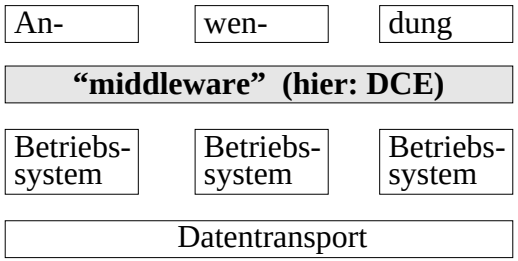
DCE - Distributed Computing Environment

- Entwickelt von einem herstellerübergreifenden Konsortium (“OSF” - Open Software Foundation)
  - Anfang der 90er Jahre, u.a. DEC, IBM, Siemens, HP...
  - trotz CORBA noch vielfältig eingesetzt in grossen Organisationen

- System aus zusammenwirkenden Softwarekomponenten (Werkzeuge, Dienste, Laufzeitmechanismen) zur Realisierung verteilter Anwendungen in offenen heterogenen Umgebungen

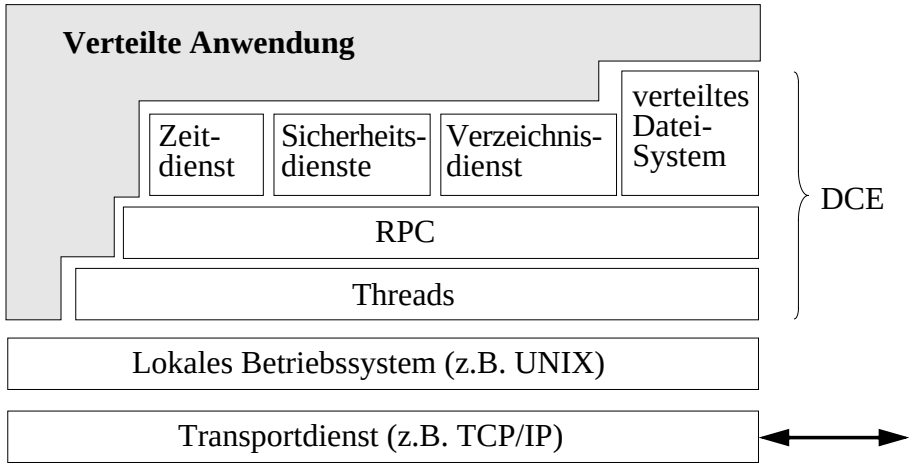


- Ziel: Schaffung eines “offenen” Industriestandards für verteilte Verarbeitung

- Vorgehensweise pragmatisch: Soweit möglich, Nutzung geeigneter existierender Technologiekomponenten

- Realisierung auf verschiedenen Plattformen
  - Hardware: IBM, Sun, ...
  - Software: UNIX-basiert, z.B. HP-UX, Solaris; aber auch Windows, OS/390, Macintosh,...

Hauptkomponenten des DCE



- Baut auf lokalem Betriebssystem und existierendem Transportdienst (z.B. TCP/IP) auf

- Threads und RPC sind Basisdienste, die von anderen Diensten (aber auch von Anwendungen) benutzt werden

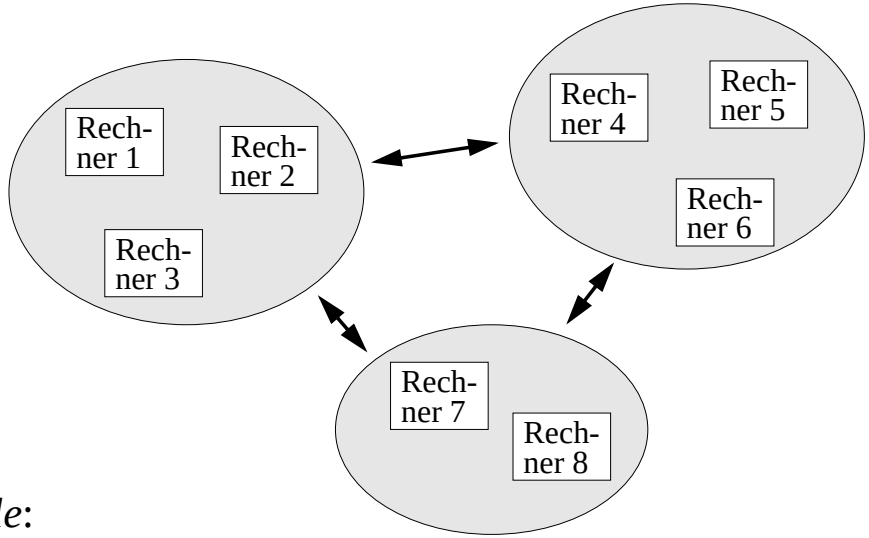
- Höhere (“verteilte”) Dienste: u.a. Dateisystem, Verzeichnis- und Namensdienst, Sicherheitsdienst

- Eine verteilte Anwendung nutzt die Dienste i.a. über Programmierschnittstellen (API: Application Programming Interface), die für die Sprache C ausgelegt sind

- Es gibt ferner Tools für Stub-Generierung von RPCs, Systemmanagement etc.

Globale DCE-Architektur: Zellen

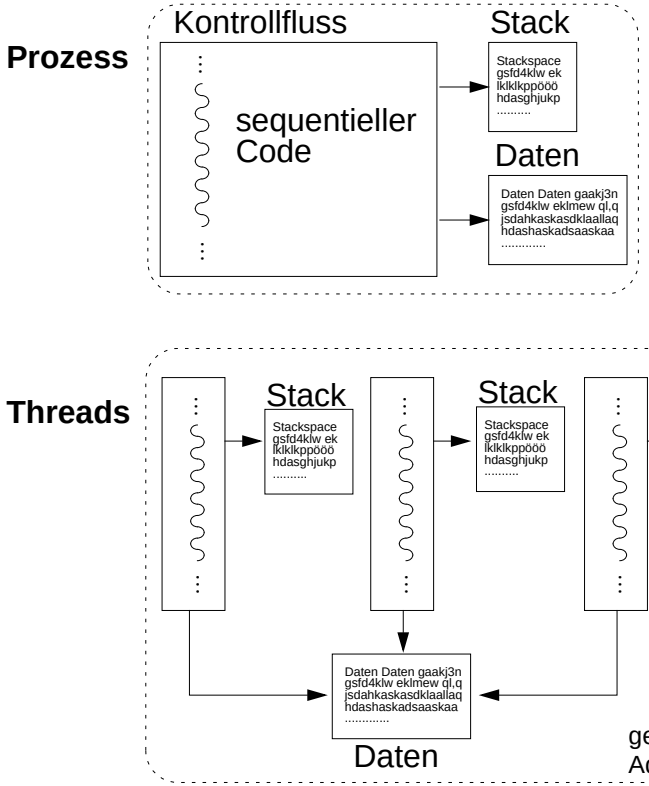
- Partitionierung der Rechner in sogen. *Zellen*
  - Subsysteme machen grosse Systeme handhabbarer



- *Zelle*:
  - Ist eine abgeschlossene organisatorische Einheit aus Rechnern, Ressourcen, Benutzern
    - z.B. Abteilung einer Firma
    - i.a. jeweils verantwortlicher Systemverwalter notwendig
    - bildet jeweils eine eigene Schutzzone bzgl. Sicherheitsaspekte
  - Hat *Cell Directory Service* (CDS), *Sicherheitsdienst* und *Zeitdienst* eingerichtet
    - realisiert durch dauerhafte Prozesse (“Dämonen”)
    - ggf. weitere Dienste, z.B. Distributed File System (DFS)
- Prozesse können per RPC zellübergreifend kommunizieren (bei Kenntnis entfernter Adressen)
- *Zellübergreifende Services* (z.B. Zeitservice, Namensverwaltung...) mittels dedizierter Protokolle

DCE: Threads

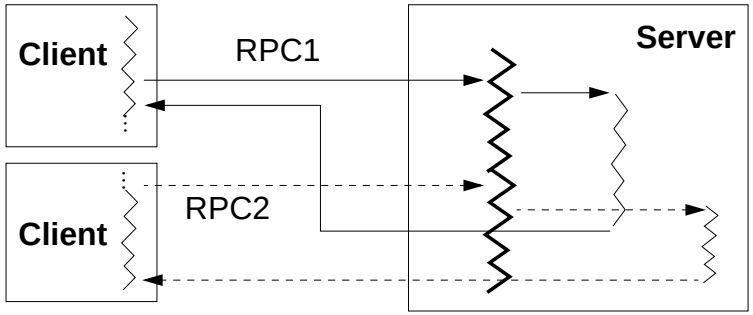
- Threads = leichtgewichtige Prozesse mit gemeinsamem Adressraum



- Einfache Kommunikation zwischen Kontrollflüssen
  - aber: kein gegenseitiger Schutz; ggf. Synchronisation bzgl. Speicher
- Thread hat weniger Zustandsinformation als ein Prozess
- Kontextwechsel daher i.a. wesentlich schneller
  - kein Umschalten des Adressraumkontexts
  - Cache und Translation Look Aside Buffer (TLB) bleiben “warm“
  - ggf. Umschaltung ohne aufwendigen Wechsel in privilegierten Modus

Wozu Multithreading bei Client-Server-Middleware?

- *Server*: quasiparallele Bearbeitung von RPC-Aufträgen
  - Server bleibt ständig empfangsbereit



- *Client*: Möglichkeit zum „asynchronen RPC“
  - Hauptkontrollfluss delegiert RPCs an nebenläufige Threads
  - keine Blockade durch Aufrufe im Hauptfluss
  - echte Parallelität von Client (Hauptkontrollfluss) und Server

DCE-Threadkonzept

- Grössere Zahl von *C-Bibliotheksfunktionen*
  - Erzeugen, Löschen von Threads
  - Synchronisation durch globale Sperren, Semaphore, Bedingungsvariablen
  - warten eines Threads auf ein Ereignis eines anderen Threads
  - wechselseitiger Auschluss mehrerer Threads (“mutex”)
  - nebenläufige Signalverarbeitung und Ausnahmebehandlung

- Pro Adressraum existiert ein eigener *Thread-Scheduler* mit wählbarer Strategie
  - verschiedene Schedulingstrategien wählbar (z.B. FIFO, Round Robin)
  - wahlweise Verwendung von Zeitscheiben (“präemptiv“)
  - wahlweise Berücksichtigung von Prioritätsstufen

Problematik von DCE-Threads

- Aufrufe des Betriebssystem-Kerns sind i.a. problematisch

  a) *nicht ablaufinvariante* (“non-reentrant”) Systemroutinen

  - interne Statusinformation, die ausserhalb des Stacks der Routine gehalten wird, kann bei paralleler Verwendung überschrieben werden
  - z.B. *printf*: ruft intern Speichergenerierungsroutine auf; diese benutzt prozesslokale Freispeicherliste, deren “gleichzeitige” nicht-atomare Manipulation zu Fehlverhalten führt
  - “Lösung”: Verwendung von “Jacket-Routinen” (wrapper), die gefährdete Routinen kapseln und Aufrufe wechselseitig ausschliessen

  b) *blockierende* (“synchrone”) Systemroutinen

  - z.B. synchrone E/A, die *alle* Threads des Prozesses blockieren würde statt nur den aufrufenden Thread
  - “Lösung”: Verwendung asynchroner Operationen zum Test auf mögliche Blockaden

---

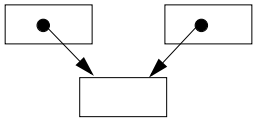
- Prinzipielle Probleme der Thread-Verwendung:
  - fehlender gegenseitiger Adressraumschutz --> schwierige Fehler
  - Stackgrösse muss bei Gründung i.a. statisch festgelegt werden --> unkalkulierbares Verhalten bei Überschreitung
  - von asynchronen Meldungen (“Signale”, “Interrupts”) an den Prozess soll i.a. nur ein einziger (der “richtige”) Thread betroffen werden
  - knifflige Synchronisation --> Deadlockgefahr

DCE-RPC

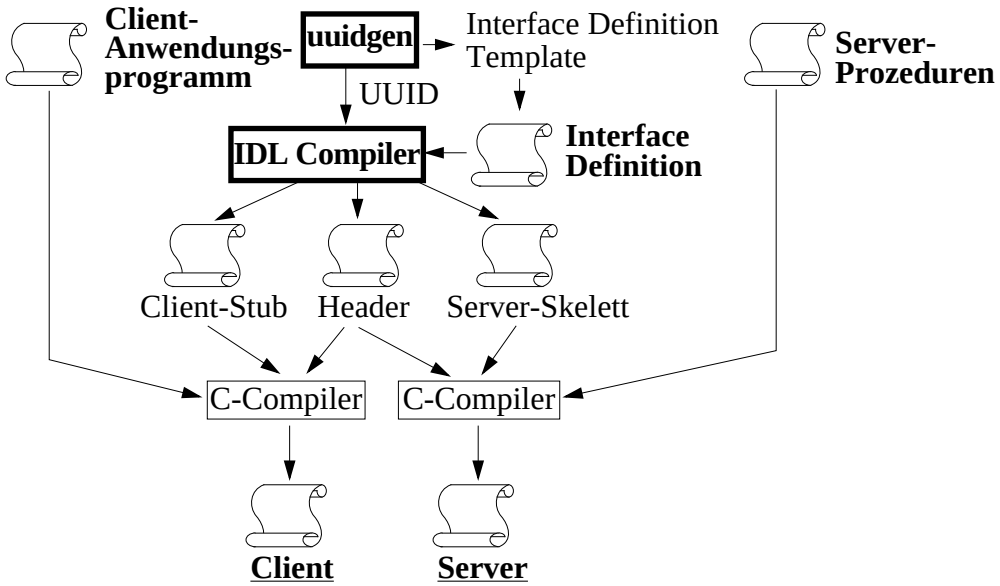
- Weder kompatibel zu Sun-RPC noch zur OSI-Norm
- Ein- und Ausgabeparameter: out, in, in/out
- Nahezu beliebige Parametertypen
  - alle C-Datentypen ausser Prozeduradressen

  - auch verzeigerte Strukturen, dynamische arrays
  - Zeiger werden automatisch dereferenziert und als Wert übergeben; jedoch Vorsicht bei Aliaszeigern!

    
- Automatische Formatkonvertierung zwischen heterogenen Rechnern
  - Prinzip: “Receiver makes it right”
- Beschreibung der Schnittstelle durch deklarative Sprache IDL (“Interface Description Language”)
  - analog, aber nicht identisch zu Sun-RPC
  - IDL-Compiler (entspricht etwa rpcgen bei Sun-RPC) erzeugt Stubs für Client und Server, in denen u.a. die Konvertierung erfolgt

DCE: Erzeugen von Client- und Server-Programmen

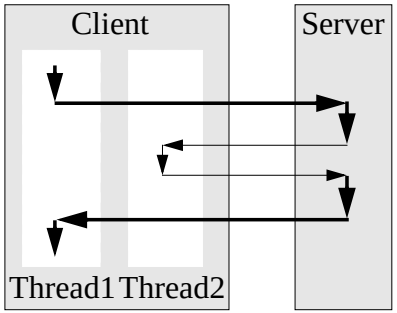


- UUID (Universal Unique Identifier) ist eine aus Uhrzeit und Rechnerkennung generierte systemweit eindeutige Kennung der Schnittstelle

DCE-RPC: Besonderheiten

- *Asynchrone Aufrufe* durch explizite parallele Threads
  - Kritik: umständlich, Threads sind potentiell fehleranfällig

- *Rückrufe* (“call back RPC”)
  - temporärer Rollentausch von Client und Server
  - um evtl. bei langen Aktionen Zwischenresultate zurückzumelden
  - um evtl. weitere Daten vom Client anzufordern
  - Client muss Rückrufadresse übergeben



- *Pipes* als spezielle Parametertypen
  - sind selbst keine Daten, sondern ermöglichen es, Daten stückweise zu empfangen (“pull”-Operation) oder zu senden (“push”)
  - evtl. sinnvoll bei der Übergabe grosser Datenmengen
  - evtl. sinnvoll, wenn Datenmenge erst dynamisch bekannt wird (z.B. Server, der sich Daten aus einer Datenbank besorgt)

- *Context-handles* zur aufrufglobalen Zustandsverwaltung
  - werden vom Server dynamisch erzeugt und an Client zurückgegeben
  - Client kann diese beim nächsten Aufruf unverändert wieder mitsenden
  - Kontextinformation zur Verwaltung von Zustandsinformation über mehrere Aufrufe hinweg z.B. bei Dateiserver (read; read) sinnvoll
  - Vorteil: Server arbeitet “zustandslos“

DCE-RPC: Probleme

Zum Beispiel:

My server gets a stack error when sending large objects. How can I avoid this?

Each thread in a process is assigned a fixed area for its procedure-call stack. The stubs normally marshall and unmarshall parameters in space allocated on the thread's stack. If the parameters are large, the stack size may be exceeded. In most thread implementations, the stack size cannot be increased after the thread is created. For threads created explicitly by your application, you can adjust the size of the thread stack by setting an attribute before calling pthread_create(). However, server threads are created automatically, so that method won't work; instead, call rpc_mgmt_set_server_stack_size() before starting the threads with rpc_server_listen().

Another possibility is to use the "heap" attribute to have some parameter types marshalled on the heap instead of the stack.

You should know that current implementations of the IDL compiler generate recursive code to marshall linked lists. Therefore, passing a long linked list may cause stack overflow due to all the recursive calls.

DCE-RPC: Anmeldung von Diensten

- Ein Dienst muss mittels mehrerer Systemaufrufe an drei Stellen bekannt gemacht werden
  - dazu gehört stets auch die Bekanntgabe der vom IDL-Compiler erzeugten und registrierten Dienstschnittstelle

1) "Exportieren" des Dienstes durch Anmeldung beim Directory Service der eigenen Zelle
   - Bekanntgabe der Adresse der Server-Maschine
   - ermöglicht es Clients, den Server zu lokalisieren

2) Adresse des Dienst-Prozesses ("endpoint") in eine "endpoint-map" der Server-Maschine eintragen
   - Endpoints entsprechen Ports bei TCP/IP
   - Map wird auf jedem Rechner von einem RPC-Dämon verwaltet

3) Registrieren beim lokalen RPC-Laufzeitsystem
   - damit können eintreffende Aufrufe an den zuständigen Dienstprozess weitergeleitet werden ("dispatching")
   - Angabe, wie viele Aufrufe serverseitig gepuffert werden sollen

- Schliesslich teilt der Dienst dem RPC-Laufzeitsystem mit, dass er bereit ist, Aufrufe entgegenzunehmen ("listen")
  - Angabe, wieviele Aufrufe maximal gleichzeitig bearbeitet werden können --> automatisches Erzeugen von Threads

Bindevorgang beim DCE-RPC

- Binden = (dyn.) Zuordnung von Client und Server
- Bindevorgang wird eingeleitet durch RPC-Aufruf:

1) RPC-Laufzeitsystem des Client stellt fest, dass Prozedur nicht lokal verfügbar ist
2) Befragung des Cell Directory Services (CDS)
3) CDS liefert Netzadresse der Server-Maschine
4) Client wendet sich an den RPC-Dämon der Server-Maschine
5) Client erhält dortigen Endpoint des Dienstes

- *zweiphasiger Ablauf* vorteilhaft, da Netzadressen von Services i.a. stabil sind, während sich Endpoints i.a. nach Neustart eines Rechners ändern

---

- Statt des o.g. *automatischen Bindens*, das für den Client transparent abläuft, ist auch *explizites Binden* möglich:
  - umständlicher, aber flexibler
  - z.B. programmierte Auswahl eines Backup-Servers, wenn Bindevorgang mit Primärserver unmöglich
  - z.B. explizite Auswahl eines Servers einer Gruppe (Lastausgleich etc.)

- Dienste haben eine *Hauptversion* und eine *Unterversion*
  - wird beim IDL-Compilieren angegeben, z.B. “3.2”
  - beim Binden wird automatisch überprüft:
    - Hauptversion.Client = Hauptversion.Server ?
    - Unterversion.Client ≤ Unterversion.Server (Aufwärtskompatibilität!) ?

DCE-RPC: Semantik

- Semantik für den *Fehlerfall* ist wählbar:

(a) *at most once*
- bei temporär gestörter Kommunikation wird Aufruf automatisch wiederholt; eventuelle Aufrufduplikate werden gelöscht
- Fehlermeldung an Client bei permanentem Fehler
- ist default

(b) *idempotent*
- keine automatische Unterdrückung von Aufrufduplikaten
- Aufruf wird ein-, kein-, oder mehrmals ausgeführt
- effizienter als (a), aber nur für wiederholbare Dienste geeignet

(c) *maybe*
- wie (b), aber ohne Rückmeldung über Erfolg oder Fehlschlag
- noch effizienter, aber nur in speziellen Fällen anwendbar

- Optionale *Broadcast*-Semantik
  - Nachricht wird in einem LAN an mehrere Server geschickt
  - RPC ist beendet mit der ersten empfangenen Antwort

DCE: Sicherheit

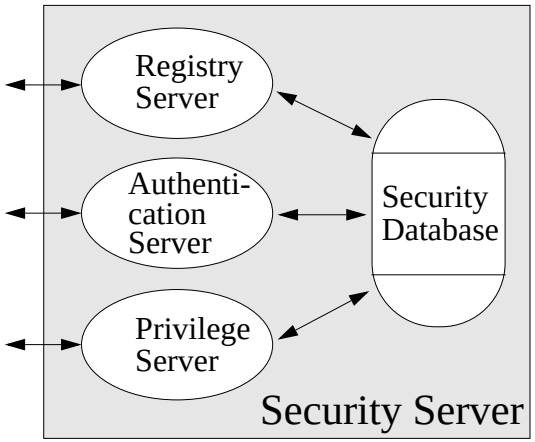
- Verwendung des Kerberos-Protokolls
  - Vertraulichkeit durch Sitzungsschlüssel (--> DES)
  - gegenseitige Authentifizierung
  - selektive Autorisierung von Clients für bestimmte Dienste
  - Schlüsselverwaltung
  - zusätzlich auch asymmetrische Verfahren (“public key”)

- Wählbare Sicherheitsstufen bei der Kommunikation
  - Authentifizierung nur bei Aufbau der Verbindung (“binding”)
  - Authentifizierung pro RPC-Aufruf
  - Authentifizierung pro Nachrichtenpaket
  - Zusätzlich Verschlüsselung jedes Nachrichtenpaketes
  - Schutz gegen Verfälschung (verschlüsselte Prüfsumme)

- Autorisierung ist mittels Zugriffkontroll-Listen realisiert
  - es gibt zahlreiche verschiedene Typen von Rechten
  - Gruppenbildung von Benutzern / Clients möglich
  - ACL-Manager bei den Servern verwaltet lokale Kontroll-Listen
  - Clients schicken eine verschlüsselte, authentische und gegen Replays gesicherte Repräsentation ihrer Rechte mit jedem Aufruf mit (PAC = Privilege Attribute Certificate); wird vom ACL-Manager überprüft

- Werkzeuge zur Systemadministration
  - Eintragen / Ändern von Rechten etc.
  - Installation zellübergreifender Sicherheitsdienste
  - hierzu spezieller “Registry-Server”

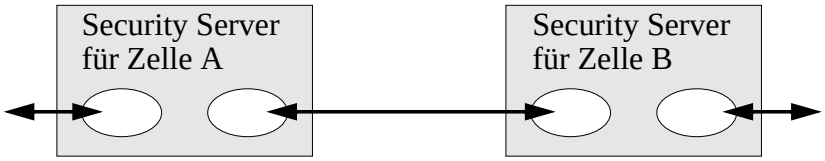
DCE-Sicherheitsdienste



- *Registry Server:* Verwaltung von Benutzerrechten; Dienste für Systemverwaltung
- *Datenbasis* enthält private Schlüssel (u.a. Passwörter in verschlüsselter Form...)
- *Privilege-Server* überprüft Zugangsberechtigung; u.a. bei login

- Sicherheitsdienst kann *repliziert* werden, um hohe Verfügbarkeit zu erreichen
  - nur Primärkopie kann Daten aktualisieren, Replikate sind “read only”
  - Primärkopie aktualisiert gezielt die Replikate

- *Zellenübergreifende Sicherheitsdienste:*



- ein Security Server A nimmt gegenüber einem Security Server B eine Clientrolle ein (“vertritt” die Clients seiner Zelle)
- ein Security Server besitzt im Gegensatz zu anderen Clients nicht einen einzigen geheimen Schlüssel, sondern es werden paarweise spezifische Schlüssel (“Surrogate”) vereinbart

Weitere DCE-Komponenten

- Cell Directory Service (CDS)
  - realisiert Zuordnung von Namen und Adressen
  - verwaltet Namen (mit Attributen) einer Zelle
  - Beispiel für Attribute: *Name, Standort, Typ* eines Druckers
  - Replikation (zwecks Fehlertoleranz) möglich (dabei “Konvergenzlevel” einstellbar)
- Global Directory Service (GDS)
  - Bindeglied zwischen verschiedenen CDS
  - hierarchischer Namensraum

Namensverwaltung

- Distributed File System (DFS)
  - ortstransparenter Dateizugriff
  - Caching beim Client steigert Effizienz (“Session-Semantik”)
  - mehrere Read-only-Replikate möglich
  - Unterstützung von Recovery, Migration und Backup
  - Synchronisation gleichzeitiger Zugriffsversuche
  - Gruppierung durch “File Sets“ (Gruppen von Dateien, die zusammen gelagert werden sollten)
  - nutzt DCE-RPC
- Distributed Time Service (DTS)
  - Synchronisationsprotokoll zwischen mehreren lokalen Zeitservern
  - Einbeziehung externer Zeitgeber (z.B. Funk- und Atomuhren)
  - Kopplung mit NTP-Protokoll möglich

DCE: Pragmatisches

Es gibt verschiedene Administrationstools

- Anzeigen und verändern von Information
- command line interface oder graphische Benutzungsoberfläche

---

Kritik an DCE: Komplexität

- Funktionsfülle (> 200 Funktionen)
  - wann benutzt man was?
  - Problem der wechselseitigen Beeinflussung („feature interaction“)
  - Semantik bei Kombination verschiedener Mechanismen u.U. unklar
- Grösse
- mangelnde Effizienz