

Sovellusaluemalli *Domain Model*

Luento 8

24.9.2015

581385 Ohjelmistoarkkitehtuurit

1

Oppimistavoitteet

- Sovellusalueen mallintaminen
 - Tietomalli, invariantit, toiminnalliset skenaariot
- Suunnittelun mallintamisen perusteet
 - Komponentit, konnektorit, rajapinnat ja portit

24.9.2015

581385 Ohjelmistoarkkitehtuurit

2

Esimerkkiohjelmisto – *Yinzer*¹

- *Yinzer* on internetissä toimiva sosiaalisen median palvelu, joka tarjoaa jäsenilleen työhön liittyvää verkostoitumista ja työpaikkailmoittelua Pittsburghin alueella. Jäsenet voivat lisätä toisia palvelun jäseniä kontakteihinsa, julkaista työpaikkailmoituksia, ehdottaa jotain kontaktiaan avoimeen työpaikkaan ja vastaanottaa sähköposti-ilmoituksia itselleen sopivista työpaikoista.

¹Yinzer = slanginimityshenkilölle, joka on kotoisin Pittsburghista (PA, U.S.A.)

24.9.2015

581385 Ohjelmistoarkkitehtuurit

3

SOVELLUSALUEMALLI

24.9.2015

581385 Ohjelmistoarkkitehtuurit

4

Sovellusaluemallin käyttö

- Sovellusaluemallinnuksen tehtävä on varmistaa, että toteutettavan ohjelmiston kannalta oleelliset sovellusalueen piirteet on ymmärretty oikein
- Tuloksena suppeampi malli kuin koko liiketoiminnan mallinnuksessa (*business modelling*), jonka tarkoitus on kattavasti kuvata jonkin yrityksen liiketoiminnan tavoitteet ja päätöksentekoprosessit (*enterprise architecture*)
 - Liiketoimintamallin perusteella voidaan päättää, mitä liiketoiminnan prosesseja automatisoidaan, ja sovellusaluemallin avulla kuvataan automatisoitavan osuuden piirteet

24.9.2015

581385 Ohjelmistoarkkitehtuurit

5

Sovellusaluemallin käyttö

- Sovellusaluemalli auttaa pitämään ohjelmistototeutukseen liittyvät suunnittelupäätökset erillään sovellusalueen käsitteistä (vastaavuussuhde)
- Suhteellisen yksinkertaisia sovellusaluealleja voidaan käyttää kommunikoinnissa sovellusalueen asiantuntijoiden (ja asiakkaiden) kanssa tuloksellisemmin kuin suunnittelumalleja
 - Mallin pohjalta rakentuu *yhteinen kieli* ohjelmistokehittäjien ja sovellusasiantuntijoiden välille

24.9.2015

581385 Ohjelmistoarkkitehtuurit

6

Sovellusaluemallin käyttö

- Sovellusaluemalleista on eniten hyötyä järjestelmissä, joiden sovellusalue on suhteellisen monimutkainen ja jota kehittäjät eivät vielä hallitse
- Esimerkiksi laiteajureiden kehittäjillä on tyypillisesti paljon teknisiä haasteita ja hankalia laatuominaisuuksia toteutustyössä, mutta sovellusalue on suhteellisen suppea
 - Sovellusaluemallista on heille vähemmän hyötyä
 - He ovat usein itse alueen parhaita asiantuntijoita
- Ennen pitkää projektissa kuin projektissa tulee kuitenkin vastaan ongelma, jossa sovellusaluemalli voi auttaa
- *Sovellusalueen puutteellinen ymmärtäminen* on monien IT-projektien epäonnistumisen perimmäisiä syitä

24.9.2015

581385 Ohjelmistoarkkitehtuurit

7

Epäilyksiä

- Sovellusalue on jo tunnettu – miksi mallintaa sitä?
 - Näin osittain onkin, ainakin "kuumilla" uutuusaloilla, mutta "tylsät business-jutut" voivat olla huonosti ymmärrettyjä, missä piilee riski
- Sovellusalue on niin simppele, ettei sitä kannata mallintaa
 - Ymmärtävätkö kehittäjät ja käyttäjät käyttäjien tarpeet varmasti samalla tavalla?

24.9.2015

581385 Ohjelmistoarkkitehtuurit

8

Epäilyksiä

- Sovellusalue on yhdentekevä arkkitehtuurin suunnitteluratkaisujen kannalta
 - Järjestelmien integroinnissa tulee tilanteita, joissa integroitavien (osa-)järjestelmien erilaiset tulkinnot samoista sovellusalueen käsitteistä johtavat yhteensopivuusongelmiin. Myös arkkitehtuurit ovat tällöin yleensä yhteensopimattomia.
- On jonkun toisen homma määritellä vaatimukset
 - Ehkä, mutta näillä henkilöillä ei välttämättä ole ymmärrystä, mitkä asiat voivat aiheuttaa ongelmia arkkitehtuurin kannalta, ja heitä joutuu joka tapauksessa muutenkin avustamaan

24.9.2015

581385 Ohjelmistoarkkitehtuurit

9

Epäilyksiä

- Sovellusalueen oppi parhaiten vähitellen, koodatessa
 - Varmasti näin on, kaikissa tilanteissa tämä ei vain ole mahdollista, esimerkiksi evaluoitaessa valmiita ohjelmistoja integroitavaksi osaksi isompaa järjestelmää
 - Suurissa projekteissa yksittäisten kehittäjien käsitykset sovellusalueesta voivat haitallisesti erkaantua
- Sovellusaluemallinnus on itseään ruokkiva prosessi, joka ei lopu koskaan (analysis paralysis)
 - Tämä on todellinen riski!

24.9.2015

581385 Ohjelmistoarkkitehtuurit

10

Analysis paralysis

- Paras lääke tähän tautiin on miettiä ensin, mihin sovellusalueeseen liittyviin kysymyksiin todennäköisesti tarvitaan vastauksia suunnittelu- ja toteutustyön aikana
 - Kun vastaukset on saatu, mallinnus voidaan lopettaa
- Kysymykset riippuvat projektin riskeistä, tyypillisiä ovat *käytettävyyteen* ja *yhteensopivuuteen* (interoperability) liittyvät riskit
- On nähtävä, milloin mallinnuksen jatkamisesta ei enää saada lisäarvoa verrattuna muihin aktiviteetteihin - esimerkiksi prototyypin tekemiseen

24.9.2015

581385 Ohjelmistoarkkitehtuurit

11

Sovellusaluemallin osat

- Tietomalli (Information model)
- Invariantit ja rajoitteet (Invariants, constraints)
- Tilannekuva (snapshot)
- Toiminnallinen skenaario (functionality scenario)

24.9.2015

581385 Ohjelmistoarkkitehtuurit

12

Tietomalli

- Helpoimmin tehtävä ja kaikkein hyödyllisin sovellusaluemallin osa, joka määrittelee sovellusalueen *asiat* ja niiden väliset *suhteet*
- Tietomalli voidaan määritellä tekstuaalisesti (ikäin kuin *sanastona*) tai graafisesti UML-kaavioina (käsite- eli luokkakaaviot)
 - UML:n käytössä noudatettava pidättyvyttä, että myös sovellusalueen asiantuntijat ymmärtäisivät ne (SME = subject matter expert)
 - Tarkoitus on kuvata *sovellusalueita*, ei suunnitteluratkaisuja!

24.9.2015

581385 Ohjelmistoarkkitehtuurit

13

Tekstimuotoinen tietomalli

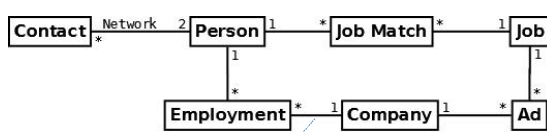
Tyyppi	Maaritelma
Ad (mainos)	Ilmoitus, jolla haetaan henkilöä (Person) yhtiössä (Company) olevaan avoimeen työpaikkaan (Job)
Company (yhtio)	Työnantaja, joka tarjoaa töitä (Job) henkilöille (Person)
Contact (kontakti)	Kahden henkilön (Person) välinen suhde, joka tarkoittaa, että henkilöt tuntevat toisensa
Employment (työsuhde)	Suhde, joka ilmaisee, että henkilö on tai on ollut töissä (Job) yhtiössä (Company)
Job (työ, toimi)	Yhtiön (Company) toimi tai toimen kuvaus
Job Match (rekrytointiehtodutus)	Toimen (Job) ja henkilön (Person) välinen suhde, joka tarkoittaa, että henkilö voi olla sopiva toimeen toisen henkilön mielestä
Person (henkilö)	Joku, joka voidaan ottaa työsuhteeseen

24.9.2015

581385 Ohjelmistoarkkitehtuurit

14

Tietomalli UML-kaaviona



Tyyppien (luokan) väliset suhteet (assosiaatiot) ja suhteisiin liittyvät määrälliset rajoitukset ovat suoraan näkyvissä

24.9.2015

581385 Ohjelmistoarkkitehtuurit

15

Invariantit ja rajoitteet

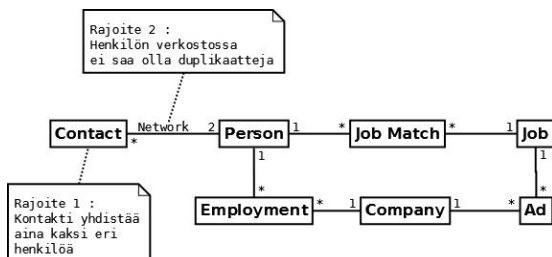
- Tietomalli määrittelee sovellusalueen mallinnuksessa käytettävän sanaston
- Invariantit eli rajoitteet ilmaisevat väitteitä tai ehtoja (predicate), joiden pitää aina olla tosia
- Osa rajoitteista voidaan ilmaista suoraan kaavioissa (assosiaatioiden roolien määrälliset rajoitteet)
- Muu rajoite voidaan liittää UML-kaavioon huomautuksena (note) tai kirjata se johonkin erilliseen dokumenttiin
 - Formaaliilla OCL-kielellä voidaan ilmaista täsmällisiä UML-mallia koskevia rajoitteita, assosiaatioketjuja pitkin navigoiden

24.9.2015

581385 Ohjelmistoarkkitehtuurit

16

Invariantit



24.9.2015

581385 Ohjelmistoarkkitehtuurit

17

Tilannekuva

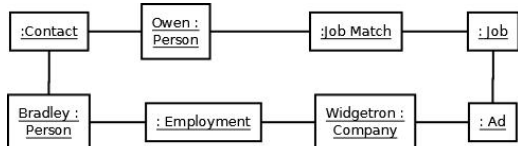
- Tietomalli kuvaa sovellusalueen asioita tyyppien tasolla, ei konkreettisten olioiden
- Tilannekuva taas kuvaa jossakin konkreettisessa tilanteessa muodostuvan olioiden eli *tyyppien instanssien* ja niiden välisten *yhteyksien* konfiguraation
- UML:n oliokaaviota käytetään olioiden ja niiden välisten *linkkien* (suhteiden eli assosiaatioiden instanssien) kuvaamiseen tilannekuvaossa
 - Jokainen kaavion olio kuuluu johonkin tyyppiin ja jokainen kaavion linkki on jonkin tyyppien välisen suhteen yksittäinen ilmentymä
- Tilannekuvia voi käyttää apuna rajoitteita mietittäessä, esimerkiksi voiko henkilö olla oma kontaktinsa

24.9.2015

581385 Ohjelmistoarkkitehtuurit

18

Tilannekuva



24.9.2015

581385 Ohjelmistoarkkitehtuurit

19

Toiminnalliset skenaariot

- Tilannekuvat näyttävät, minkälaisia olioita ja niiden välisiä linkkejä jollain tietyllä ajan hetkellä on olemassa
- Tietomalli ja invariantit kuvaavat taas kaikki mahdolliset tilanteet
 - Yksittäisten tilanteiden on oltava tietomallin asettamien rajoitteiden mukaisia (esim. assosiaatioiden kardinaliteetit)
- Nämä eivät kuitenkaan kuvaa, miten sovellusalueen tila muuttuu ajanhetkestä toiseen
- *Skenaariot* kuvaavat tapahtumasarjoja, jotka aiheuttavat muutoksia tilanteista toisiin

24.9.2015

581385 Ohjelmistoarkkitehtuurit

20

Skenaarioesimerkki

Nimi	Owen saa toita Widgetronista
Alkutila	Bradleyon toisissa Widgetron -yhtiössä
Toimijat	Owen, Bradley, Widgetron
Askelet	<ol style="list-style-type: none"> 1. Owen ja Bradley tapaavat ammatillisessa konferenssissa, vaihtavat käyntikortteja ja liittyvät toistensa verkostoon (Contact) 2. Bradleyn työnantaja (Company) Widgetron julkaisee hakuilmoituksen (Ad) sovelluskehittäjän toimeen (Job) 3. Bradley ehdottaa (Job Match) Owenia sopivaksi henkilöksi Widgetronille 4. Widgetron palkkaa (Employment) Owenin sovelluskehittäjäksi

24.9.2015

581385 Ohjelmistoarkkitehtuurit

21

Skenaarioista

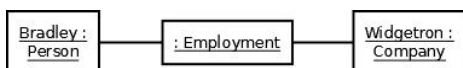
- On tärkeää laatia skenaario niin, että jokainen skenaarion askel aiheuttaa tilanteen muuttumisen
 - Auttaa keskittymään tietomallin kuvaamiin asioihin ja jättää turhan yksityiskohdat ja rönsyt pois
- On hyvä ajatella tilannekuvaa ennen ja jälkeen jokaisen askelen, mikä auttaa kytkemään käyttäytymisen tiukasti tietomalliin

24.9.2015

581385 Ohjelmistoarkkitehtuurit

22

Skenaarion tilannekuvat - alkutila

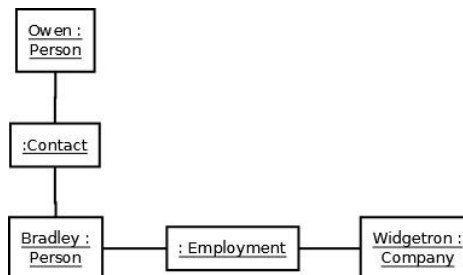


24.9.2015

581385 Ohjelmistoarkkitehtuurit

23

1. Askeleen jälkeen

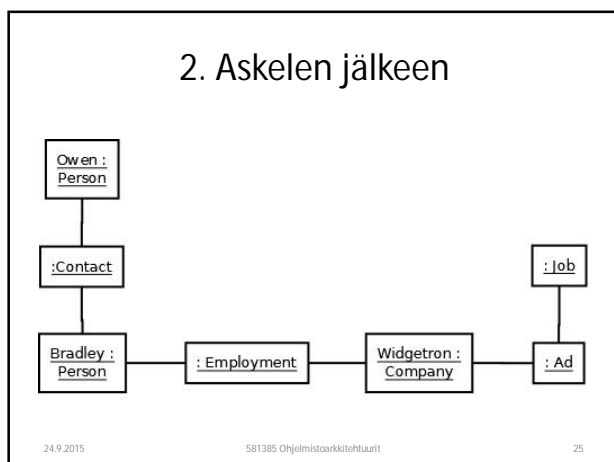


24.9.2015

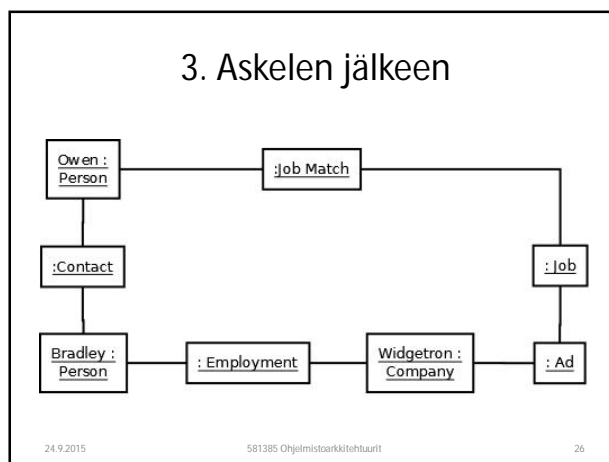
581385 Ohjelmistoarkkitehtuurit

24

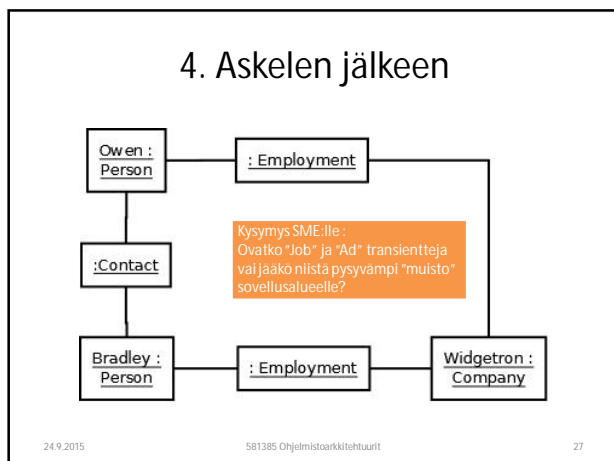
2. Askelen jälkeen



3. Askelen jälkeen



4. Askelen jälkeen



Skenaarioista

- Skenaario kertoo reaali maailman asioista, eikä ota kantaa, miten ne ohjelmistossa toteutetaan
- Skenaario kuvaa yhden mahdollisen tapahtumien polun ja siihen liittyvät tilanteet
 - Useimmiten muutama skenaario perustapauksista riittää
- UML:n tilakaaviot (State diagram) ja aktiviteettikaaviot (Activity diagram) sopivat yleisempiin kuvauksiin, jotka määrittelevät kaikkien mahdollisten polkujen joukon
 - Vaativat myös enemmän työtä

24.9.2015

581385 Ohjelmistoarkkitehtuurit

28

Yhteenveto

- Sovellusalue malli auttaa pitämään suunnittelupäätökset erillään sovellusalueen ymmärtämiseen liittyvistä kysymyksistä
- Sovellusalue malli määrittelee yhteisen kielen SME:iden ja kehittäjien välille
- Mallin laajuus ja syvyys on syytä päättää etukäteen miettimällä mihin kysymyksiin sen avulla pitää voida vastata ja minkälaisia erikoistilanteita ja "outouksia" on käsiteltävä
- Malli on väistämättä todellisuuden yksinkertaistus mutta sen pitää olla silti täsmällinen
- Nyökkisääntö: jos jonkin sovellusalueen piirteen ymmärtämättä jääminen aiheuttaa riskin, se pitää mallintaa, muuten ei

24.9.2015

581385 Ohjelmistoarkkitehtuurit

29

Huomautus

- *Entity-Relationship*-mallinnus on yleisesti käytetty tietojärjestelmien tietosisällön analysointi ja määrittely formalismi
 - http://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model
- Menetelmä on erityisesti tarkoitettu relaatiotietokantojen rakenteen suunnittelun apuvälineeksi ja tarjoaa hieman UML:stä poikkeavan semantiikan
 - Kokonaisuutena UML tarjoaa monipuolisemman joukon kaaviotyyppejä sovellusalueen mallinnukseen, mutta ER-malleissa on omat vahvat puolensa

24.9.2015

581385 Ohjelmistoarkkitehtuurit

30

SUUNNITTELUMALLIN PERUSELEMENTTEJÄ

24.9.2015

581385 Ohjelmistoarkkitehtuurit

31

Arkkitehtuuriajattelun elementtejä

- Ennen varsinaista suunnittelumalliin perehtymistä, käydään läpi peruskäsitteitä, joilla on merkittävä käytännöllinen ja myös käsitteellinen rooli arkkitehtuurisuunnittelussa
 - Komponentti
 - Konnektori
 - Portit ja rajapinnat

24.9.2015

581385 Ohjelmistoarkkitehtuurit

32

KOMPONENTTI

24.9.2015

581385 Ohjelmistoarkkitehtuurit

33

Komponentti

- Komponentti on arkkitehtuuri-elementti, joka
 - kapseloi osan järjestelmän toiminnallisuutta ja/tai dataa
 - rajoittaa pääsyn tähän osaan tarkoin määritellyn rajapinnan kautta tapahtuvaksi ja
 - jolla on eksplisiittisesti määritellyt riippuvuudet suoritusympäristönsä.
- Voi olla iso (osajärjestelmä tai koko järjestelmä) tai pieni (yksittäinen operaatio)
 - Voi *koostua* muista komponenteista, joiden avulla toteuttaa tarjoamansa palvelun (component assembly)
- Näkyy ulospäin vain rajapintansa kautta = tarjoaa rajapinnan muille elementeille: määrittelee, miten elementin palveluja käytetään, mutta ei niiden toteutusta
 - Rajapinnan toteuttava olio (tai muutama) "edustaa" komponenttia ulospäin

24.9.2015

581385 Ohjelmistoarkkitehtuurit

34

Komponentti

- Eksplisiittisesti määritellyt riippuvuudet
 - vaatimukset komponenteille, joiden kanssa toimii yhteistyössä – ns. vaadittu rajapinta (required interface)
 - tarvittavat resurssit esimerkiksi tiedostot tai hakemistot
 - ohjelma-alusta
 - laitteistoalusta
 - Määrittely metatadana, pakettiriippuvuuksina, dokumentteina
- Komponentit voivat olla
 - Sovellusriippuvia – kehitetty tietyn sovelluksen tarpeisiin
 - Sovellusalueerippuvia - uudelleenkäytettävissä sovellusalueen sisällä (domain specific)
 - Yleiskäyttöisiä – käytettävissä useilla sovellusalueilla

24.9.2015

581385 Ohjelmistoarkkitehtuurit

35

Komponentti

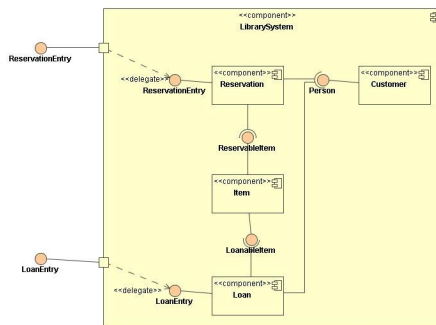
- CBD –komponentti (CBD = Component Based Development)
 - Aikoinaan tavoiteltiin komponenttimarkkinoita, joilta olisi voitu hankkia tarvittavia ohjelmistojen "binaarisia rakennuspalikoita" samaan tapaan kuin elektronikkakomponentteja - ajatus ei toteutunut
 - Nykyään voi kuitenkin hankkia erilaisia web-APIen kautta käytettäviä *palveluita* (melkein sama asia): tunnistaminen, maksupalvelut jne.
- Niinpä nykyään useimmat komponentit jaellaan paketteina lähde- tai käännettynä koodina, josta suoritusaikainen komponentin ilmentymä instantioidaan (jonkin standardin määrittämällä tavalla)
 - Esim. ladataan pakkauksen sisältämä Java-luokka ja luodaan siitä olio, joka puolestaan luo olioita muista pakkauksen sisältämistä luokista muodostaen näiden kanssa suoritusaikaisen komponentti-instanssin
 - Vaadin komponenttikirjasto <https://vaadin.com/directory#!browse>

24.9.2015

581385 Ohjelmistoarkkitehtuurit

36

Komponentit UML 2.0:ssa



24.9.2015

581385 Ohjelmistoarkkitehtuurit

37

KONNEKTORI

24.9.2015

581385 Ohjelmistoarkkitehtuurit

38

Konnektori

- Konnektori (connector)
 - Arkkitehtuuri-elementti, jonka tehtävänä on *mahdollistaa ja hallita komponenttien vuorovaikutusta*
 - Esimerkkejä konnektortyypeistä: proseduurikutsu, etäproseduurikutsu, jaetun datan käyttö, sanomanvälitys, jne
- Vastuita
 - Kontrollin siirto
 - Tiedon siirto
 - Tuki- ja alustapalvelut (facilities) esim. toiminnon käynnistys, viestinvälitys, transaktiot, persistenssi, jne

24.9.2015

581385 Ohjelmistoarkkitehtuurit

39

Konnektorien tehtäviä

- Tyypillisiä palveluita
 - Kommunikointi – tiedonsiirto (communication)
 - esim. sanomanvälitys
 - Koordinointi – kontrollin siirto (coordination)
 - esim. yksinkertainen proseduurikutsu - monimutkainen kuormantasausliittymä
 - Konversio – komponenttien vuorovaikutustapojen yhteensovitus (conversion)
 - esim. tietomuotokonversiot, kääreet (wrappers)
 - Toiminnan mahdollistaminen – tukitoiminta (facilitation)
 - esim. ajastus, samanaikaisuuden hallinta

24.9.2015

581385 Ohjelmistoarkkitehtuurit

40

Konnektortyyppejä

- Konnektortyytit
 - Proseduurikutsu (procedure call)
 - Tapahtuma (event)
 - Tietokytös (data access)
 - Linkitys (linkage)
 - Tietovuo (stream)
 - Välittäjä (arbitrator)
 - Sovitin (adaptor)
 - Välityspalvelu (distributor)

24.9.2015

581385 Ohjelmistoarkkitehtuurit

41

Konnektorin toteutus

- Komponentit *sovelluskohtaisia* palveluita, konnektorit *sovellusriippumattomia* vuorovaikutustapoja
- Ohjelmistojen toteutuksessa konnektoreilla
 - Ei usein ole omaa koodia eikä identiteettiä
 - Ei käännoisyksikkövastaavuutta
- Toteutus on hajautettu useaan moduuliin ja useaan vuorovaikutusmekanismiin

24.9.2015

581385 Ohjelmistoarkkitehtuurit

42

Konnektorit ja arkkitehtuurityylit

- Käsitteellisellä tasolla
 - Keskeisiä identiteetin omaavia arkkitehtuuri-elementtejä
 - Kuvaavat kaiken vuorovaikutuksen komponenttien välillä
 - Tarvitsevat omat määrytykset
- Konnektoreilla on usein tärkeä rooli *arkkitehtuurityyleissä* ja ratkaisumalleissa
 - Vaatteen tekevät henkilön ja konektoori tyylin
 - Mahdollistavat sovellusriippuvan toiminnallisuuden eriyttämisen komponentteihin

24.9.2015

581385 Ohjelmistoarkkitehtuurit

43

RAJAPINNAT JA PORTIT

24.9.2015

581385 Ohjelmistoarkkitehtuurit

44

Komponenttien rajapinnoista

- Ohjelmistorakennetta voidaan tarkastella komponentteina
- Komponentti muodostaa toiminnallisen kokonaisuuden, joka tarjoaa joukon loogisesti toisiinsa liittyviä palveluja
 - Palvelut käyttävät samoja tietorakenteita
 - Palveluja käytetään tyypillisesti samassa yhteydessä
 - Toiminnallisuus peruste komponentin olemassaololle
- Komponentti on työnjaon perusyksikkö kaikissa kehitystyön vaiheissa

24.9.2015

581385 Ohjelmistoarkkitehtuurit

45

Komponenttien rajapinnoista

- Yleinen ohjelmistosuunnittelun periaate: erotetaan se *mitä halutaan tehdä* siitä *miten toiminta toteutetaan*
 - Komponentin ei pitäisi riippua tietyistä toisista komponenteista, vaan joukosta (abstrakteja) *palveluja*, joita muut komponentit tarjoavat
 - "Abstrakti palvelu" määritellään rajapintana (interface)
- Minimaalinen rajapintamäärittely: palveluiden kutsumuodot (signatures)
 - kutsu voi olla muutakin kuin paikallinen tai etäproseduurikutsu
 - Esim. viesti-/tapahtumapohjainen protokolla

24.9.2015

581385 Ohjelmistoarkkitehtuurit

46

Komponenttien rajapinnoista

- Rajapinnat määräävät tavat, joilla komponentit kommunikoivat keskenään
 - Myös suurin osa arkkitehtuurityyleistä ja suunnittelumalleista perustuu rajapintojen käyttämiseen
- Huolellinen rajapintasuunnittelu edellytys
 - Työn järkevälle jakamiselle
 - Ohjelmiston keskeisten laatuominaisuuksien, kuten testattavuuden, ylläpidettävyyden ja muunneltavuuden varmistamiselle
- Rajapintasuunnittelu alkaa tyypillisesti heti keskeisten arkkitehtuuriratkaisujen tekemisen (esim. arkkitehtuurityylien valitsemisen) jälkeen

24.9.2015

581385 Ohjelmistoarkkitehtuurit

47

Rajapinnan suunnittelu

- "How to Design a Good API and Why it Matters"
 - Joshua Bloch, Google
 - <http://lcsd05.cs.tamu.edu/slides/keynote.pdf>

24.9.2015

581385 Ohjelmistoarkkitehtuurit

48

Komponenttien rajapinnoista

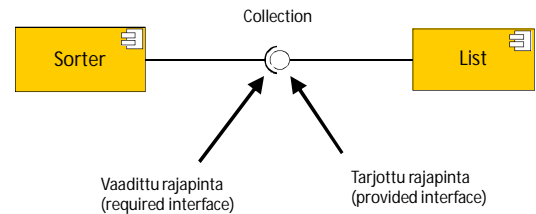
- Komponentti voi joko tarjota (eli toteuttaa) tai vaatia rajapinnan -> *tarjotut* ja *vaaditut* rajapinnat (provided and required interfaces)
- Tietty nimetty rajapinta on yleensä toisen komponentin vaatima ja toisen tarjoama
- Ohjelmointikielet, kuten Java tai C++, eivät tarjoa välineitä vaadittujen rajapintojen eksplisiittiseen kuvaamiseen komponenttien (pakkausten) tasolla (pitää kuvata ei-formaalilla dokumentaatiolla)
 - Eräät komponenttiarkkitehtuurit ja sovelluskehukset taas tukevat riippuvuuksien konfigurointia ja automaattista instansiointia komponentteihin liitetyn metadatan perusteella
 - Javan annotaatioita voi käyttää riippuvuuksien injektointia varten (kieli ei määrittele kuitenkaan injektointimekanismia)

24.9.2015

581385 Ohjelmistoarkkitehtuurit

49

Vaadittu ja tarjottu rajapinta



24.9.2015

581385 Ohjelmistoarkkitehtuurit

50

Portti

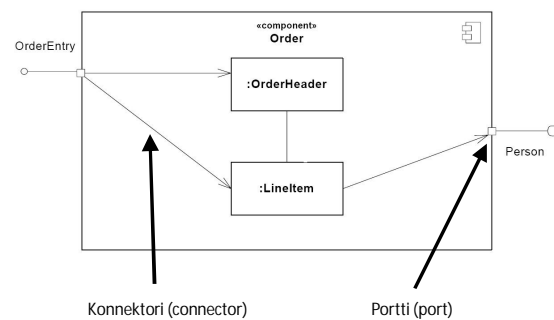
- Portti
 - Liitoskohta, jonka kautta komponentti on yhteydessä ulospäin
 - Mekanismi, jolla voidaan koota yhteen rajapintoja liittämällä ne samaan porttiin, esim. roolirajapinnaksi
- Portteja ei välttämättä tarvitse käyttää mallinnuksessa – jokaisella rajapinnalla implisiittisesti oma porttinsa
- Porttiin voi liittyä oma protokollansa ja tilakoneensa
 - Riippuu porttiin kytketyistä konnektoreista ja rajapintojen semantiikasta

24.9.2015

581385 Ohjelmistoarkkitehtuurit

51

Portti



24.9.2015

581385 Ohjelmistoarkkitehtuurit

52