3. CORBA referencer og Naming Services

- Objekt-referencer (og narrowing)
- Naming service (herunder Java RMI)
- By-value og by-reference
- Kode-eksempel

Objekt-referencer

Det er objekt-reference man arbejder på client-side. Når man kalder en metode på referencen, sørger ORB'en for at route kaldet over på det korrekte CORBA-objekt på serveren.
Generelt bruges en objekt reference til at **lokalisere** CORBA-objekter. **Kender typen** på CORBA objektet der refereres til. Skjuler at der forbindes til en server. *Location transparency*.

- **Object reference instance:** Dette er udgaven, klienten ser.
- **Interoperable Object Reference (IOR):** Object referencen som byte stream (fx ved marshalling af en object reference som parameter). Består af:
    - *RepositoryId:* Navnet på typen ud fra IDL.
    - *Profiles:* Indeholder information om, hvor et objekt befinder sig, og hvordan man får fat i det. Fx IIOP-profil, som består af host, port og object_key (som fortæller, hvor på serveren man finder objektet – typisk POA med ObjectId). Kan håndtere flere profiler for at man fx kan vælge mellem flere forskellige servere (ved replication).
- **Stringified Object Reference:** IOR som tekststreng. Starter med IOR: og et flag, som angiver little-/big-endian. Kan gemmes i en fil og deles via fx en web-server. Hver byte i IOR → hex.

Man skal konvertere object-referencen til en derived type. To udfordringer: *svært at cast i nogle sprog + ORB'en skal kende nedarvningshierarkiet.* Her bruges blot **narrowing**, hvilket sikrer os, at det automatisk sker. Sikrer også type safety.

Naming Services

Hvert objekt har et unikt ID, som ikke nødvendigvis forstås af mennesker, men forstås af servere og klienter. Vi kan spørge en NameService efter et objekt.

1. Klient spørger NameService (kendt – givet som fx stringified IOR (-ORBInitRef NameService=IOR:0101…) eller Object URL (corbaloc::host:port/NameService) i parameter) efter objekt.
2. Får en objekt-reference, som der kaldes metoder på. ORB'en sender kald videre til CORBA-objektet.

Naming Services er bygget hierarkisk op – lidt ligesom DNS.

Java RMI: Bruger udelukkende en RMI Registry (minder om Naming Service), hvor man slår objekter op vha. host, port og service navn (objektet): rmi://host[:port]/ServiceName. Her får man også en stub at arbejde på, hvor RMI sørger for at route kald til servere.

By-value og by-reference

- **Structs** overføres by-value. Kun værdierne overføres. Når de kommer tilbage er de ikke ens mere. Fylder mindre at overføre.
- **Valuetypes** overføres by-value. Indeholder også metoder, så de kan lidt mere. Dog skal metoderne implementeres lokalt på klient, da man ikke arbejder på CORBA-objekt, men udelukkende lokalt. Ellers som struct.
    - Bruges for at undgå roundtripping-anti-pattern (fx spørge et interface om data flere gange).
    - Bruger *factory* til at kreere objekter server-side. Nedbringer antallet af kørende CORBA objekter på serveren.
- **Interfaces** overføres by-reference. Metoder udføres her server-side.

SE KODE

Vi kunne have valgt at lave et hierarki på Naming Servicen, således at der var en NamingContext til Doctors og en til Patients, hvilket ville tillade en mange-til-mange-binding i forhold til den nuværende én-til-mange.