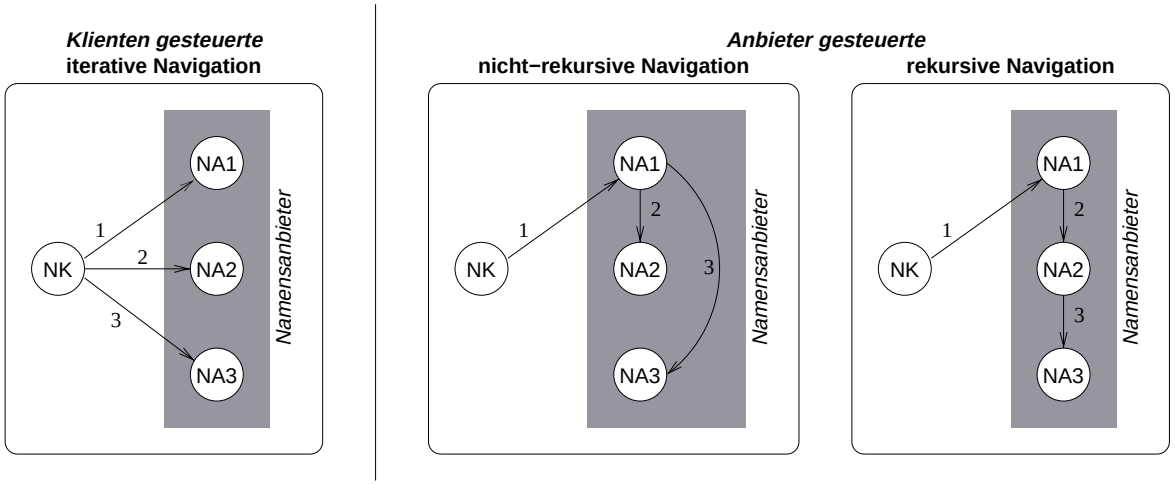
Ergänzend bzw. alternativ zum klientenseitig kontrollierten Suchprozess gibt es die **Anbieter gesteuerte Navigation**, bei der ein Namensanbieter die Auflösung eines Namens koordiniert und andere gleichrangige Anbieter kontaktiert, wenn er selbst den Namen nicht auflösen kann:

nicht-rekursive $\sim$ der ausgewählte Anbieter kommuniziert (iterativ oder über Multicast) mit gleichrangigen Namensanbietern

rekursive $\sim$ der ausgewählte Anbieter beauftragt einen anderen gleichrangigen Namensanbieter, mit der Auflösung fortzufahren

- der jeweils beauftragte Anbieter speichert ein (größeres) Präfix des Namens

Klienten vs. Anbieter gesteuerte Navigation



Auffinden der Namensanbieter

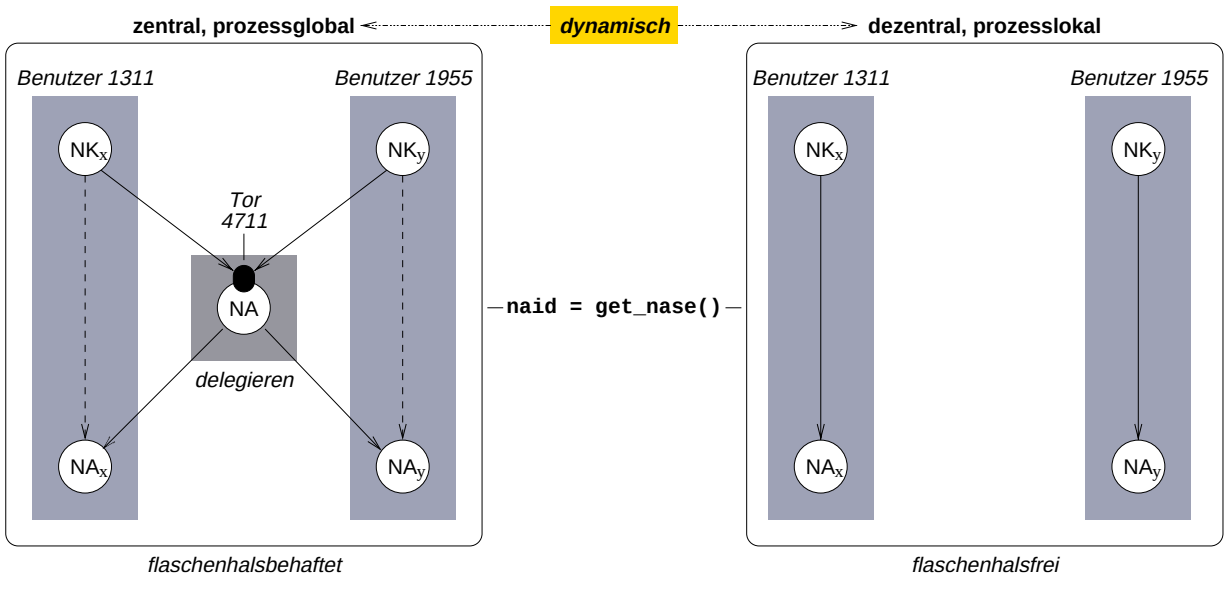
- im Regelfall sind Namensanbieter (voneinander isolierte) Prozessinkarnationen
  - die Namensauflösung geht mit Fernaufrufen an Namensanbietern einher
  - dazu ist die Adresse des den RPC empfangenden Prozesses zu ermitteln
  - die Adresse ist Attribut des Namens eines jeweiligen Namensanbieters
  - der Name ist bei einem (anderen) Namensanbieter nachzuschlagen **!?**
- ein Henne-Ei-Problem: wo kommt die Adresse des ersten Namensanbieters her?
  1. sie ist statisch allen Klienten bekannt und verweist z.B. auf ein **Tor** (*port*)
  2. sie ist Attribut der einen Klienten repräsentierenden Prozessinkarnation [6]
- der Toransatz unterstützt *anwendungsbezogene Namensräume* nur suboptimal

Anwendungsbezogene Namensräume

- Namen verteilter Anwendungen entsprechen logischen/virtuellen Adressen
  - gleiche logische Adressen sind auf verschiedene physikalische abzubilden
  - gleiche {Namen,Adressen} liegen in disjunkten {Namens,Adress}räumen
  - verteilte Anwendungen erfordern individuelle Abbildungsfunktionen[9]
- „Stammnamensanbieter“ sind *prozessrelativ* zu adressieren bzw. aufzufinden
  - torbasierte Adressierung führt alle Klienten zum selben Namensanbieter
    - es sei denn, die Toradresse ist prozessrelativ bzw. ein Prozessattribut
  - der „zentrale Namensanbieter“ muss Klienten Namensräumen zuordnen

[9] In Analogie dazu, dass in konventionellen (nicht verteilten) Systemen die Abbildungsfunktion für logische/virtuelle Adressen ebenfalls anwendungsbezogen und für jeden (schwergewichtigen) Prozess eigenständig definiert ist.

Stammnamensanbieter (1) Adressierung



Stammnamensanbieter (2) — Auflösung

prozessglobal typischerweise über eine Benutzerkennung (*User ID*)

- der zentrale Namensanbieter erfragt die *UID* des Klienten vom System
  - sie ggf. der Aufforderungsnachricht zu entnehmen, ist unsicher [warum?]
  - Auswahlhilfe ist z.B. die vom `receive()` gelieferte *PID* [warum?]
- anhand der *UID* wird der anwendungsbezogene Namensraum ausgewählt
  - ggf. wird die Aufforderung zur Namensauflösung delegiert

prozesslokal sicher und vergleichsweise sehr einfach, analog zu `getpid(2)`

- der Kern liefert das der Namensanbieteradresse entspr. Prozessattribut

Der Klient fordert schließlich „seinen“ Namensanbieter zur Namensauflösung auf

Stammnamensanbieter (3) — Bindung

- die Stammnamensanbieteradresse wird bei der Prozesserzeugung festgelegt
  - das entsprechende Prozessattribut wird z.B. (analog `fork(2)`) vererbt
    - vom erzeugenden Elternprozess auf den zu erzeugenden Kindprozess
  - oder es wird explizit vom Eltern-/Kindprozess definiert (bzw. verändert)

- die Zuordnung des Prozessattributs ist eine **privilegierte Operation** [warum?]
  - sie ist vom (verteilten) Betriebssystem entsprechend zu kontrollieren[10]
  - die (kontrollierte) Änderung zur Prozesslaufzeit ist ggf. zu unterstützen

[10] Auch in konventionellen (nicht verteilten) Systemen wird nicht jeder Prozess nach Belieben und unkontrolliert auf die MMU zugreifen und diese ggf. programmieren dürfen, um seinen logischen Adressraum verändern zu wollen.

Name als „Fähigkeit“ (*Capability*)

- den Namen einer Ressource zu kennen, eröffnet einen allgemeinen Zugriffsweg
  - Ressourcen zu verbergen bedeutet auch, Namen nicht zu veröffentlichen
  - **Zugriffsschutz** ist erreichbar, wenn Namen nicht erraten werden können
  - es besteht jedoch ein *Tradeoff* zwischen lesbare und nicht erratbare Namen
  - der daraus resultierende Konflikt führt letztlich zu Kompromislösungen
- die Kompromislösung setzt auf eine **Identifikation** von Ressourcen → S. 9
  - d.h. von Programmen erzeugte, verifizierbare und interpretierbare Namen
  - dadurch wird die *kontrollierte Adressierung* der Ressourcen möglich [2, 3]
  - „Sicherheitskerne“ (*security kernels*) stellen die Überwachungsfunktionen

„Geheimlagerung“ — *Caching*

- Ergebnisse von Namensauflösungen zwischenzulagern, verkürzt Antwortzeiten
  - zu den positiven Suchergebnissen werden die negativen ggf. auch gespeichert

    { Klient, Anbieter } befragt sein „Namensversteck“ vor { dem Anbieter, anderen Anbietern }
  - Kommunikation („Fernaufrufe“) mit den Namensanbietern wird eingespart
- zusätzlich können Namensanbieter aus dem Navigationspfad eliminiert werden
  - Namensauflösung ist dann trotz Anbieterausfälle (eingeschränkt) möglich
- die Namensdaten sind typischerweise (relativ) stabil → Konsistenzwahrung

Nachschlagedienste

Namensdienst *speichert* eine Auflistung von Bindungen zwischen Namen und Attributen aus einem oder mehreren Namenskontexten

Verzeichnisdienst ist ein *attributbasierter Namensdienst*, der Einträge anhand auf Attributen basierenden Angaben *sucht*

Erkennungsdienst ist ein Verzeichnisdienst, der die in einer *spontanen Netzwerkumgebung* bereitgestellten Betriebsmittel (bzw. Dienste) registriert

Namensdienst — *Name Service*

- die zentrale Funktion, die zu unterstützen ist, ist die **Namensauflösung**[11]
  - Attribute sind nach einem bestimmten vorgegebenen Namen zu suchen
- der Dienst ist wegen der Offenheit verteilter Systeme oft isoliert; Motivation:

  Vereinheitlichung Ressourcen, die von unterschiedlichen Diensten verwaltet werden, verwenden dasselbe Namensschema (z.B. URL)

  Integration ermöglicht es, dass die administrativen Domänen keine völlig unterschiedlichen Namenskonventionen verwenden
- derartige Dienste werden auch als **weiße Seiten** (*white pages*) bezeichnet

[11] Weitere Funktionen betreffen das Ersetellen/Löschen von Bindungen, das Auflisten gebundener Namen, sowie das Hinzufügen und Entfernen von Kontexten.

Verzeichnisdienst — *Directory Service*

- sucht nach Einträgen von Namen, die bestimmten Attributen entsprechen[12]
  - *Active Directory*, X.500, *Lightweight Directory Access Protocol* (LDAP)
- Attribute sind für die Bezeichnung von Objekten leistungsfähiger als Namen:
  - es können Programme geschrieben werden, um Objekte, für die keine Namen bekannt sind, gemäß genauer Attributspezifikationen auszuwählen
  - der Außenwelt wird die Struktur eines Unternehmens nicht offen gelegt
- derartige Dienste werden auch als **gelbe Seiten** (*yellow pages*) bezeichnet

[12] Dabei ist auf die **Dualität** von *Name* und *Attribut* (des Namens) hinzuweisen. Siehe auch S. 14.

Erkennungsdienst — *Discovery Service*

- **spontane Netzwerke** stellen besondere Anforderungen an Nachschlagedienste:
  - Geräte schließen sich ohne Vorwarnung und ohne „Spielraum“ für eine administrative Vorbereitung an das bestehende Netz an
  - Klienten und Anbieter (d.h. Dienste) ändern sich dynamisch, sie müssen jedoch ohne Benutzereingriff integriert werden
- bietet eine Schnittstelle zur automatischen {R,Der}egistrierung von Diensten
  - Klienten können u.a. prüfen, welche der benötigten Dienste verfügbar sind
- die Dienstsuche muss nicht zwingend von Benutzern d.h. Menschen ausgehen
  - ebenso können „Geräte“ die Suche auslösen: *eingebettete Systeme*, Jini[7]

Zusammenfassung

- Namensdienste speichern Objektattribute, die sie auf Anfrage zurückgeben
  - ein zentrales und besonders wichtiges Attribut ist die *Adresse*
- zentraler Entwurfsaspekt dabei ist die Struktur eines Namensraums, d.h.
  - ob bestimmte syntaktische Regeln für die Namen gelten sollen
  - ob der Namensraum hierarchisch oder flach organisiert sein soll
- eng damit verbunden ist das Auflösungsmodell und die Navigationsvarianten
  - Regeln, nach denen ein Name in eine Attributmenge aufzulösen ist
  - iterative, Multicast, rekursive oder nicht-rekursive Navigation
- Replikation und „Geheimlagerung“ tragen zur Erhöhung der Verfügbarkeit bei

Referenzen

[1] G. Coulouris, J. Dollimore, and T. Kimberg. *Verteilte Systeme: Konzepte und Design*. Pearson Education, 2002. ISBN 3-8273-7022-1.

[2] R. S. Fabry. Capability-Based Addressing. *Communications of the ACM*, 17(7):403–412, July 1974.

[3] S. J. Mullender and A. S. Tanenbaum. The Design of a Capability-Based Distributed Operating System. *The Computer Journal*, 29(4):289–299, 1986.

[4] R. Needham. Names. In S. Mullender, editor, *Distributed Systems, an Advanced Course*, pages 315–326. ACM Press/Addison-Wesley, second edition, 1993.

[5] E. Organick. *The Multics System: An Examination of its Structure*. MIT Press, 1972.

[6] M. Sander, H. Schmidt, and W. Schröder. Naming in the PEACE Distributed Operating System. In *Proceedings of the International Workshop on Communication Networks and Distributed Operating Systems within the Space Environment*, pages 293–302, Nordwijk, The Netherlands, October 24–26, 1989. ESTEC.

[7] J. Waldo. The Jini Architecture for Network-centric Computing. *Communications of the ACM*, 42(7):76–82, 1999.