Arkkitehtuurityylejä ja suunnittelutaktiikoita

Luento 5

Oppimistavoitteet

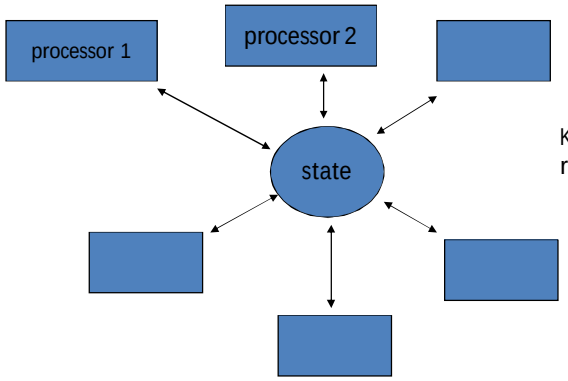
- Arkkitehtuurityylejä
  - jaettu tietovarasto, viestinvälitysarkkitehtuurit, vertaisverkkoarkkitehtuuri , map-reduce
- Suunnittelutaktiikoita laatuominaisuuksien saavuttamiseksi
  - Suorituskyky- ja muunneltavuustaktiikat

ARKKITEHTUURITYYLIT (OSA 2)

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



Käsittelijät toisistaan riippumattomia

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 → 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 päällekäisten päivitysten hallinta on mietittävä tarkkaan

Viestinvälitysarkkitehtuurit

- *Implicit invocation*
- Komponentit kommunikoivat lähettämällä toisilleen *viestejä*
- Viesti voidaan lähettää tietylle kohteelle tai yleislähetyksenä kaikille tunnetuile kohteille tai kaikille kiinnostuneille kohteille
- Viestintä voi olla
  - sykronista (lähettäjä jää odotamaan) tai
  - asynkronista (lähettäjä jatkaa toimintaansa)

Viestinvälitysarkkitehtuurit

- Julkaisija – Tilaaja (Publish – Subscribe)
- Tuottaja – Kuluttaja (Supplier- Consumer)
- Yksinkertaisimmillaan julkaisija pitää kirjaa vastaanottajista ja tietosisällön muuttuessa välittää tiedon tilaajille proseduurikutsun välityksellä.
- Tilaaja rekisteröityy vastaanottajaksi ja toimittaa rekisteröinnin yhteydessä vastaanottorajapinnan (call-back)
- Yllä oleva toimii ohjelman sisäisesti
- Verkkoratkaisussa tarvitaan *kommunikointiprotokolla* ja *välittäjä* (*broker*) huolehtimaan viestinvälityksestä

Viestinvälitysarkkitehtuurit Julkaisija-Tilaaja

- *Välittäjä* tietää vastaanottajat (rekisteröityminen tai konfigurointi)
- Julkaisija toimittaa viestin välittäjälle
- Välittäjä toimittaa viestin edelleen siitä *kiinnostuneille* tilaajille
- Tilaaja käsittelee viestin

Viestinvälitysarkkitehtuurit Julkaisija-Tilaaja

- Arkkitehtuurissa määriteltävä seuraavat asiat:
  - Keskenään kommunikoivat komponentit
  - Viestit, joiden avulla kommunikointi tapahtuu ilman, että lähettäjä tuntee vastaanottajan (tai vastaanottaja lähettäjän) sijaintia
  - Operaatiot, joilla komponentit reagoivat viesteihin (komponenttien ymmärtämä kieli)
  - Säännöt, joiden avulla komponentit ja viestit rekisteröidään järjestelmälle

Viestinvälitysarkkitehtuurit
Julkaisija-Tilaaja

- Säännöt, joiden perusteella välittäjä tietää, mille komponentille viesti on lähetettävä
  - Viestin vastaanottajan selvittäminen:
    - yleisviesti (multiple dispatch; viesti kaikille, jotkut käsittelevät, toiset eivät)
      - etu: vastaanottajan ei tarvitse etukäteen kertoa, mitä viestejä se haluaa vastaanottaa
    - komponentit kertovat rekisteröinnin yhteydessä, mitä viestejä ne haluavat vastaanottaa (tai keneltä)

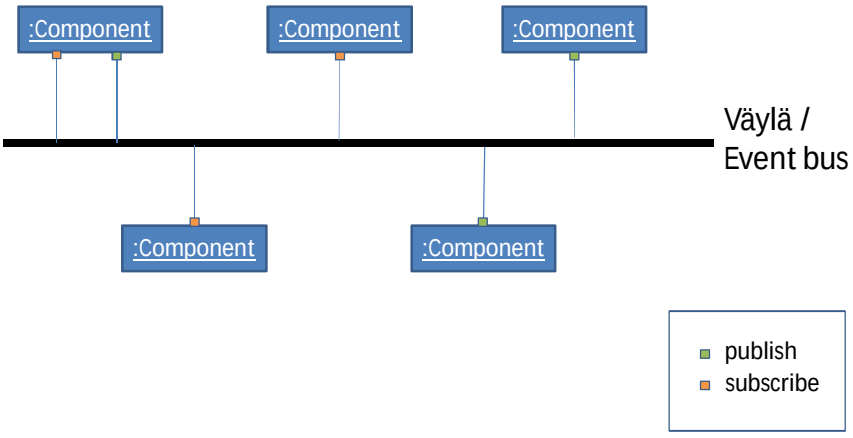
Viestinvälitysarkkitehtuurit
Julkaisija-Tilaaja

- Rinnakkaisuus: missä määrin välittäjä ja komponentit toimivat rinnakkain
  - viestinvälittäjillä ja vastaanottajilla viestijonot (välittäjän laatikko, vastaanottajan laatikko)
  - Puskurointi, palvelutasot

Viestinvälitysarkkitehtuurit

- *Tapahtumapohjaiset* arkkitehtuurit (*event-based*, *event driven*)
- Komponentit kommunikoivat aiheuttamalla tapahtumia (event)
- Tapahtumat voivat välittyä kaikille muille komponenteille, joista jotkut käsittelevät ja toiset eivät noteeraa
  - Tapahtumien lähetys voidaan myös rajoittaa julkaisija – tilaaja rakenteen tapaan – tapahtuma lähetetään vain tapahtuman tyypistä kiinnostuneille
  - komponentteja ei ole kuitenkaan lokeroitu joko julkaisijoiksi tai tilaajiksi vaan komponentti voi toimia kumpanakin

Viestinvälitysarkkitehtuurit Tapahtumapohjainen

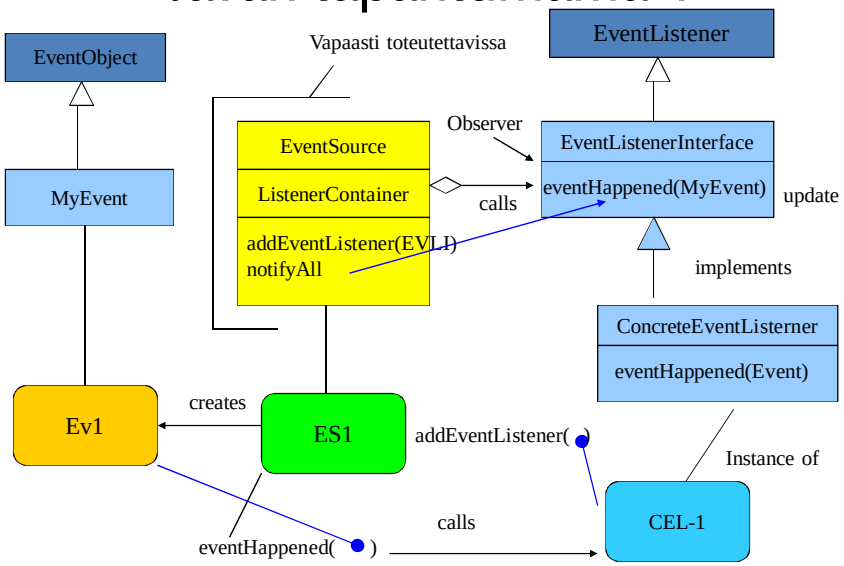


Viestinvälitysarkkitehtuurit
Tapahtumapohjainen

- Tapahtuma
  - Tilanne, joka voi sattua ohjelman suorituksen aikana
  - Edellyttää reagointia joiltain järjestelmän osilta
  - *Lähde* = tapahtuman synnyttävä komponentti
  - *Tarkkailija* = tapahtumaan reagoiva komponentti
- Lähde lähettää tapahtumailmoituksen sitä tarkkailemaan rekisteröityneelle tarkkailijalle
  - Esimerkiksi *Signals & Slots*[1] Qt-sovelluskehyksen olioarkkitehtuurissa
- Lähde tietää dynaamisesti tarkkailijoidensa olemassaolon, mutta ei niiden tarkkaa tyyppiä
  - Tapahtumaväylää käytettäessä lähteet eivät tiedä tarkkailijoistaan (vrt. MVC, Observer -mallit)
- Ilmoituksen lähetys voidaan hoitaa proseduurikutsuna tai sanomanvälityksenä (event bus, message bus)
  - proseduurikutsu synkroninen (esim Qt Signals & Slots)
  - Sanomajono asynkroninen/synkroninen

[1]http://en.wikipedia.org/wiki/Signals_and_slots,
http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html

Javan tapahtumamalli



Viestinvälitysarkkitehtuurit

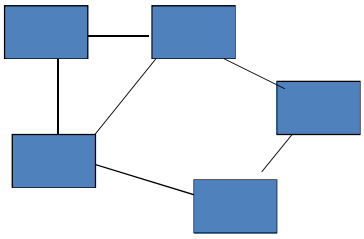
- Etuja
  - Joustavuus, muunneltavuus, modulaarisuus
    - Uusien tuottajien/julkaisijoiden ja kuluttajien/tilaajien dynaaminen lisääminen
    - Välittäjän/väylän tekemät automaattiset esitystapamuunnokset, erilaisia palvelutasoja toteutettavissa (varmistettu perillemeno vs. fire and forget)
    - Komponenttien omatahtinen evoluutio
  - Paljon käytettyjen välittäjä-/väyläratkaisujen hyväksi kehittynyt laatu
    - Luotettavuus, skaalautuvuus, suorituskykyoptimoinnit, jne.

Viestinvälitysarkkitehtuurit

- Haittoja
  - Välittäjän tai tapahtumaväylän tuoma suoritusrasite (overhead) verrattuna suoriin (point-to-point) yhteyksiin

Vertaisverkkoarkkitehtuuri

- *peer-to-peer*
- Verkko, jonka solmut voivat toimia sekä *asiakkaina* että *palvelimina*



Puhtaassa vertaisverkossa kaikki solmut ovat tasavertaisia
Verkko on *symmetrinen* ja *ei-hierarkinen*

Vertaisverkkoarkkitehtuuri

- Palvelupyyntö etenee verkossa solmusta solmuun kunnes löytyy solmu, joka kykenee täyttämään pyynnön
- Rakenteettomassa verkossa pyynnön eteneminen sokeaa
  - Solmut tuntevat suorat naapurinsa, mutta eivät näiden tarjoamia palveluita
- Rakenteellisessa solmut jakavat rakennetietoa palveluista ja tätä käytetään ohjaamaan etenemistä
- Edellyttää verkkoprotokollaa

Vertaisverkkoarkkitehtuuri

- Etuja
  - Saatavuus paranee, jos palvelu/resurssi on hajautettu useina (osa-)kopioina verkon solmuihin (esimerkiksi jonkin mediatiedoston fragmentit)
  - Vikasietoisuus kasvaa, koska yksittäisen solmun vikaantuminen ei ole kriittistä (tiedon/palvelun toisteisuus, vaihtoehtoiset saantipolut)
  - Skaalautuvuus ja laajennettavuus ovat hyvät, ei yksittäistä kriittistä komponenttia (no single point of failure)

Vertaisverkkoarkkitehtuuri

- Haittoja
  - Verkkoon voi muodostua saaria tai klikkejä, jonka solmut eivät ole yhteydessä klikin ulkopuolisiin solmuihin
  - Yksittäiset haut saattavat aiheuttaa paljon turhaa liikennettä (resurssia etsitään turhaan, tai se löytyy monista solmuista)
  - Verkon rakenne ei ole vakaa (solmuja tulee ja lähtee)

Vertaisverkkoarkkitehtuuri

- Vertaissolmujen rinnalle onkin usein pakko luoda pysyviä *Super-* tai *Ultra-solmuja* jotka
  - Liittävät uusia solmuja verkkoon
  - Kytkeytyvät suoraan moniin alempiin solmuihin ja toisiinsa hakujen optimoimiseksi
  - Katso esimerkiksi Gnutella, Skype

Map-Reduce -arkkitehtuuri

- Motivaatio
  - Hyvin suuren datamäärän hajautettu tallennus ja prosessointi
  - Data ja käsittelyoperaatiot suhteellisen yksinkertaisia, mutta niitä on todella paljon
- Perusperiaate
  - Koko datamassa jaetaan pienempiin osiin (*splits*)
  - Osien käsittely jaetaan rinnakkain suoritettaviin tehtäviin, jotka tuottavat paikallisen välituloksen (*map*-operaatio)
  - Välitulokset yhdistetään globaaliksi tulokseksi (*reduce*-operaatio)

Map-Reduce -arkkitehtuuri

- Etuja
  - Skaalautuvuus, saatavuus, suorituskyky
  - Katso esim. Rackspace –esimerkki 1. luennolta
- Huomattavaa
  - Suoritusnopeus riippuu siitä, miten map-operaatio onnistuu jakamaan syötedatan ”yhtä vaativiin” palasiin

SUUNNITTELUTAKTIIKAT

Laatutekijät arkkitehtuurisuunnittelussa

- Onnistunut arkkitehtuuri on edellytys järjestelmän keskeisten laatuvaatimusten saavuttamiseksi
- *Pelkät* arkkitehtuuritason ratkaisut eivät kuitenkaan takaa laatuvaatimusten täyttymistä
  - Myös yksityiskohdat ja toteutustapa vaikuttavat, mutta eivät voi parantaa huonoa arkkitehtuuriratkaisua
- Onnistuneen järjestelmän edellytyksenä on sekä kokonaisuuden (~ arkkitehtuuri) että yksityiskohtien (~ toteutus) laadun varmistaminen

Laatutekijät arkkitehtuurisuunnittelussa

- Laatuominaisuudet (a.k.a laatuattribuutit, laatutekijät) vaikuttavat toisiinsa
  - Esimerkiksi millä tahansa muulla laatuattribuutilla on yleensä negatiivinen vaikutus suorituskykyyn (vrt. esim. muokattavuus, turvallisuus, saatavuus, …)
  - Laatuominaisuuksien tasapainottelua (trade-off) ei voi yleensä välttää (esim. Rackspace –tapaus 1. luennolta)
- Ei voida keskittyä laatuattribuutteihin yksittäin, vaan kokonaisuus ratkaisee

Laatutekijät arkkitehtuurisuunnittelussa

- Laatuattribuutteja
  - accessibility, accountability, accuracy, adaptability, administrability, affordability, auditability , autonomy, availability, credibility, process capabilities, compatibility, composability , configurability, correctness, customizability, degradability, determinability, demonstrability, dependability, deployability , distributability, durability, efficiency, evolvability, extensibility, failure transparency, fault-tolerance, fidelity, flexibility, installability, Integrity, interchangeability, interoperability , learnability, maintainability , mobility, modularity,nomadicity, operability. orthogonality, portability, precision, predictability. provability. recoverability, relevance, reliability, repeatability, reproducibility, resilience, responsiveness, reusability, robustness, safety, scalability, seamlessness, self-sustainability, serviceability, securability, simplicity, stability, standards compliance, survivability, sustainability, tailorability, testability, timeliness, traceability, ubiquity, understandability, upgradability, usability

  (Wikipedia)
- Ohjelmistojen laatumalli: ISO 25000 standardisarja (a.k.a *SQuaRE*, luettavissa Kumpulan tiedekirjastossa)

Laatutekijät arkkitehtuurisuunnittelussa

- Bass et al.[1] ovat määritelleet *taktiikoita* laatuominaisuuksien saavuttamiseen
  - Taktiikka on suunnittelupäätös, joka suoraan vaikuttaa laatuominaisuuden toteutumiseen
  - Tyylejä ja malleja yksityiskohtaisempia ja lähempänä toteutusta
- Tarkastellaan esimerkkeinä suorituskykyä ja ylläpidettävyyttä

[1]http://etutorials.org/Programming/Software+architecture+in+practice,+second+edition/Part+Two+Creating+an+Architecture/Chapter+5.+Achieving+Qualities/

Laatutekijät arkkitehtuurisuunnittelussa

- *Suorituskykytaktiikoiden* tavoitteena on saavuttaa määrätty reaktio-/vastausaika johonkin järjestelmän vastaanottamaan tapahtumaan
- Vastausaika riippuu
  - 1. resurssien käytöstä
  - 2. odotusajasta
    - Kilpailu resursseista
    - resurssin saatavuus
    - Riippuvuus muusta toiminnasta
- Resurssitarpeeseen vaikuttavat taktiikat
  - laskennan tehostaminen (algoritmi, välitulosten käsittely)
  - yleisrasitteen vähentäminen (esim. etäkutsut paikallisiksi)
  - tapahtumamäärän vähentäminen
  - tapahtumien vastaanoton kontrollointi
  - käsittelyajan rajoittaminen
  - jonojen koon rajoittaminen

Laatutekijät arkkitehtuurisuunnittelussa

- Resurssien hallintataktiikat
  - Rinnakkaisuuden lisääminen
  - Toiminnan / datan monistaminen
  - Resurssien lisääminen
- Resurssien allokointitaktiikat
  - jono
  - prioriteettijono
  - dynaaminen prioriteettijono
  - staattinen allokointi

Laatutekijät arkkitehtuurisuunnittelussa

- Muunneltavuustaktiikat (modifiability tactics)
  - Ohjelmiston muunneltavuutta mitataan laskemalla aikaa ja kustannuksia, jotka kuluvat ohjelmistomuutosten toteuttamiseen, testaamiseen ja käyttöönottoon
  - Muunneltavuutta lähellä olevia tekijöitä: adaptability, evolvability, maintainability
- Muunneltavuustaktiikat voidaan jakaa kolmeen ryhmään
  - Muutosten lokalisointiin perustuvat taktiikat
  - Heijastusvaikutusten (ripple effect) estämiseen perustuvat taktiikat
  - Sidonnan viivästämiseen perustuvat taktiikat

Muunneltavuustaktiikat Muutosten lokalisointi

- Tavoitteena rajoittaa odotettavissa olevien muutosten vaikutus mahdollisimman pieneen joukkoon komponentteja
  - Määrätään komponenttien vastuut siten, että odotettavissa olevat muutokset koskevat rajattua joukkoa komponentteja
- Lokalisointitaktiikkoja:
  - Komponentin vastuiden semanttinen yhtenäisyys (semantic coherence)
    - Vastuut hoidettavissa ilman laajaa ulkopuolista kommunikointia
  - Yleiskäyttöisten palvelujen eristäminen omiin moduuleihin
  - Odotettavissa oleviin muutoksiin varautuminen
  - Yleistäminen ja parametrisointi
    - Erikoistaminen parametrien kautta – parametrien tulkinta
  - Vaihtoehtojen rajoittaminen

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Ohjelmamuutoksen heijastusvaikutukset (*ripple effects*) tarkoittavat muutoksia, jotka täytyy tehdä moduuleihin, joita varsinainen muutos ei suoraan koske
- Jos moduulia A muutetaan, niin moduulia B joudutaan muuttamaan vain siksi, että A:ta on muutettu – ei siksi että B:n ominaisuuksia on ollut tarve muuttaa
  - Tämä heijastusvaikutus johtuu siitä, että B:n toteutuksessa on jokin riippuvuus A:n toteutukseen

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Moduulien välisiä riippuvuustyyppejä:
  - Syntaktinen (muotoa koskeva) riippuvuus koskien dataa tai palveluja
    - Yhteinen formaatti datalle, palveluiden kutsumuodot
  - Semanttinen (merkitystä koskeva) riippuvuus koskien dataa tai palveluja
    - Datan/palvelun käyttäjän on tulkittava merkitys samoin kuin tuottajan
  - Järjestysriippuvuus koskien dataa tai kontrollia
    - Datan saapuminen tietyssä järjestyksessä
  - Palveluiden suorittaminen tietyssä järjestyksessä , esim. A.n pitää olla suorittanut palvelu A1 ennen kuin B voi suorittaa palvelun B1
  - Rajapinnan identiteetti tunnettava
    - Nimi tai kahva (”handle”) oltava tiedossa käännös ja suoritusaikana

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Moduulien välisiä riippuvuustyyppejä…
  - Suoritusaikainen sijainti tunnettava
    - Esim. IP-osoitteen tietäminen
  - Datan tai palvelun laatutaso tunnettava
    - esim. tarkkuus (onko sentit mukana?)
  - Komponentin olemassaolon edellyttäminen
    - Ei voida luoda dynaamisesti
  - Moduulin käyttämien resurssien tunteminen

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Semanttisiin riippuvuuksiin perustuvia heijastusvaikutuksia on yleisesti ottaen vaikea estää
- Voi olla epätarkoituksenmukaista tarjota kaikille käyttäjille semantiikaltaan täsmälleen samanlaista rajapintaa ja käyttäytymistä kuin ennen muutoksia
  - Vanhan implementaation muuttamiseen on yleensä painavia syitä
  - Kahta erillistä implementaatiota voidaan joutua ylläpitämään siirtymäkauden ajan (vanhaa uuden rinnalla)
- Seuraavassa esitetyistä taktiikoista mikään ei välttämättä estä semanttisiin riippuvuuksiin perustuvia heijastusvaikutuksia

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Tiedon piilotus (Information hiding)
  - Odotettavissa oleviin muutoksiin varautumista
    - Todennäköisesti muuttuvien ominaisuuksien piilottaminen on moduulijaon yhtenä pääperiaatteena
- Olemassa olevien rajapintojen säilytys (Maintain existing interfaces)
  - Jos B riippuu A:n tarjoaman rajapinnan nimestä tai kutsumuodoista, rajapinnan ja syntaksin säilyttäminen tarkoittaa, ettei B:tä tarvitse muuttaa A:n päivityksen yhteydessä
    - Ei välttämättä auta, jos riippuvuudet ovat semanttisia tai palveluiden/datan laatutasoon tai resurssien käyttöön liittyviä
    - Rajapintojen lisääminen, sovittimen (adapter) tai edustajaolion käyttö (proxy)

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Kommunikaatioväylien rajoittaminen
  - Rajoitetaan niiden moduulien määrää, joiden kanssa tietty moduuli käyttää samaa dataa
  - Rajoitetaan niiden moduulien määrää, jotka tuottavat tietyn moduulin käyttämää dataa tai jotka käyttävät sen tuottamaa dataa
- Välittäjän käyttö
  - Jos B:n riippuvuus A:sta ei ole semanttista tyyppiä, on aina mahdollista lisätä välittäjä B:n ja A:n väliin
  - Välittäjä piilottaa riippuvuuden konkreettisen ilmentymän ja tarjoaa B:lle muuttumattoman näkymän A:han
  - Välittäjä vastuulla on vain riippuvuuden hallinta, ei mitään muuta
  - Erilaisia välittäjiä eri tarpeisiin: nimipalvelin, resurssimanageri

Muunneltavuustaktiikat Sidonnan viivästäminen

- Edellä kuvatut taktiikat pyrkivät vähentämään muutosten kustannuksia pienentämällä muutettavien moduulien määrää
- Niistä ei kuitenkaan ole hyötyä, jos halutaan lisätä ohjelmiston muunneltavuutta koskien moduulien käyttöönottotapaa ja –aikaa tai halutaan mahdollistaa muiden kuin kehittäjien tekemän muutokset

Muunneltavuustaktiikat Sidonnan viivästäminen

- *Sidonnan viivästäminen* tukee molempia tavoitteita
  - Periaate: lisäominaisuus tai muunneltu toiminnallisuus voidaan ottaa käyttöön ja kytkeä muuhun ohjelmistoon ajoaikana (vs. koostamisaika, build-time)
- Nopeuttaa käyttöönottoa: järjestelmää ei tarvitse ajaa alas (ja käynnistää uudelleen)
- Vaatii lisäinfrastruktuuria -> kasvattaa kustannuksia

Muunneltavuustaktiikat Sidonnan viivästäminen

- Ajonaikainen rekisteröinti (runtime registration) mahdollistaa plug-and-play operaatiot. Rekisteröinti aiheuttaa lisäkustannuksia. Tuottaja/kuluttaja rekisteröinti voidaan tehdä ajoaikaisesti tai latauksen yhteydessä.
- Konfigurointitiedostoissa voidaan säädellä käynnistyksen yhteydessä tehtäviä sidontoja ja parametreja.
- Polymorfismi mahdollistaa metodikutsujen myöhäisen sidonnan.
- Komponentin korvaus mahdollistaa latausaikaisen sidonnan.
- Protokollien noudattaminen mahdollistaa prosessien ajonaikaisen sidonnan