Ohjelmistoarkkitehtuurit, syksy 2012
Harjoitukset 2 (19.9.-20.9.2012)

1. Luennoilla oli esillä alkuperäinen moni-ikkunaisiin työasemasovelluksiin tarkoitettu arkkitehtoninen MVC (model view controller) ratkaisumalli. Mallia on käytetty paljon ja siitä on kehitetty lukuisia muunnoksia. Tutustu Microsoftin MVVM (Model-View-ViewModel) malliin ja vertaa sitä alkuperäiseen MVC:hen. Miten MVVM eroaa MVC:Stä?
    - MSDN Magazine (Febr. 2009) - "WPF Apps With The Model-View-ViewModel Design Pattern" http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

    Lähteitä:
    [1] http://en.wikipedia.org/wiki/Model_View_ViewModel
    [2] http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

    Pääpointit:
    - Graafisten käyttöliittymien toteutukseen hyödyntäen *data binding* tekniikkaa
        - Käyttöliittymän View komponentti määritellään deklaratiivisesti XML –kuvauksena (ei ohjelmakoodina esim C#)
        - ViewModel on datan muunnoskomponentti (mediator), joka kytkeytyy varsinaiseen tietomalliin (Model) ja tarjoaa View:lle helppokäyttöisen liittymän tietosisältöön ja sitä muokkaaviin operaatioihin -> ViewModel tarjoaa tieto-oliot (View:lle) määrämuotoisten rajapintojen kautta (view data binding), jotka kehitystyökalut osaavat suoraan kytkeä XML:llä annetun UI-määrittelyn elementteihin ja attribuutteihin (View:n määrittelyssä ei tarvita siis lainkaan ohjelmointia).
        - View:n ei tarvitse tuntea Modelin toteutustekniikkaa
        - Riippuvuudet View -> ViewModel -> Model
    - MVC:n Controller-rooli sulautuu View:iin (tapahtumien ja UI:n tilan käsittely) ja ViewModelin operaatioihin (data & command binding)
    - Roolitus View:n ja ViewModelin väillä jossain määrin joustava

2. Ohjelmatason ratkaisumallit (design patterns) ovat hyväksi havaittuja tapoja ratkaista jokin ohjelmarakenteeseen liittyvä ongelma. Selvitä seuraavien ratkaisumallien idea ja käyttötilanteet:
    - Sovitin (adapter)
    - Strategia (strategy)
    - Julkisivu (facade)

    Adapter: http://en.wikipedia.org/wiki/Adapter_pattern

    Strategy: http://en.wikipedia.org/wiki/Strategy_pattern

    Façade: http://en.wikipedia.org/wiki/Facade_pattern

3. Arkkitehtuurimalleissa ja ohjelmatason ratkaisumalleissa on sovellettu yleisiä ohjelmistojen suunnitteluperiaatteita. Selvitä seuraavien periaatteiden ideat ja mieti missä edellä käsitellyissä ratkaisu- tai arkkitehtuurimalleissa niitä on käytetty (perustele):
    - Separation of Concerns
    - Information Hiding

    SoC: http://en.wikipedia.org/wiki/Separation_of_concerns

    - Different or unrelated responsibilities should be separated from each other
    - Collaborating components that contribute to the solution of a specific task should be separated from components involved in other tasks
    - If a component plays different roles in different contexts, the roles should be separate independent and separate from each other in the component
    - Käytännössä kaikki mallit noudattavat tätä; Façade siltä osin, jos se tarjoaa tietyn asiakasryhmän näkökulmasta liittymän järjestelmään – eri käyttäjille voidaan tarjota erikoistunut façade

    Information Hiding: http://en.wikipedia.org/wiki/Information_hiding

    - Eräs vanhimmista ja ehkä tärkein ohjelmistosuunnittelun periaatteista
    - *Muuttuvat osat* ohjemiston toimintojen toteutuksesta sijoitetaan omiin moduuleihinsa, jotka tarjoavat *vakaan rajapinnan* (muuttumattoman) toimintoihin. Kullakin moduulilla on oma ”salaisuutensa” (yleensä implementaatiotekniikkaan liittyvä), jota muut eivät tiedä -> ei voi syntyä riippuvuutta eri moduulien implementaatioiden välille (muuta kuin rajapintojen kautta). Implementaatiot voivat muuttua toisistaan riippumatta, jos moduulirajapinnat pysyvät samoina.
    - Facade ilman muuta, Strategia, Adapter, MVC jne

4. Tutustu Struts web-sovelluskehykseen. Selvitä sen arkkitehtuurin pääperiaatteet sekä avainteknologiat, joista Struts riippuu. Mitä arkkitehtuurityylejä tai ratkaisumalleja Struts:ssa käytetään? Eroavatko Struts:n käyttämät mallit luennoilla esitetyistä? Miten?
    - http://struts.apache.org/

    Struts ja muut web-frameworkit soveltavat yleisesti MVC arkkitehtuurimallia. Näissä View ei yleensä ole suoraan GUI elementti, vaan komponentti, joka luo web-palvelimen selaimelle palauttaman XML/HTML dokumentin (~dynaamisesti luotu websivu), jonka selain sitten ”renderöi” käyttäjälle. Controller –komponentti toimii palvelupyntöjen (http GET/PUT) vastaanottajana, valitsee sopivan käsittelijän prosessoimaan pyynnön. Käsittelijä prosessoi pyynnön käyttäen siihen Modelia, jonka jälkeen Controller valitsee View:n. View luo sitten vastauksen, ja palauttaa sen pyynnön lähettäjälle.

    Teknologiat: key tehchnologies primer http://struts.apache.org/primer.html mm. Java EE web services, Servlet API, JSP API, HTTP, ...

    Arkkitehtuuri: http://www.dzone.com/tutorials/java/struts/struts-tutorial/struts-mvc-architecture-tutorial.html

MVC:tä ei siis käytetä suoraan GUI –käyttöliittymäkontekstissa vaan web-palvelun toteutuksessa erottamaan (web-)resurssin representaation luonti palvelupyyntöjen hallinnasta ja palveluoperaatioiden suorittamisesta.

Kaikki asiakaspyynnöt menevät aina Controllerin (ActionServlet) kautta modelille, ei suoraan View:stä Modeliin.

AJAX –tyyppisessä palvelun toteutuksessa suuri osa tiedon prosessoinnista voi tapahtua asiakkaan päässä (web-selaimessa), jos asiakkaalle palautetaan interaktiiviseen toimintaan ja tiedon prosessointiin kykenevä View. Tällöin View :n (asiakkaan web browserin) ja Controllerin (web-applikaation) kulkee lähinnä muuttunut data. Nyt View ja Controller siis sijaitsevat http-yhteyden erottamina eri laitteilla.