Klient-server og tre-lags-arkitekturen
Version 1.0

Henrik Bærbak Christensen
Datalogisk Institut
Aarhus Universitet

September 2011

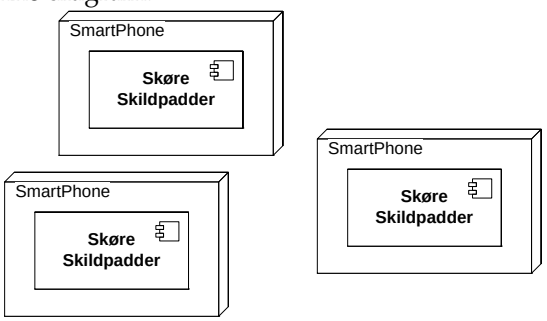
Denne note beskriver de arkitekturer, *klient-server* og *tre-lags-arkitekturen*, som it systemer med mange brugere, der deler information, er bygget op omkring. Noten er skrevet til brug i det gymnasiale forsøgsfag i informationsteknologi.

1 Fra en simpel arkitektur til en tre-lags-arkitektur

Du og et par af dine venner er dygtige til at programmere, og I har lavet gratis spillet "Skøre Skildpadder", som kører på smartphones. Spillet betstår af en lang række baner, som de skøre skilpadder skal klare for at score point og komme til næste bane.

Spillet bliver en stor success, og Danmark rundt dyster folk energisk om at opnå high score i de forskellige baner: "Ha ha, jeg kom op på over 71300 point i bane 2-13!" bliver der hørt i krogene på gymnasierne.

Informationsteknologisk set har du og dine venner lavet et program, der kører isoleret på hver enkelt smartphone. Det kan vi beskrive ved hjælp af denne diagram.



Hver "kasse" symboliserer en computer (her en smartphone), og programmet "Skøre skilpadder" er symboliseret ved et rektangel med et

ikon. Der er kun tegnet tre computere her, men selvfølgelig kan der være mange, der på samme tid spiller skøre skildpadder[1].

Det giver jer en ide—hvorfor kan man ikke se sine venners high score, når man har gennemført en bane? Altså, når man gennemfører en bane, så kan man se, hvor mange point man har fået, se hvad verdensrekorden er i øjeblikket, og se hvem der har den. Og hvis du slår den—ja så kommer dit navn til at stå som verdensrekordindehaver. Sejt!

Det giver bare nogle informationsteknologiske udfordringer, nemlig "hvordan finder vi ud af, at Mathildes high score på 93400 point i bane 3-12 er den højeste i verden?" og "hvordan giver vi lige Mikkels telefon besked om, at rekorden på 93400 point på bane 3-12 er opnået af Mathilde?"

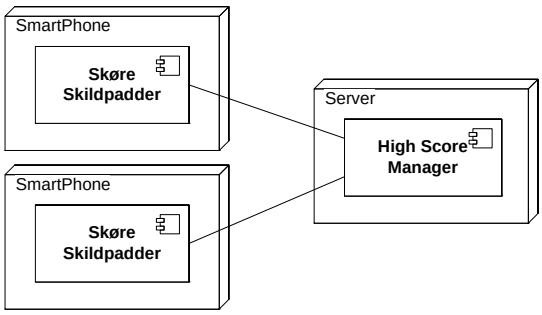
For at se, hvordan I kan programmere en løsning på den udfordring, kan vi som tankeeksperiment tænke over, hvordan vi kunne gøre det *uden brug af it*. Forestil dig, hvordan det kunne gøres, hvis det bare drejede sig om at holde styr på banernes high score på et enkelt gymnasium. Her kunne I afsætte en tavle i et fællesrum til "skøre skilpadder high scores." Der står alle banenavne, deres nuværende high score, og hvem der har opnået den: "Bane 3-12: Mathilde med 93400 point." Så kan Mikkel komme forbi, se tavlen, og se om han kan slå den. Hvis han gør, ja, så går han hen til tavlen, sletter Mathildes high score og skriver sin egen istedet.

Informationsteknologisk løser man problemet efter samme model: man køber en speciel computer, som kører et program, hvis eneste formål er at modtage og opbevare high score information—ligesom tavlen i fællesrummet. I diagrammet nedenfor er det symboliseret ved "High score manager" programmet.

En sådan central computer med programmer til at modtage og lagre data kalder man en *server*. Hver gang en spiller afslutter en bane i spillet på sin smartphone, tjekker spillet, om det er en ny high score og sender så informationen til serveren. Altså skal programmet nu en ekstra ting: både sørge for et sjovt spil, men også kontakte og sende information til serveren. På samme måde kan skøre skilpadde kontakte high score manageren på serveren og bede om high score på en bestemt bane.

Denne nye rolle kalder man, at skøre skilpaddeprogrammet er *klient* (*client* på engelsk), og man taler om en *klient-server*-arkitektur.

[1] Notation er *Unified Modeling Language*[Fowler, 2004] og diagrammet kaldes et *deployment diagram*, da det viser, hvilke programmer der kører (engelsk: "deployed") på hver computer.



Bemærk at der nu er tegnet en streg mellem skøre skilpaddeprogrammet og high score manager programmet, som symboliserer, at der sendes information frem og tilbage mellem de to programmer. Bemærk også, at der ingen streger er mellem spilleprogrammerne—al information går igennem serverprogrammet.

En servercomputer skal, ligesom tavlerne, "anbringes" så alle computere har adgang til den. Det vil sige, at den skal køre døgnet rundt, være koblet til internettet og have en "adresse", som alle skøre skilpaddeprogrammer kender.

Klient-server-arkitekturen er vidt udbredt, når et stort antal klienter har brug for at se den samme information. En af de bedst kendte eksempler er world-wide-web sider, hvor selve sidernes indhold ligger på en web-server. Din web-browser er så klient, som kan kontakte og hente hjemmesiderne og vise dem på din maskine.

Så, du og dine venner programmerer spillet om, så det bliver klient, altså henter high score fra serveren, hver gang man går til en ny bane. Hvis man slår high score, så sendes ens navn og pointtal tilbage til serveren, så alle andre, der derefter spiller banen, vil se den nye high score.

OK, så langt så godt. Nu bliver programmet en verdensomspændende dille, men det skaber nogle nye udfordringer. Den første udfordring er, at vores server bliver bombarderet med forespørgsler om high scores—og selvom vi har købt en stor stærk maskine, så kan den slet ikke følge med de 10.000 forespørgsler, der kommer hvert minut til den. Den anden udfordring er, at vi vil tilbyde, at man laver spillergrupper, som kan konkurrere indbyrdes—f.eks. kan alle spillere på Egå gymnasium tilmelde sig "Egaa Gym" gruppen og nu se både verdensrekorden samt Egaa gymnasium rekorden for hver bane. Man kan tilmelde sig flere grupper, så man kan konkurrere med venner, familie, klassekammerater, vennerne fra fodboldklubben, osv.

Og hvordan programmerer I så det? Lad os igen prøve et tankeeksperiment, hvor vi (i hvert fald delvist) løser problemet uden it. Hvor I før havde en enkelt pointtavle i gymnasiets fællesrum, så bliver I nu nødt til dels at have mange pointtavler (en for fodboldklubben, en for Mathildes familie, en for Egaa gymnasium, osv.), dels bliver I nødt til at samle tavlerne et sted, hvor alle har adgang (ikke kun de, som går på Egaa gymnasium), og dels bliver I nødt til at ansætte nogle folk til at holde styr på alle tavlerne. De ansatte skal lave nye

tavler for nye grupper, opdatere tavlerne, når folk laver en ny high score, og tage imod alle de mange forespørgsler fra spillere, der ustandeligt spørger, hvad stillingen er på alle de tavler, de selv er interesserede i. Med andre ord en hel lille "pointtavle håndteringsorganisation."

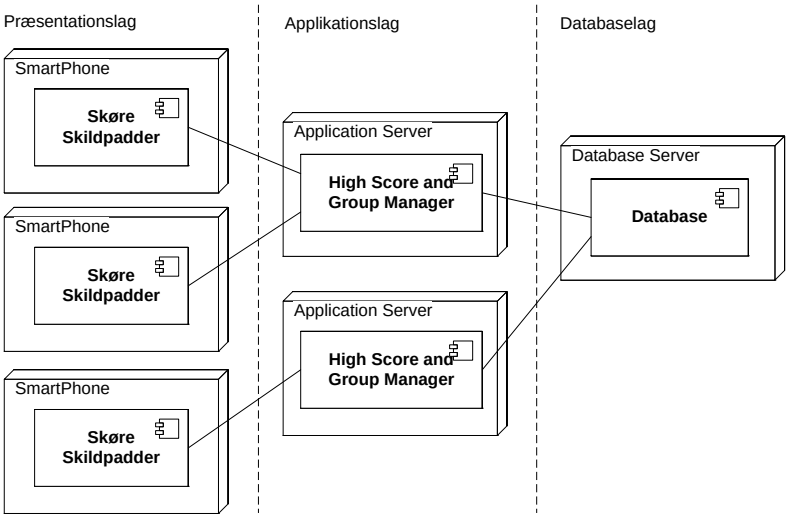
Selve det at opdatere pointtavlerne er ikke nogen helt simpel opgave, som følgende scenarie viser. Lad os forestille os, at Mathilde har scoret 93400 point på bane 3-12 og at hun er med i grupperne "Mathildes familie", "Egaa Gym", og "Mathildes fodboldklub"—og at den hidtige high score i de tre grupper henholdsvis er 73800, 103600, og 12400 point. Når Mathilde kommer til "pointtavle håndterings organisationen" med sin nye score, ja, så skal hendes navn og score sættes ind på tavlerne for hendes familie og for fodboldklubben, men naturligvis *ikke* på Egaa gymnasietavlen. Med andre ord skal der foretages en del beregninger af de ansatte, førend de eventuelt skal opdatere nogle af tavlerne.

Se, denne model vil i princippet kunne virke, men selvfølgelig være alt for omstændelig i praksis! Oversætter I imidlertid modellen til it, giver det meget mere mening: det er meget lettere at sende information frem og tilbage mellem computere end fysisk at skulle gå hen til en "pointtavleorganisation", og fordi de manuelle procedurer, som de ansatte i vores tankeeksperiment laver, istedet kan udføre billigt af nye programmer.

Informationsteknologisk er denne model en videreudvikling af klient-server-arkitekturen og kaldes en *tre-lags-arkitektur* (*three tier architecture*). Man splitter serveren op i to dele: en del som er ansvarlig for effektiv lagring, søgning, og opdatering af data, *databaselaget*, og en anden del som er ansvarlig for at foretage de beregninger og procedurer som vi ønsker, *applikationslaget*. Klienten, som er ansvarlig for at vise informationen og at interagere med brugeren, kalder man ofte for *præsentationslaget*.

I vores eksempel vil databaselaget altså indeholde information om, hvilke grupper/pointtavler der findes, hvilke personer der er med i hvilke grupper, og selvfølgelig hvilke high scores der er i de enkelte grupper. Applikationslaget vil være ansvarlig for at oprette grupper, modtage scores fra klienterne og beregne, hvilke pointtavler der skal opdateres og endelig sende scores ud til klienterne igen. Og endelig er præsentationslaget på klienterne ansvarlig for at vise high scores samt sende og modtage dem til og fra applikationslaget.

Vi kan skitsere en sådan arkitektur således:



Typisk vil man i store systemer bruge mange computere for at køre disse lag for at kunne klare store mængder forespørgsler fra mange klienter. Typisk vil databaselaget køre på en meget kraftig computer, og applikationslaget køre på en hel mængde relativt kraftige computere. Derfor vil man ofte høre begreberne "databaseserver" og "applikationsservere" om de computere og det software, som danner henholdsvis databaselaget og applikationslaget. Og endelig kører præsentationslaget på utallige computere som f.eks. smartphones, laptops, og/eller desktop maskiner.

Når Mikkel starter på en bane, kontakter hans skildpaddeprogram nu en applikationsserver, der finder ud af, hvilke grupper han er med i og sender de relevante high scores. Det gør applikationsserveren på sin side ved at kontakte sin databaseserver. Når Mikkel så har vundet banen, bliver hans score sendt til applikationsserveren, som nu finder frem til, hvilke gruppers high score der skal opdateres. Kun hvis der faktisk er en ny high score i en af grupperne, vil applikationsserveren opdatere data på databaseserveren.

Fordelen ved denne arkitektur er, at

- den kan håndtere meget store mængder data og mange forespørgsler, da der kan være mange applikationsservere, og de derfor kan fordele arbejdet imellem sig
- kun hvis der er nye high scores i en gruppe, er der brug for at kontakte databaseserveren, som skal ændre data
- mange procedurer og beregninger ligger i programmer på applikationsserverne, og det vil sige, at hvis man finder en fejl i det program, der laver beregningen, ja så behøver man kun at installere nye programmer på applikationsserverne istedet for i alle klientprogrammerne.

Ulemper er der også, og det er prisen. Det er temmelig dyrt at have en hel flok computere kørende dag og nat. Typisk vil applikationsservere og database servere stå i specielle serverrum, hvor der er køling og specielt brandslukningudstyr, og backup af databaserne. Så små firmaer har som regel deres servere stående hos andre firmaer, *hosting firmaer*, som har specialiseret sig i netop at holde mange servere i kontinuert drift.

2 Den korte version

2.1 Klient-server-arkitektur (Client-server)

I en klient-server-arkitektur deler man et it-system op i en klientdel og i en serverdel.

Klienten er ansvarlig for

- præsentation af grafik, tekst, og information til brugeren
- interaktion med brugeren (museklik, tastatur, osv.)
- kommunikation af information til og fra serveren

Serveren er ansvarlig for

- opbevare og redigere information/data som deles af klienterne
- foretage simple procedurer og beregninger på informationen
- sende og modtage information til og fra klienterne

Et klient-server it-system har een server og mange klienter. Klienterne er de "aktive" forstået således, at det altid er dem, der starter en forespørgsel eller ændring af data. Serveren er "reaktiv" forstået således, at den ikke selv tager initiativ til noget, men venter på forespørgsler fra klienter. En forespørgsel vil så bevirke, at serveren udfører procedurer, ændrer data, og til sidst sender et svar tilbage til klienten. Information flyder mellem klienten og serveren via en netværksforbindelse, typisk internettet. Klienter kan ikke sende og modtage information fra andre klienter: alting sker via serveren.

En klient-server-arkitekturs fordele er

- Alle klienter ser det samme data. Det er nok den vigtigste egenskab: ændringer foretaget af en klient ses af alle andre klienter næste gang, de læser fra serveren. Med andre ord, så er der ikke brug for, at man kopierer data frem og tilbage mellem klienterne indbyrdes.
- Alle programmer, der ligger på serveren, kan ændres/opdateres, uden der skal ændres i de programmer, der kører på klienterne. Det gør det billigere at vedligeholde.

Ulempen er den øgede omkostning på grund af, at serverne skal køre i døgndrift, og den mere komplicerede programmering, der skal til for at lave klient og server programmerne.

2.2 Tre-lags-arkitektur (Three-tier)

I en tre-lags-arkitektur deler man et it-system op i tre dele: præsentations-, applikations- og databaselagene. Den er en specialisering af klient-server-arkitekturen på den måde, at server siden bliver delt op i yderligere lag for at forbedre ydelsen.

Præsentationslaget (klienten) er ansvarlig for

- præsentation af grafik, tekst, og information til brugeren
- interaktion med brugeren (museklik, tastatur, osv.)
- kommunikation af information til og fra applikationslaget

Applikationslaget er ansvarlig for

- foretage komplicerede procedurer og beregninger på informationen
- sende og modtage information til og fra klienterne og databaselaget

Databaselaget er ansvarlig for

- opbevare og redigere information/data som deles af klienterne effektivt
- foretage effektive søgninger i meget store datamængder
- sende og modtage information til applikationslaget

Et it-system med en tre-lags-arkitektur har mange klienter, et større antal applikationsservere og en enkelt eller få databaseservere. Ligesom for klient-server-arkitekturen er klienterne de aktive og applikations- og databaseserverne reaktive. Klienter kan ikke kommunikere indbyrdes eller med databaseserveren, kun med applikationsserverne.

En tre-lags-arkitekturs fordele er

- Alle de samme fordele som for klient-server-arkitekturen
- Kan håndtere meget store datamængder og/eller meget stort antal samtidige klienter, da der kan sættes et større antal applikationsservere op.
- Procedurer og beregninger i applikationslaget kan være langt mere komplicerede end for en typisk klient-server-arkitektur, da der er mere computerkraft til at foretage dem.

Ulempen er igen store driftsomkostninger og endnu mere kompliceret programmering for at håndtere det ekstra lag.

3 Eksempler på systemer

3.1 Netbank

Alle danske banker tilbyder i dag en netbankløsning til deres kunder. Via netbank kan mange almindelige bankforretninger, såsom overførsel af penge mellem ens konti, betaling af girokort, gennemsyn af kontobevægelser, osv., foretages derhjemme og stort set på alle tider af døgnet.

Netbank er et klassisk eksempel på en tre-lags-arkitektur.

- *Præsentationslaget* er et relativt simpelt program som kører i din webbrowser. Det kan ikke meget mere end vælge nogle procedurer (som f.eks. at se kontobevægelser, betale en regning, overføre penge mellem konti, osv.) og så vise resultatet af disse procedurer. Man taler ofte om en *tynd klient*, altså en klient, som kan meget lidt—de avancerede beregninger sker på server siden.
- *Applikationslaget* indeholder de programmer, der kan lave dine bankforretninger. Nogle er simple: at overføre penge fra en af dine konti til en anden er jo kun at lave nogle ændringer i de databaser, hvori dine kontioplysninger ligger. Andre er mere komplicerede: at betale et girokort eller overføre penge til en udenlandsk konto betyder, at der skal sendes informationer til andre banksystemer, og der skal reageres, når der senere kommer svar tilbage om at transaktionen gik godt eller eventuelt ikke kunne gennemføres.
- *Databaselaget* tager sig af at opbevare og tilføje alle bevægelser på alle konti i banken, personoplysninger, osv.

Da bankerne var nogle af de første til at tage it til sig (i gamle dage da det hed EDB: elektronisk databehandling) vil de fleste programmer der kører i applikations- og databaselaget køre på en enkelt meget stor computer, en *mainframe*. Mainframes er meget driftsikre og kan stadig køre programmer som er udviklet i sprog fra 1960. Da mange bankforretninger ikke har ændret sig fundamentalt siden dengang, er langt den billigste og mest driftsikre løsning forsat at benytte de gamle programmer.

Da klientprogrammet kører i din web-browser bliver det faktisk downloaded til din computer, hver gang du klikker på linket til din netbank (endnu en grund til at det skal være en tynd klient, som ikke tager for lang tid at downloade.)

3.2 World of Warcraft

World of Warcraft[2] er et MMORPG, *massive multiplayer online role playing game*, hvor spillere via deres avatar ("character") løser opgaver, kæmper, handler, og

[2]Registreret varemærke for Blizzard, *www.blizzard.com*.

bevæger sig i en virtuel verden sammen med andre spilleres avatarer. Adgangen til den virtuelle verden er betinget af, at man betaler et abonnement. World of Warcraft (WoW) havde i oktober 2010 over 12 millioner betalende abonnementer.

For at illusionen om, at mange spillere bevæger sig rundt i den virtuelle verden samtidigt, skal opretholdes, er der nogle ekstreme krav til den underliggende it-arkitektur. For at indse det, så forestil dig at to spilleres avatarer mødes til en kamp, og at de to spillere sidder ved hver sin maskine, den ene i Danmark, den anden i Rusland. Hvis den ene angriber og den anden forsvarer sig, så er det naturligvis meget vigtigt for, at spillerne synes, det går retfærdigt til, at spillet reagerer med det samme og reagerer korrekt, når man trykker på keyboardet (at undgå "lag"). Hvis for eksempel man ser sin avatar blive angrebet og man omgående taster en hotkey, som laver et godt forsvar, men forsvaret først bliver udført et sekund senere og dermed alt for sent, ja, så har man en dårlig oplevelse som spiller. Men informationen om, hvad den enkelte spiller foretager sig, skal jo formidles via serveren mellem de to computere, der står langt fra hinanden—og dermed er der en forsinkelse i signalet. Hvis så serveren ydermere er belastet af mange samtidige spillere, kan det gå galt, og informationerne bliver så forsinkede, at spillet slet ikke synes at reagere tidsnok på ens handlinger.

Blizzard har lavet en tre-lags-arkitektur som specifikt er gearet til at håndtere disse strenge krav[3].

Overordnet set er det en tre-lags-arkitektur.

- *Præsentationslaget* er det program man køber og installerer på sin computer. Det er et meget stort program (man kalder det ofte en *tyk klient*, fordi klient programmet i sig selv er komplekst og kan en masse.) som er ansvarlig for at tegne grafikken i høj opløsning og med mange billeder per sekund så man får en flydende oplevelse af at bevæge sig rundt i den virtuelle verden. Programmet indeholder endvidere al informationen om selve den virtuelle verden (landskab, træer, bygninger, veje, etc.) da disse datamængder er så store, at det vil tage alt for lang tid at sende dem fra serverne. Det betyder, at Blizzard kun kan lave ændringer i den verden, som du spiller i, når de kommer med opdateringer til klienten og nye expansion packs.

- *Applikationslaget* omfatter en stor mængde servere, som kaldes *realm-servere*. Realm betyder "rige" eller "kongerige", og når man opretter en avatar, så vælger man, at den skal "bo" i et bestemt rige—eller rettere al information om din avatar lagres på en ganske bestemt server. Det vil sige, at du kun kan møde andre spilleres avatarer, som ligger på den samme realm-server. Da der er sat en grænse for, hvor mange avatarer der er plads til på den enkelte server, begrænser Blizzard trafikken til den enkelte server, hvilket betyder mindre informationsmængder og dermed mere præcis styring af din avatar. Endvidere er realm-serverne spredt ud geografisk, så man bør vælge et realm, der er tæt på ens geografiske

[3]Den konkrete arkitektur er naturligvis en hemmelighed, som Blizzard ikke vil offentliggøre, så denne beskrivelse er blot et kvalificeret gæt.

region. Blizzards europæiske afdeling har servere i England, Tyskland, Frankrig, Spanien, og Rusland. En server, der er tættere på, kan svare hurtigere, fordi der er færre mellemstationer ("routere") mellem servere og dit klientprogram.

Ud over at videresende dine handlinger i det pågældende realm til de andre spillere, så tager realm-serverne sig også af login—Blizzard vil jo gerne sikre sig, at du har betalt dit abonnement, inden du bliver lukket ind.

- *Databaselaget* omfatter databaser over, hvilke abonnementer der er, deres brugernavne og passwords, og hvilke konto det enkelte abonnementet skal trækkes fra. Andre databaser har lister over, hvilke realm-servere er der tilgængelige i din geografiske region.

  Endvidere har realm serverne også tilknyttet databaser over, hvilke avatarer, der "bor" på dem, og al information omkring deres udstyr, våben, osv.

Da informationen omkring ens avatarer ligger på databaser tilknyttet realm-serverne, kan man altså fint have sin WoW-klient installeret på flere computere derhjemme og fint spille videre på ens avatar, ligegyldig hvilken computer man spiller på. Man kan også spille på en af sine kammeraters maskiner, bare man logger på med ens eget brugernavn og password. Hvis det var klientprogrammet (præsentationslaget), der var ansvarlig for dette, ville ens avatar jo være begrænset til en enkelt computer.

4 Øvelser

Brug tre-lags-arkitekturen til at beskrive f.eks. facebook, twitter, eller et andet socialt netværk, du kender. Hvilke programmer kører på, hvilke computere, og hvad er de enkelte lag/servere/klienter ansvarlige for at have af data og procedurer?

Forestil dig, at der er 100.000 klienter, som spiller skøre skilpadder samtidigt, og en af dem laver en ny high score på en bane. Hvordan skulle de 99.999 andre klienter se den, hvis der *ikke* var en server (altså, hvis vi ikke havde lavet en klient-server-arkitektur)? Prøv at beskriv en anden arkitektur, og hvordan informationen om high score skulle sendes frem og tilbage mellem klienterne for at alle "bliver enige" om, hvad der er high score.

Undersøg forskellen i arkitektur mellem din computers filsystem og så DropBox systemet.

Litteratur

[Fowler, 2004] Fowler, M. (2004). *UML Distilled–A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 3 edition.