Arkkitehtuurityylejä ja patterneja

Luento 4

Oppimistavoitteet

- Esimerkkejä yleisesti käytetyistä arkkitehtuuripatterneista ja -tyyleistä
- Muutama käydään läpi perusteellisemmin (N-Tier, Klassinen MVC, web-MVC, tietovarasto- ja tietovuotyylit)

Tyylien ja patternien käytöstä

- Platoninen vs. ”ruumiillistunut” tyyli/patterni
  - Platoninen tyyli/patterni on ideaalinen kuvaus arkkitehtuuriratkaisusta
  - Käytännön toteutuksissa esiintyvät tyylien/patternien instanssit (”ruumillistumat”) ovat harvoin täysin ”puhtaita”, eli niissä on tehty jotain perusteltuja poikkeuksia ja lisäyksiä ratkaisun rakenteisiin ja rajoitteisiin
  - Tyylit ja patternit ovat *yleistyksiä*
    - Yleistys, joka määrittelee arkkitehtuurien *luokan*
  - On myös paljon sovellusalue- ja teknologiakohtaisia patterneja, joilla on rajallisempi sovellusala
    - Esim. yksityiskohtaisia ”reseptejä” tietyn teknologian käyttöön

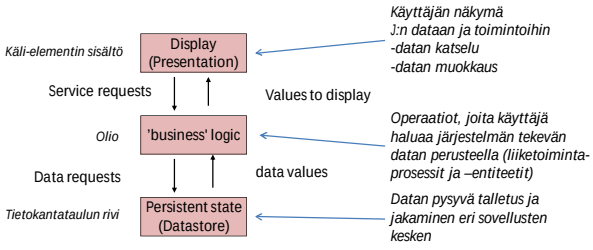
Ajatuskoe

- Konkreettinen sovellusesimerkki voi auttaa ymmärtämään tyylejä ja niiden käyttöä
- Ajatellaan web-kauppaa, jossa myydään jotain fyysisiä tuotteita (esim. vaatteita ym.)
  - Mitä kaikkia toimintoja web-kaupan tietojenkäsittelyjärjestelmän pitää toteuttaa? Mitä kaikkea sen pitää tehdä?
  - Mitä laatuvaatimuksia toimintoihin liittyy?
  - Mitä teknisiä asioita/ratkaisuja toteutukseen liittyy?

Esimerkkejä

ARKKITEHTUURIPATTERNEJA

Kolmitasoarkkitehtuuri (N-Tier)



N-Tier?

- Käytännössä isoissa järjestelmissä puhutaan kolmen tason sijaan usein *monitasoisesta rakenteesta* (N-Tier)
- Business logic –taso jaetaan yleensä edelleen (1) palvelupyyntöjen vastaanoton ja käyttäjä-istuntojen hallinnan tasoon sekä (2) palvelut toteuttavien operaatioiden ja ”business-olioiden” tasoon
- Business-oliot taas käyttävät yritysjärjestelmätason palveluita (tietokannat, toiset järjestelmät)
- *N* on siis yleensä 4 tai jopa 5

Kolmitasoarkkitehtuuri (N-Tier)

- Sovellusalue
  - Hajautetut, data-intensiiviset informaatiojärjestelmät
- Edut
  - Helpottaa tietoturvan, suorituskyvyn ja saatavuuden optimointia eri tasoilla niille sopivilla tavoilla
  - Lisää järjestelmän modulaarisuutta ja muunneltavuutta, koska eri tasojen välille täytyy sopia määrämuotoiset kommunikaatioprotokollat (-> loose coupling)
- Haitat
  - Kustannukset, monimutkaisuus

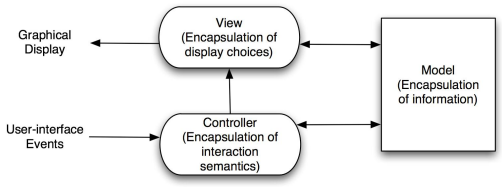
Klassinen Model-View-Controller (MVC) [malli-näkymä-ohjain]

- 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.

Klassinen 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

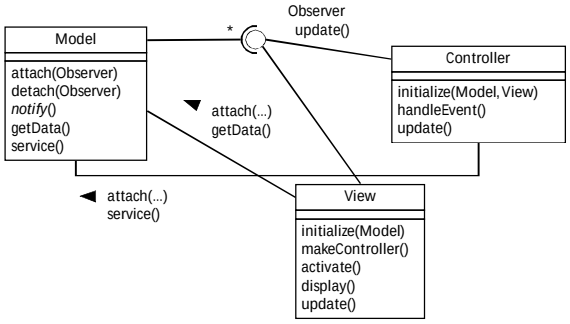
Klassinen MVC



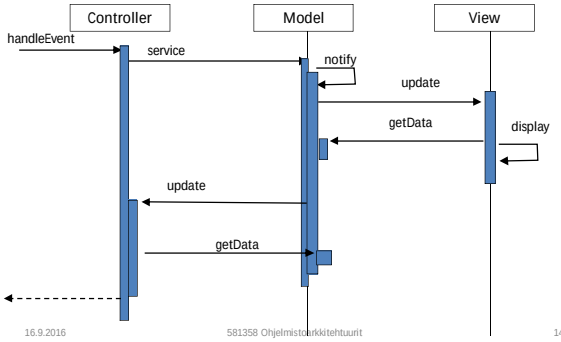
Klassinen MVC

- Mallikomponentit eivät riipu mistään (tietystä) näkymä- tai ohjainkomponentista
- Muutokset mallikomponenteissa välitetään näkymille *Tarkkailija*-suunnittelumallin mukaisesti
  - Eli vain niille näkymille, jotka ovat ilmaisseet kiinnostuksensa muutoksiin

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, Web -applikaatiot)
- Ongelmia
  - Lisää mutkikkuutta
  - Yksinkertainen käyttäjätoimenpide saattaa aiheuttaa monia päivityksiä näkymään (päivityksen laajuutta voi olla vaikea kontrolloida)
  - Voi olla työlästä löytää sopiva datan esitysmuoto ja haku-/päivitysoperaatiot mallin rajapintaan käyttöliittymän (näkymän) suorituskyvyn (responsiivisuus) kannalta

Muunnelmia

- Monesti ei ole tarpeen erottaa *ohjainta* ja *näkymää*
  - Erityisesti, jos kullakin näkymällä olisi joka tapauksessa oma ohjain, jonka kautta voidaan suoraan manipuloida näkymää
    - Voimakas kytkentä (strong coupling)
  - Käytännössä monissa käyttöliittymäkehyksissä (GUI frameworks) ”kontrollerikoodi” liitetään UI elementteihin suoraan tapahtumankäsittelijöinä
- Ts. komponentti voi toimia sekä näkymä- että ohjainkomponenttina. Yhteen malliin voi liittyä monta <näkymä, ohjain> -paria.
- Huom – juuri ohjaimen rooli ja tehtävät vaihtelevat eniten MVC-patternin eri ”ruumiillistumissa”!

Web-MVC

- MVC –patternin idea erottaa sovelluksen data ja tila näiden esittämisestä käyttäjälle on niin hyödyllinen, että sitä sovelletaan laajasti web-sovellusten toteutuksessa
  - Myös eri konteksteissa, eli sekä web-*palvelimen* (server, backend) että *asiakkaan* (selain, client, front-end) puolella
- MVC:n käyttö palvelinpuolella: http://advancedkittenry.github.io/koodaaminen/arkkitehtuuri/index.html
- Client-side MVC: AngularJS https://angularjs.org/
  - ”Model-View-Whatever”

MVC + N-Tier?

- Ovatko nämä vaihtoehtoisia, toisensa poissulkevia patterneja, vai voisiko ne yhdistää?

Esimerkkejä

ARKKITEHTUURITYYLIT

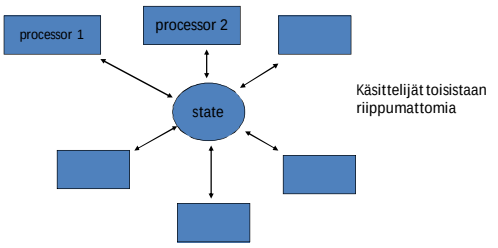
Tyylien luokittelua

- Eri lähteet luokittelevat tyylejä eri tavoin
  - Fairbanks ei esimerkiksi luokittele tyylejä lainkaan
  - Bass, Clements ja Kazman (*Software Architecture in Practice,* 3. painos) taas kutsuvat tyylejä patterneiksi ja luokittelevat ne omalla tavallaan
- Seuraavassa noudatetaan kurssin edellisen oppikirjan (Taylor & al.) luokitusta (ei kattava - luokat eivät pistevieraita)
- Tarkastellaan muutamaa esimerkkiä eri luokista
  - Jaettu tietovarasto
  - Tietovuopohjaiset tyylit

Jaettuun tietovarastoon perustuva arkkitehtuuri

- *Data centered - Shared state - Repository*
- Joukko komponentteja ylläpitää yhteistä tilaa tietovarastossa
  - Erilaisia muunnelmia sen mukaan, kuinka aktiivinen rooli tietovarastolla on
- Kahdenlaisia komponentteja
  - Keskitetty tietorakenne: tietovarasto
  - Asiakaskomponentit: hakevat tietoa tietovarastosta ja muokkaavat sitä
- Järjestelmän kontrolli määräytyy tietovaraston tilan mukaan

Jaetun tietovaraston arkkitehtuuri



Jaetun tietovaraston arkkitehtuuri

- Vuorovaikutus käsittelijöiden välillä tapahtuu ainoastaan tietovaraston kautta
- Käsittelijöiden tietovarastoon tekemät muutokset johtavat vaiheittain haluttuun lopputulokseen
- Käsittelijän aktivointi
  - Käsittelijät voivat pollata tietovarastoa tutkiakseen onko tila sellainen, mistä käsittelijä pystyy jatkamaan toimintaa
    - jos on, käsittelijä tuottaa tuloksia, jotka kirjaa tietovarastoon
  - Tietovarasto voi aktivoida käsittelijän jonkin säännön perusteella, esimerkiksi sisällön muuttumisen perusteella
    - vrt tietokantatriggerit

Jaetun tietovaraston arkkitehtuuri

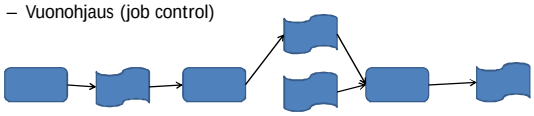
- Käsittelijät voivat toimia rinnakkain
  - Edellyttää samanaikaisuuden hallintaa: tiedon lukitus à miten estetään yhtä käsittelijää varaamasta koko tietovarastoa itselleen (liian pitkäksi aikaa)?
- Esimerkkejä
  - Tekoälysovellukset (*blackboard*-arkkitehtuuri), Wikit, Google docs, MS Sharepoint, muut verkkotyöalustat…

Jaetun tietovaraston arkkitehtuuri

- Etuja
  - Muunneltavuus & laajennettavuus (itsenäiset, toisistaan riippumattomat käsittelijät)
  - Rinnakkaisuuden hyödyntäminen
- Haittoja
  - Rinnakkaisuuden ja päällekkäisten päivitysten hallinta on mietittävä tarkkaan

Tietovuopohjaiset tyylit

- *Data flow styles*
- Pääpaino siinä miten tieto liikkuu toisistaan riippumattomien tietoa prosessoivien 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, esim. erilaiset raportointisovellukset
    - XML-muunnokset käyttäen dokumenttipuuta

Tietovuopohjaiset tyylit

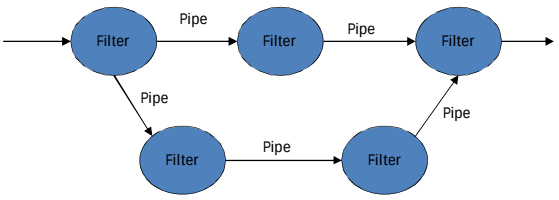
- Erätöiden sarja - etuja:
  - Muunneltavuus, vaiheet ovat riippumattomia toisistaan (vuonohjaus hoitaa koordinoinnin)
  - Yksinkertainen rakenne, suoritus etenee yksi vaihe kerrallaan
  - Potentiaalisesti hyvä suorituskyky (throughput)
- Haittoja:
  - Tulos on valmis ja saatavilla vasta, kun viimeinen vaihe on valmis (ei inkrementaalisuutta)
  - Rinnakkaisuutta ei voi hyödyntää

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 à 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 – etuja
  - Muunneltavuus, suotimet itsenäisiä prosessoreita, järjestelmän helppo konfigurointi eri käyttötarkoituksiin (-> yksi Unix-komentorivi)
  - Rinnakkaisuuden hyödyntäminen
  - Tulosten inkrementaalinen saatavuus
- Haittoja
  - Ei sovi interaktiivisiin sovelluksiin
  - Kovin eri tahtiin toimivat suotimet voivat aiheuttaa vaikeuksia väylien toteutukselle (puskurointi)
  - Syötteen loppumisen havaitseminen voi vaatia erikoistoimia

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