Arkkitehtuurityylejä ja ratkaisumalleja

Luento 4

Oppimistavoitteet

- Esimerkkejä yleisesti käytetyistä arkkitehtuurisista ratkaisumalleista ja tyyleistä
- Muutama käydään läpi perusteellisemmin (MVC, tietovuotyylit, kerrosarkkitehtuurit)

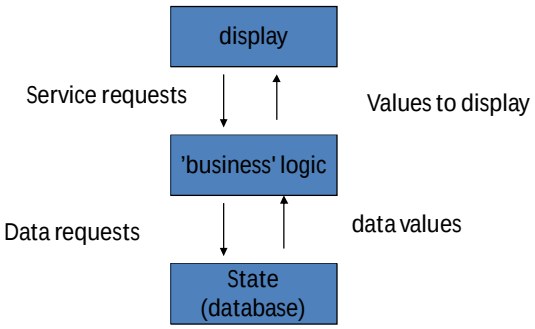
Tyylien ja ratkaisumallien käytöstä

- Platoninen vs. ”ruumiillistunut” tyyli/malli
  - Platoninen tyyli/malli on ideaalinen kuvaus arkkitehtuuriratkaisusta
  - Käytännön toteutuksissa esiintyvät tyylien/mallien instanssit (”ruumillistumat”) ovat harvoin täysin ”puhtaita”, eli niissä on tehty jotain (yleensä perusteltuja) poikkeuksia ratkaisun rakenteisiin ja niitä koskeviin sääntöihin
  - Tyylit ja ratkaisumallit ovat yleistyksiä
  - On myös paljon sovellusalue- ja teknologiakohtaisia ratkaisumalleja, joilla on rajallisempi sovellusala

Esimerkkejä

ARKKITEHTUURISET RATKAISUMALLIT

Kolmitasoarkkitehtuuri (N-Tier)



Model-View-Controller (MVC) [malli-näkymä-ohjain]

- Sovellusalue: monimuotoisia käyttöliittymänäkymiä sisältävät interaktiiviset järjestelmät
- Motivaatio:
  - Ohjelmistolla on erilaisia käyttäjiä ja näillä erilaisia tarpeita käyttöliittymään liittyen
  - Samaa informaatiota pitää pystyä esittämään ja käsittelemään eri muodoissa
  - Käyttöliittymän tulisi olla helposti muutettavissa
- Krasner, G., Pope S: Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, *JOOP* Aug./Sep.,1988.

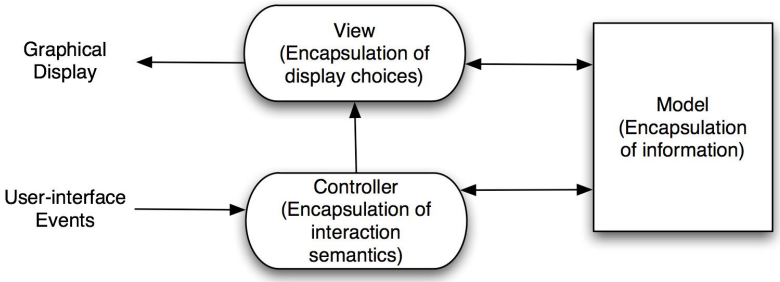
MVC

- Kolmenlaisia komponentteja
  - Malli-komponentit ovat vastuussa laskentaan liittyvän tiedon (tai tilan) säilytyksestä
  - Näkymä-komponentit ovat vastuussa tiedon esittämisestä käyttäjälle
  - Ohjain-komponentit ovat vastuussa käyttäjän ja järjestelmän välisen vuorovaikutuksen ohjaamisesta
    - Ottavat esimerkiksi vastaan hiiren näpäyksiä, näppäimen painamisia jne. ja muuntavat nämä mallikomponenttien tilaan vaikuttaviksi operaatioiksi

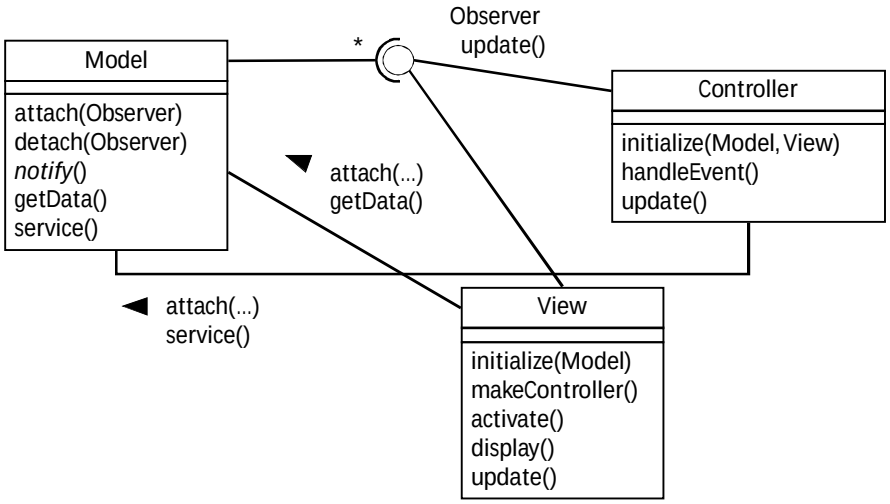
MVC

- Malli-komponentit eivät riipu mistään (tietystä) näkymä- tai ohjain-komponentista
- Muutokset malli-komponenteissa välitetään näkymille Tarkkailija-suunnittelumallin mukaisesti (eli niille näkymille, jotka ovat ilmaisseet kiinnostuksensa muutoksiin)

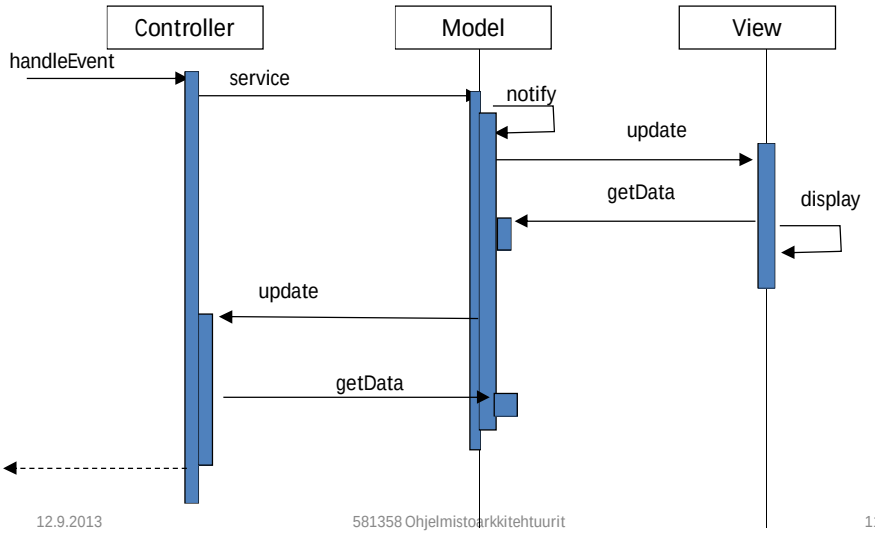
MVC



Variaatio MVC-mallista
(Buschmann & al.: Architectural patterns)



MVC - toiminta



MVC - arviointia

- Etuja
  - Useita näkymiä samaan tietoon
  - Näkymiä on helposti lisättävissä jopa dynaamisesti
  - Ohjaimen voi helposti vaihtaa
  - Voi toimia perustana sovelluskehykselle (Smalltalk)
- Ongelmia
  - Lisää mutkikkuutta
  - Yksinkertainen käyttäjätoimenpide saattaa aiheuttaa monia päivityksiä näkymään (päivityksen laajuutta voi olla vaikea kontrolloida)

Muunnelmia

- Monesti ei ole tarpeen erottaa *ohjainta* ja *näkymää*
  - Erityisesti, jos kullakin näkymällä olisi joka tapauksessa oma ohjain, jonka kautta voidaan suoraan manipuloida näkymää
  - → Päivitys näkymään (ja malliin)
  - → Malli ilmoittaa muutoksesta (muille) näkymille
- Ts. komponentti voi toimia sekä näkymä- että ohjainkomponenttina. Yhteen malliin voi liittyä monta <näkymä, ohjain> -paria.

Esimerkkejä

ARKKITEHTUURITYYLIT

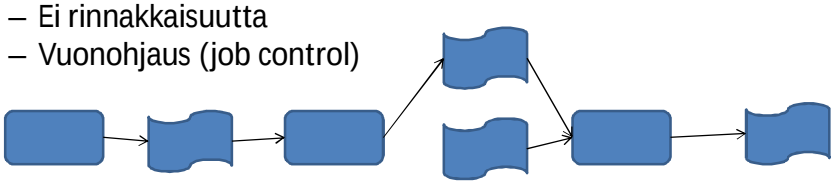
Tyylien luokittelua

- Eri lähteet luokittelevat tyylejä eri tavoin
- Seuraavassa noudatetaan kurssin edellisen oppikirjan (Taylor & al.) luokitusta (ei mitenkään kattava - luokat eivät pistevieraita)
  - Fairbanks ei luokittele tyylejä
- Tarkastellaan vain muutamaa esimerkkiä eri luokista
  - Tietovuopohjaiset
  - Kerrosrakenteiset
  - Tulkkaavat
  - Epäsuoran aktivointiin perustuvat

Tietovuopohjaiset tyylit

- *Data flow styles*
- Pääpaino siinä miten tieto liikkuu riippumattomien komponenttien välillä

- *Erätöiden sarja* (batch sequential)
  - Perinteinen eräajosovellus
  - Tiedon käsittely jakautuu vaiheisiin,
  - Lopputulos valmis kun kaikki vaiheet on suoritettu
  - Ei rinnakkaisuutta
  - Vuonohjaus (job control)



Tietovuopohjaiset tyylit

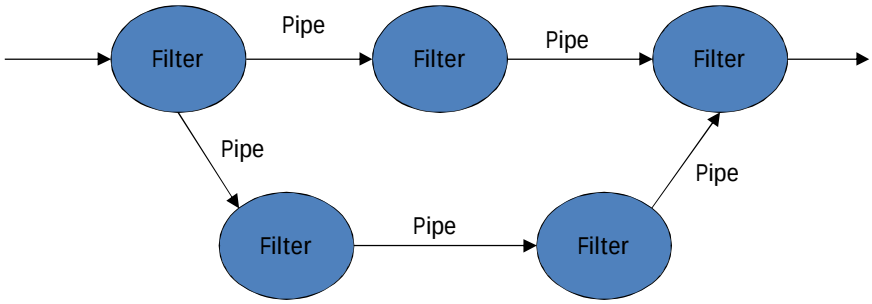
- Erätöiden sarja jatkuu:
  - Tieto liikkuu kokonaisuuksina vaiheelta toiselle esimerkiksi ajastetusti, vuonohjauskielen tai käyttäjän ohjaamana
  - Vaiheelle voi tulla useita syöttöaineistoja ja se voi tuottaa useita tulosaineistoja
  - Vaihe käsittelee syöttöaineistot kokonaisuudessaan ennen tulosaineiston luovutusta eteenpäin seuraavalle vaiheelle
  - Sovellusalueita:
    - Monivaiheiset tiedonjalostusprosessit, jossa aineistoa ei voi käsitellä osina
    - XML-muunnokset käyttäen dokumenttipuuta

Tietovuopohjaiset tyylit

- Väylät ja suodattimet (*pipes and filters*)
  - Koostuu *prosessointiyksiköistä* (filters) ja niitä yhdistävistä tietovirtaa kuljettavista *väylistä* (pipes)
  - Prosessointiyksiköt käsittelevät aktiivisesti tietoa
  - Väylät siirtävät tietoa eteenpäin passiivisesti
  - Prosessointiyksiköt ovat (”puhtaassa tapauksessa”) täysin itsenäisiä
    - Lukevat syötevirtaa ja tekevät sen perusteella laskentaa → lähettävät eteenpäin muokatun syötevirran

Tietovuopohjaiset tyylit

Väylät ja suodattimet jatkuu…



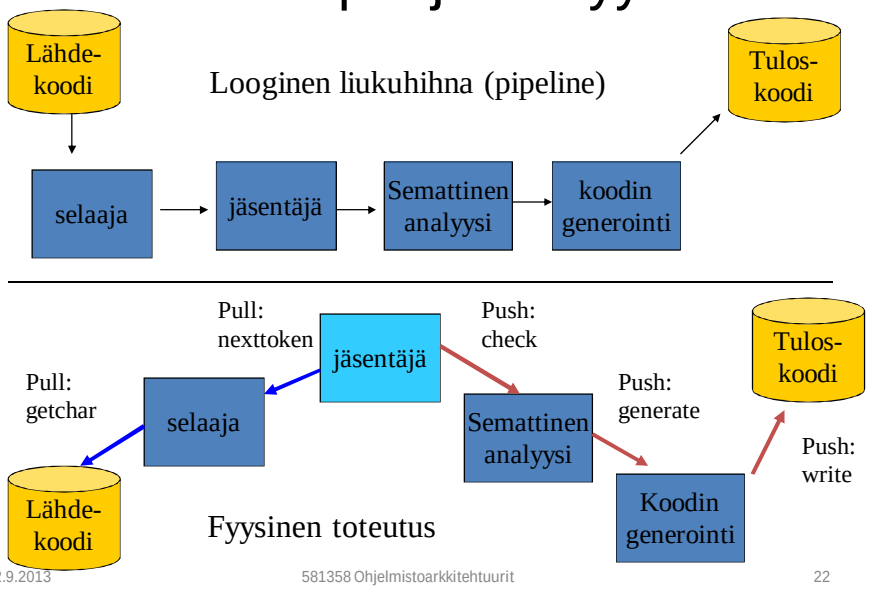
Tietovuopohjaiset tyylit

- Väylät ja suodattimet jatkuu…
- Kukin prosessointiyksikkö pitää voida toteuttaa itsenäisenä entiteettinä
  - Ei jaettua tilatietoa
  - Prosessointiyksikkö riippuu ainoastaan oman syötteensä muodosta
- Prosessointi tapahtuu yhdessä vaiheessa pala kerrallaan:
  - tietoalkion (esimerkiksi tekstirivin) prosessointi ei saisi riippua jonkin tulevan tietoalkion prosessoinnista
    - tällöin suodattimet voivat toimia rinnakkain
    - Unix-väyliin kytketyt käsittelyvaiheet ovat usein sellaisia, että rinnakkaisuus estyy – käsittely voi edetä vasta kun koko aineisto saatu, esim järjestäminen suodattimena
  - Tietoa voidaan puskuroida, mutta laajamittainen puskurointi rikkoo arkkitehtuurin perusajatuksen, ja vaikeuttaa esimerkiksi syötteen rinnakkaista prosessointia

Tietovuopohjaiset tyylit

- Väylät ja suodattimet jatkuu…
- *Liukuhihna* (pipeline architecture)
  - Erikoistapaus väylät ja suodattimet mallista: Ei haarautumisia
  - Helppo synkronoida, voi edetä
    - työntämällä (push)
      - prosessointiyksikkö kutsuu seuraavan yksikön palvelua ja välittää tietoa eteenpäin (alkaa alusta)
    - tai vetämällä (pull)
      - prosessointiyksikkö kutsuu edellisen yksikön palvelua (rekursiivisesti).

Tietovuopohjaiset tyylit



Kerrosarkkitehtuurit

- *Layered architectures*
- Kerrosarkkitehtuuri koostuu tasoista, jota on järjestetty jonkin abstrahointiperiaatteen mukaan nousevaan järjestykseen
  - Usein abstrahointi tapahtuu akselilla ihminen-laite
  - Usein kerrokset ovat luonteeltaan niin teknisiä, että eri abstraktiotasojen nimeäminen ja erottaminen käsitteellisesti toisistaan on hankalaa

Kerrosarkkitehtuurit

- Virtuaalikoneet (virtual machines) kerrosarkkitehtuurityyppinä
- Perusajatus:
  - Taso toimii virtuaalikoneena, joka tarjoaa palveluita ylemmän tason korkeamman abstraktion virtuaalikoneelle. Ylemmän tason tarjoamat palvelut toteutetaan välittömästi alemman tason palveluita hyväksikäyttäen.
  - Ajatus toteutuu vain *puhtaassa (strict) kerrosarkkitehtuurissa*

Kerrosarkkitehtuurit

- Käytännössä puhtaasta kerrosarkkitehtuurista voi olla kahdenlaisia poikkeamia:
  - Palvelukutsuja tehdään alemmasta kerroksesta ylempään kerrokseen
  - Palvelukutsu ohittaa kerroksia kulkiessaan ylhäältä alaspäin (? avoin kerrosarkkitehtuuri)
- Kerrosten ohittaminen on tarpeen useimmiten tehokkuussyistä (tai sitten välissä oleva kerros ei toteuta tiettyä palvelua, joka löytyy alemmalta kerrokselta)

Kerrosarkkitehtuurit

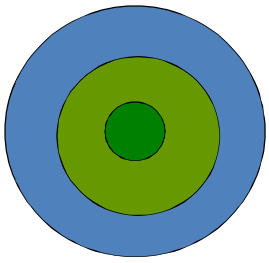
- Kerroksen ohittamisesta aiheutuva ongelma:
  - Tietty kerros tulee riippuvaiseksi myös muista kuin suoraan sen alapuolella olevasta kerroksesta
    → Kerroksen vaihtaminen hankalaa
- Palvelukutsun tekeminen alemmasta kerroksesta ylempään päin voi olla merkki vakavasta ongelmasta arkkitehtuurissa
  - Alempi kerros saattaa olla riippuvainen ylemmästä

Kerrosarkkitehtuurit

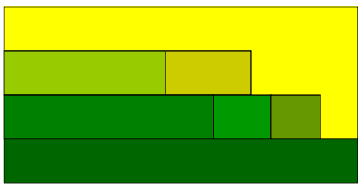
- Joskus alemman kerroksen on kuitenkin tarpeen kutsua ylemmän kerroksen koodia
  - Alemman kerroksen täytyy mukauttaa toimintaansa ylemmän kerroksen mukaan
  - Jotta alempi kerros ei tulisi (liian) riippuvaiseksi ylemmästä kerroksesta, voidaan hyödyntä *takaisinkutsuperiaatetta* (callback)
    - Alempi kerros tarjoaa rekisteröintioperaation, jonka avulla ylempi kerros rekisteröi takaisinkutsussa kutsuttavan koodin alemman kerroksen käyttöön

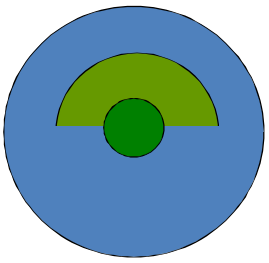
Kerrosarkkitehtuurit

Ei ohiteta kerroksia:



Ohitetaan kerroksia:

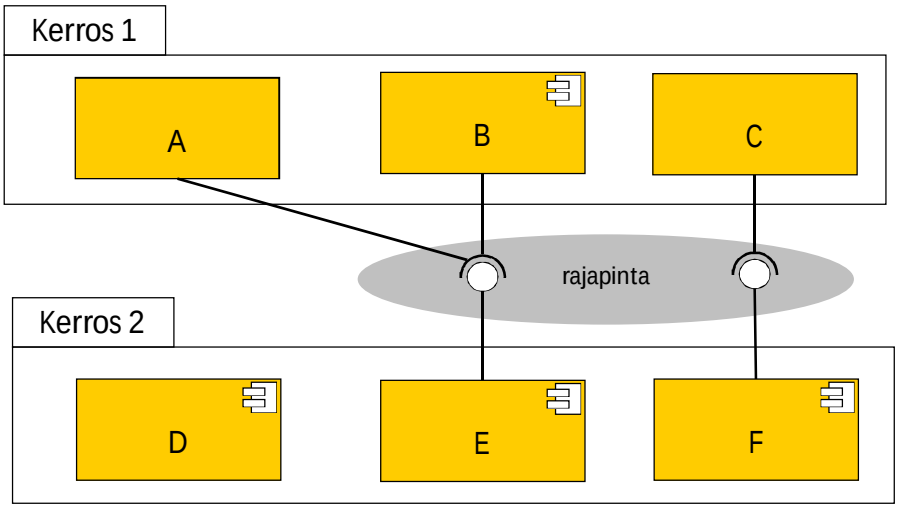




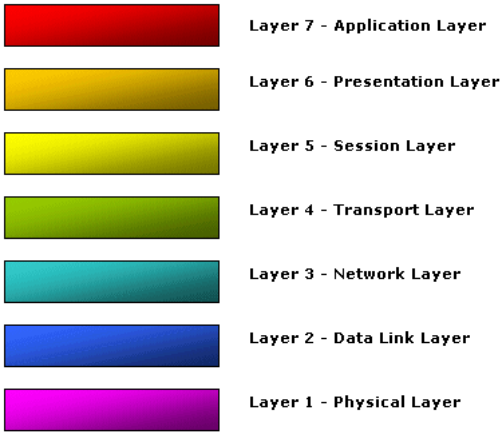
Kerrosarkkitehtuurit

- Kun kerrosten väliset rajapinnan on selkeästi määritelty, kerros voidaan vaihtaa toiseksi ilman, että se vaikuttaa muuhun järjestelmään
- rajapintojen/ kerrosten toimintojen *standardointi* lisää vaihdettavuutta

Kerrosarkkitehtuurit



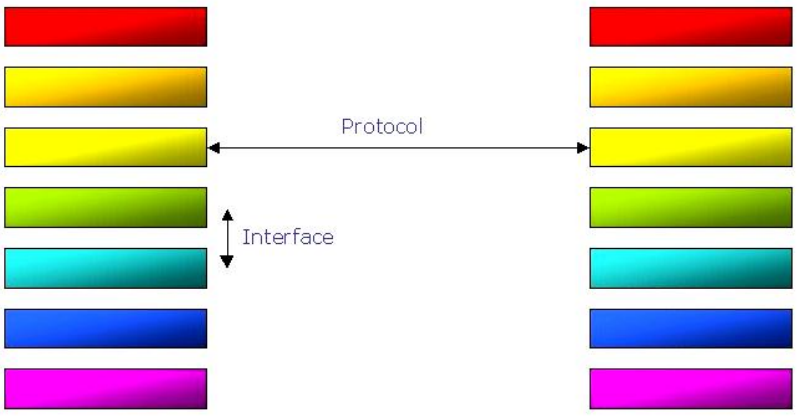
Kerrosarkkitehtuurit – OSI[1]



Eräänlainen *referenssiarkkitehtuuri* kommunikaatiojärjestelmille

[1]http://en.wikipedia.org/wiki/Osi_model

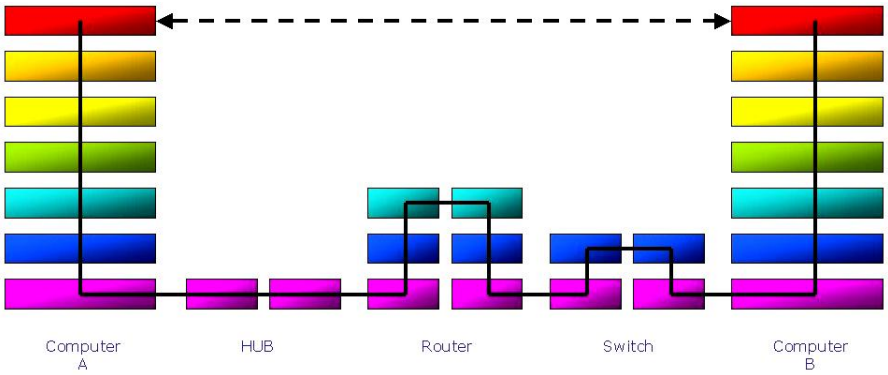
Kerrosarkkitehtuurit - OSI



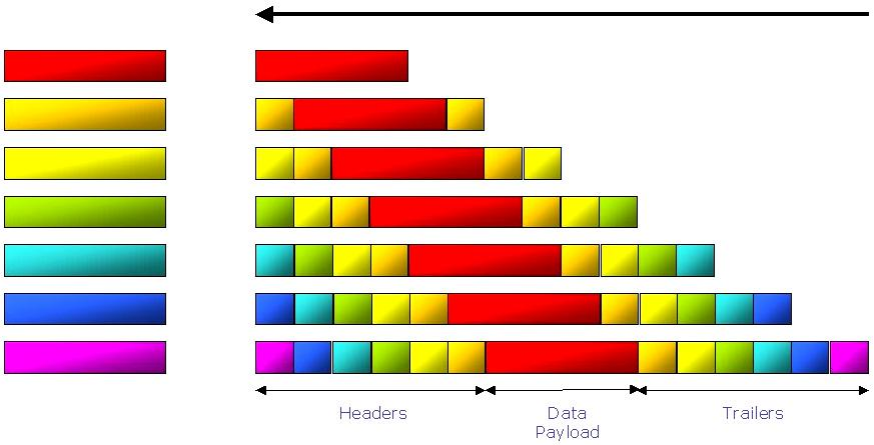
Vastinkerrokset kommunikoivat samalla abstraktiotasolla (yhteisellä protokollalla)

Kerrosarkkitehtuurit - OSI

Kokonaiskuva



Kerrosarkkitehtuurit - OSI



Jokainen kerros lisää oman palansa viestiin sen kulkiessa kerrosten läpi

Kerrosarkkitehtuurit – muita esimerkkejä

- Virtuaalikoneella rajapinta tai kieli (esim. Java bytecode), jonka avulla konetta käytetään
  - kätkee toteutuksen, mahdollistaa useita alustoja
- Sovellusrajapinnat palveluihin (API, application programming interface)
  - Esimerkiksi käyttöjärjestelmillä, luokkakirjastoilla → kätkee yksityiskohdat, parantaa siirrettävyyttä
- Asiakas-palvelin arkkitehtuuri
  - 2-tasoinen kerrosarkkitehtuuri
  - etäproseduurikutsut kommunikoinnissa

Kerrosarkkitehtuurit

- Etuja:
  - Kerrosarkkitehtuuri on yleinen malli, jota voidaan soveltaa lähes kaikissa järjestelmissä pienessä tai isossa mittakaavassa
  - Jakaa järjestelmän korkealla tasolla osiin, joiden muodostamana kokonaisuutena järjestelmä on helppo ymmärtää
  - Intuitiivinen, helposti ymmärrettävä → voidaan käyttää kommunikointiin hyvin erilaisten sidosryhmien välillä
  - Ohjaa ohjelmiston riippuvuuksien minimointiin → muutosten paikallisuus → helppo ylläpidettävyys, muutettavuus, vaihdettavuus, *standardoitavuus*
  - Kerrosarkkitehtuuri tukee ohjelmiston uudelleenkäyttöä
    - Kukin kerros toteuttaa tietyn abstraktion → pohja työn jakamiselle → tasokohtaista erikoistumista

Kerrosarkkitehtuurit

- Ongelmia:
  - Tehokkuushäviö
    - Palvelukutsu ei välity suoraan palvelun toteuttavalle kerrokselle, vaan kulkee välissä olevien kerrosten kautta
    - Palvelukutsun parametrit mahdollisesti muutetaan joka kerroksen välillä uuteen esitysmuotoon
  - Toteutustaakka: periaatteessa jokainen palvelu on toteutettava joka tasolla (vaikkakin osa toteutuksesta delegoidaan alemmalle kerrokselle)
  - Oikean (toimivan) tasojaon keksiminen on hankalaa
    - Mihin kerrokseen tietty palvelu pitäisi liittää?

Kerrosarkkitehtuurit

- *Ongelmia:*
  - *Poikkeusten käsittely*
  - Kutsu ylemmältä kerrokselta →
    poikkeus kerrosrakenteen pohjalla →
    palataan kutsupinossa taaksepäin kunnes löytyy käsittelijä poikkeukselle →
    poikkeuksen käsittely ylemmällä tasolla kuin missä poikkeus tapahtui →
    käsittelijä ei välttämättä pysty korjaamaan tilannetta
  - Ääritapaus: poikkeus palaa käyttäjälle saakka antaen mystisen virheilmoituksen
  - Miksei sitten poikkeuksia käsitellä aina siellä missä ne syntyvät?