Namens-verwaltung

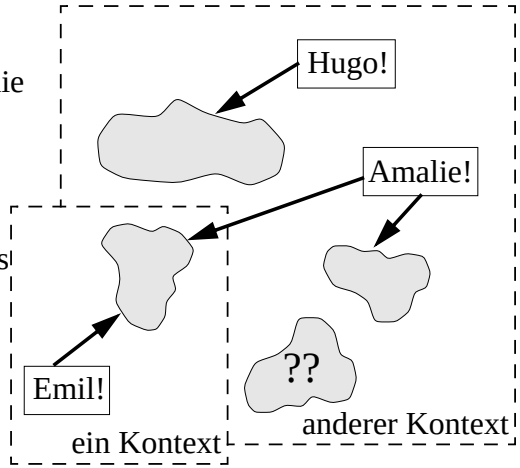
Namen und Namensverwaltung

Namen sind Schall und Rauch

Nomen est omen

- *Namen* sind Symbole, die typischerweise durch Zeichenketten repräsentiert werden
  - benutzerorientierte Namen haben im Unterschied zu Adressen (oder maschinenorientierten Namen) i.a. keine feste Länge

- Dienen der (eindeutigen) *Bezeichnung von Objekten*
  - daher oft auch “Bezeichner”
  - es gibt auch *anonyme* Objekte (z.B. dynamische Variablen, die mit “new” erzeugt werden)
  - ein Objekt kann u.U. mehrere Namen haben (“*alias*”)
  - innerhalb eines *Kontextes* muss ein Name *eindeutig* sein
  - Benutzer soll ein Objekt einfach *umbennen* können
  - gleicher Name kann zu *verschiedenen Zeiten* unterschiedliche Objekte bezeichnen



- Beispiele für bezeichnete Objekte
  - in Programmiersprachen:

    Variablen, Prozeduren, Datentypen, Konstanten...
  - in verteilten Systemen:

    Dienste, Server, Maschinen, Benutzer, Dateien, Betriebsmittel...

Zweck von Namen

Typ, Gestalt, Zweck...

- Geben Aufschluss über die Art eines Objektes
  - falls Name (für Benutzer) sinnvoll gewählt
  - z.B. Konventionen xyz.c, xyz.o, xyz.ps oder “printer”

- Dienen der *Identifizierung* von Objekten
  - daher oft auch “Identifikator” für “Name”
  - Sprechweise oft: “Objekt A” statt “das mit ‘A’ bezeichnete Objekt”

- Ermöglichen die *Lokalisierung* von Objekten
  - zwecks Manipulation der Objekte
  - über den Namen besteht eine Zugriffsmöglichkeit auf das Objekt selbst
  - Namen selbst sind aber oft unabhängig von der Objektlokation
  - besondere Herausforderung: Lokalisieren von *mobilen* Objekten

- Sind URLs Namen?
  - oder eher Adressen?
  - www.fuzzycomp.eu/Studium/bewerbung.html
  - 121.73.129.200/Studium/bewerbung.html

Namen und Adressen

- Jedes Objekt hat eine Adresse
  - Speicherplatzadressen
  - Internetadressen
  - Ethernetadressen
  - Port-Nummer bei TCP
  - ...

- Adressen sind “physische” Namen

Namen der untersten Stufe

- Adressen ermöglichen die *direkte Lokalisierung* eines Objektes

- Adressen sind ebenfalls innerhalb eines Kontextes (“Adressraum”) eindeutig

- Adresse eines Objektes ist u.U. *zeitabhängig*
  - mobile Objekte
  - “relocatable”

- *Dagegen*: Name eines Objektes ändert sich i.a. nicht
  - vgl. aber: Namensänderung bei Heirat, Zuweisung eines Alias...!

- Entkoppelung von Namen und Adressen unterstützt die *Ortstransparenz*

- Zuordnung Name --> Adresse nötig
  - vgl. persönliches Adressbuch
  - “Binden” eines Namens an eine Adresse

Binden

- Binden = Zuordnung Name --> Adresse
  - konzeptuell daher auch: Name --> Objekt
  - Namen, die bereits Ortsinformationen enthalten: “impure names”

- *Binden bei Programmiersprachen:*
  - Beim Übersetzen / Assemblieren
    --> “relative” Adresse
  - Durch Binder (“linker”) oder Lader
    --> “absolute” Adresse
  - Ggf. Indirektion durch das Laufzeitsystem
    - z.B. bei Polymorphie objektorientierter Systeme

- *Binden von Dienstaufrufen bei klassischen Systemen*
  - Dienstaufruf durch Trap / Supervisor-Call (“SVC”)
    - Name = SVC-Nummer (oder “symbolische” Bezeichnung)
  - Bei *Systemstart* wird eine Verweistabelle angelegt
    - “SVC table”, “switch vector”
  - Dienstadresse ändert sich bis zum reboot nicht

- *Binden in verteilten / offenen Systemen*
  - Dienste entstehen dynamisch, werden ggf. verlagert
    - haben ggf. unterschiedliche Lebenszyklen und -dauer
  - Binden muss daher ebenfalls *dynamisch* (“zur Laufzeit” bzw. beim Objektzugriff) erfolgen!

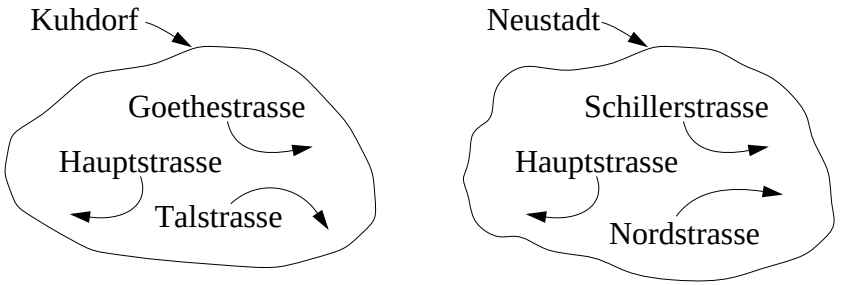
Namenskontext

- Namen werden relativ zu einem *Kontext* interpretiert (Namensraum)
  - “relative Namen” (gleiche Namen in verschiedenen Kontexten möglich)
  - *Interpretation* = Abbildung auf die gebundene Adresse oder einen Namen niedrigerer Stufe
  - Interpretation erfolgt oft mehrstufig, z.B.: Dateiname --> Adresse des Kontrollblocks --> Spur / Sektor auf einer Platte

- Namen sollen innerhalb eines Kontextes eindeutig sein
  - bzw. durch zusätzliche Attribute eindeutig identifizierbar sein

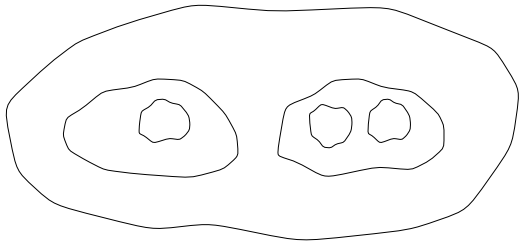
- Falls nur ein einziger Kontext existiert: *flacher Namensraum* (aus “absoluten Namen”)
  - Partition des Namensraum wird als “Domäne” bezeichnet

- Namenskontexte sind (i.a. abstrakte) Objekte, die selbst wieder einen Namen haben können
  - z.B. benannte Dateiverzeichnisse (“directory”)
  - übergeordneter Kontext --> *Hierarchie*



Hierarchische Namensräume

- Baumförmige Struktur von Namenskontexten

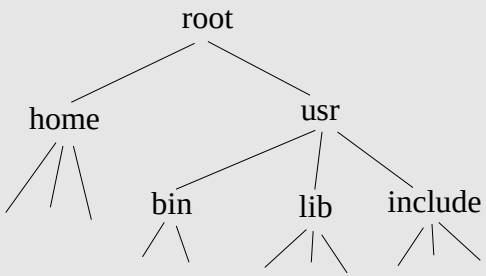


- Beispiel: Adressen im Briefverkehr
  - "Hans Meier, Deutschland" genügt nicht...

- Beispiel: Telefonsystem
  - Landeskennung
  - Ortsnetzkennung
  - Teilnehmerkennung

  32168 ist ein relativer Name, der z.B. im Kontext 08977 interpretiert werden muss

- Beispiel: UNIX-Dateisystem



Hierarchische Namensräume (2)

- Eignen sich gut für verteilte Systeme
  - besser als flache Namensräume
  - leichter skalierbar (z.B. zur Gewährleistung der Eindeutigkeit)
  - dezentrale Verwaltung der Kontexte durch eigenständige Autoritäten, die wieder anderen Autoritäten untergeordnet sind
  - Namensinterpretation stufenweise durch verteilte Instanzen
  - erleichtert Systemrekonfiguration
  - eindeutige absolute Namen durch Angabe des ganzen Pfades

- Strukturierte Namen
  - bestehen aus mehreren Komponenten
  - Komponenten bezeichnen typischerweise Kontexte
  - Bsp: root/usr/bin
  - Bsp: Meier.Talweg 2.Kuhdorf.Oberpfalz.Deutschland
  - Bsp: +49 08977 32168 (präfixfreier Code!)
  - oft geographisch oder thematisch gegliedert

- *Synonyme Namen* bezeichnen das gleiche Objekt
  - Bsp: der relative Name 'c' im Kontext 'a' bezeichnet das gleiche Objekt wie der absolute Name 'a.c'

- *Alias-Namen*: Synonyme im gleichen Kontext

Namensverwaltung (“name service”)

- Verwaltung der Zuordnung Name --> Adresse
  - Eintragen: “bind (Name, Adresse)” sowie Ändern, Löschen etc.
  - Eindeutigkeit von Namen garantieren
  - Zusätzlich ggf. Verwaltung von Attributen der bezeichneten Objekte
- Auskünfte (“Finden” von Ressourcen)
  - z.B. Adresse zu einem Namen (“resolve”: Namensauflösung)
  - z.B. alle Dienste mit gewissen Attributen (etwa: alle Postscript-Drucker) “yellow pages” <--> “white pages”
- Ggf. Schutz- und Sicherheitsaspekte
  - Capability-Listen, Schutzbits, Autorisierungen...
  - Dienst selbst soll hochverfügbar und sicher (z.B. bzgl. Authentizität) sein
- Ggf. Generierung eindeutiger Namen
  - innerhalb eines Kontextes (z.B. mit Zeitstempel oder lfd. Nummer)
  - bzw. global eindeutig (z.B. eindeutigen Kontextnamen als Präfix vor knotenlokale laufende Nummer; ggf. auch lange Zufallsbitfolge)

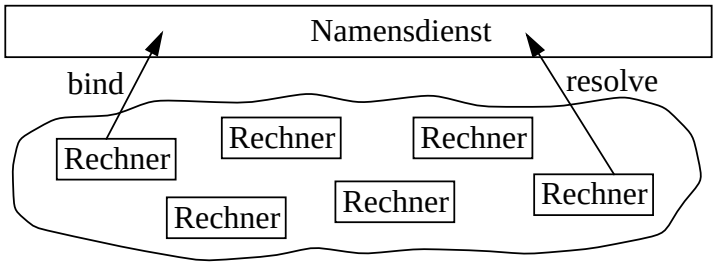
Vgl. “klassische” Dienste beim Telefonsystem:

- Telefonbuch
- Auskunft

} Abbildung Name --> Telefonnummer

- ggf. mehrstufig / dezentral: Auslandsauskunft wendet sich an Ortsauskunft im Ausland... (--> hierarchische Namenskontexte notwendig!)
- örtliche Telefonbücher sowie Ortsvorwahlverzeichnis sind repliziert
  - sonst Überlastung des zentralen Dienstes
  - Problem der verzögerten Aktualisierung (veraltete Information)
- “gelbe Seiten”: Suche nach Dienst über Attribute

Verteilte Namensverwaltung



logisch ein einziger Dienst; tatsächlich verteilt realisiert

- Jeder Kontext wird (logisch) von einem autonomen *Nameserver* verwaltet
  - ggf. ist ein Nameserver für mehrere Kontexte zuständig
  - ggf. Aufteilung / Replikation des Nameservers --> höhere Effizienz, Ausfallsicherheit
- Typischerweise *hierarchische Namensräume*
  - entsprechend strukturierte Namen
  - entsprechend kanonische Aufteilung der Verwaltungsaufgaben
  - Zusammenfassung Namen gleichen Präfixes vereinfacht Verwaltung
- Typisch: *kooperierende* Nameserver, die den gesamten Verwaltungsdienst realisieren
  - hierzu geeignete Architektur der Server vorsehen
  - Protokoll zwischen den Nameservern (für Fehlertoleranz, update der Replikate etc.)
  - Dienstschnittstelle wird i.a. durch lokale Nameserver realisiert

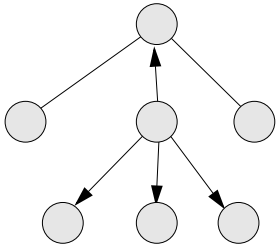
bzw. “user agent”

- *Annahmen*, die Realisierungen i.a. zugrundeliegen:
  - *lesende* Anfragen viel häufiger als schreibende (“Änderungen”)
  - *lokale* Anfragen (bzgl. eigenem Kontext) dominieren
  - seltene, temporäre *Inkonsistenzen* können toleriert werden

ermöglicht effizientere Realisierungen (z.B. Caching, einfache Protokolle...)

Namensinterpretation in verteilten Systemen

- Ein Nameserver kennt den Nameserver der nächst höheren Stufe



- Ein Nameserver kennt alle Nameserver der untergeordneten Kontexte (sowie deren Namensbereiche)

- Hierarchiestufen sind i.a. klein (typw. 3 oder 4)

- Blätter verwalten die eigentlichen Objektadressen und bilden die Schnittstelle für die Clients

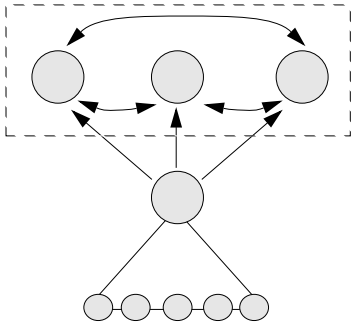
- Nicht interpretierbare Namen werden an die nächst höhere Stufe weitergeleitet (bei strukturierten Namen!)

Broadcast

- falls zuständiger Nameserver unbekannt
  ("wer ist für XYZ zuständig?" oder: "wer ist hier der Nameserver?")
- ist aufwendig, falls nicht durch Hardware etc. unterstützt (wie z.B. bei LAN)
- nur in begrenzten Kontexten anwendbar

Repliklation von Nameservern

- Zweck: Erhöhung von Effizienz und Fehlertoleranz
- Vor allem auf höherer Ebene relevant
  - dort viele Anfragen
  - Ausfall würde grösseren Teilbereich betreffen



- Server kennt alle übergeordneten Server
- Broadcast an ganze Servergruppe, oder Einzelnachricht an "nächsten" Server; anderen Server erst nach Ablauf eines Timeouts befragen

- Replizierte Server konsistent halten
  - ggf. nur von Zeit zu Zeit gegenseitig updaten (falls veraltete Information tolerierbar)
  - Update auch dann sicherstellen, wenn einige Server zeitweise nicht erreichbar sind (periodisches Wiederholen von update-Nachrichten)
  - Einträge mit Zeitstempel versehen --> jeweils neuester Eintrag dominiert (global synchronisierte Zeitbasis notwendig!)

- Symmetrische Server / Primärserver-Konzept:
  - *symmetrische Server*: jeder Server kann updates initiieren
  - *Primärserver*: nur dieser nimmt updates entgegen
    - Primärserver aktualisiert gelegentlich "read only" Sekundärserver
    - Rolle des Primärservers muss im Fehlerfall von einem anderen Server der Gruppe übernommen werden