

Intressipohjaisen ohjelmistojen teknologiat malliperustaisen sovelluskehityksen apuvälineinä

Smartphone Business Division, SysOpen Digia

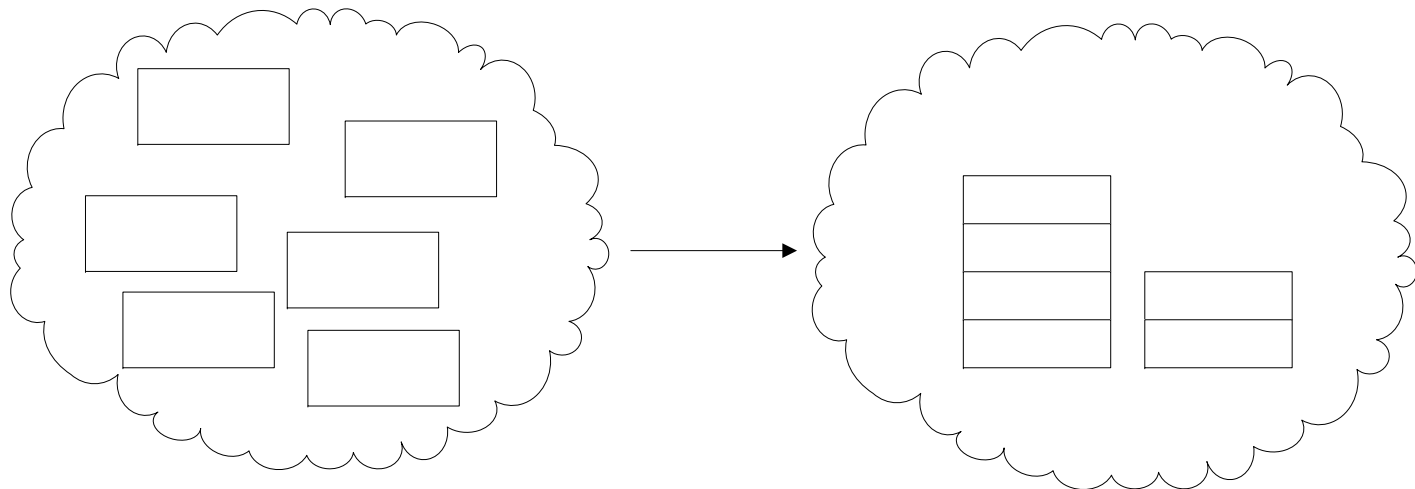
Tommi Reinikainen
Software Engineer
SysOpen Digia Plc

Sisältö

- Motivointia
- Intressi-perustainen lähtökohta
- Intressien kyselykieli
- Työkaluja intressilähtöiseen kehitykseen
- Yhteenvedo

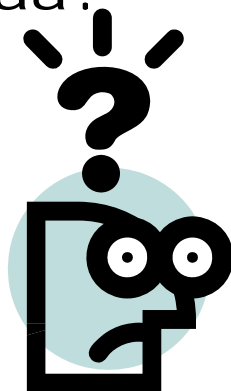
Motivointia

- Suurin osa ohjelmistoista on jaoteltu *yksiulotteisesti*. Esimerkkinä tällaisesta toimikoon modulaarinen ohjelmistojaottelu.
- Modulaarisessa jaottelussa järjestelmä jaetaan ensin pienempiin yksiköihin. Tässä yhteydessä yksikkö voi tarkoittaa vaikkapa funktiota, luokkaa tai pakkausta (Java).
- Seuraava askel on yksiköiden ryhmittely siten, että ne muodostavat suuremman kokonaisuuden, moduulin. Tällainen moduuli on luonteeltaan yksiulotteinen, sillä jokainen yksikkö kuuluu vain ja ainoastaan yhteen moduuliin.



Motivointi jatkuu

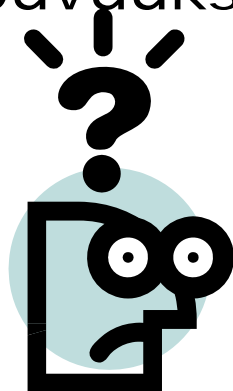
- Entä jos tarkastellulla yksiköllä ei olekaan yhtä selkeää sijoituspaikkaa?



- Tällöin yksikön sijoituspaikka käytännössä valitaan mielivaltaisesti. Mihin tahansa yksikkö sijoitetaankin, tietoa häviää.
- "*Dominoivan jaottelumekanismen tyrannia*" – Peri Tarr ("*Tyranny of the dominant decomposition*")

Motivointi jatkuu

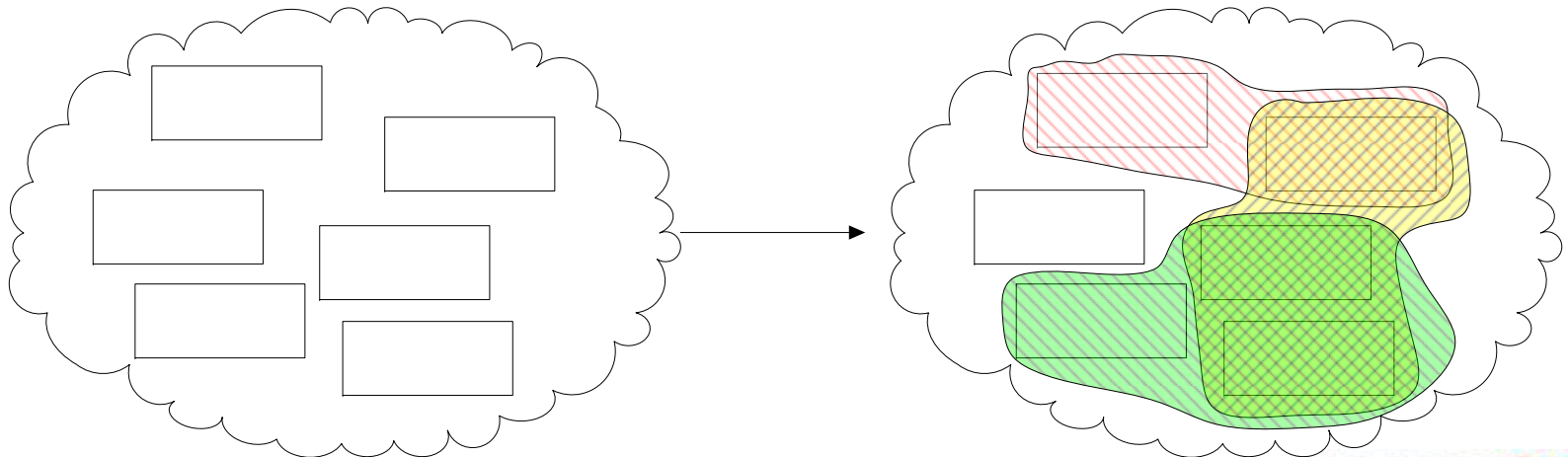
- Entäpä jos ohjelmistossa on implisiittisiä (ts. näkymättömiä) riippuvuuksia?



- Perinteiset mekanismit eivät tarjoa mitään valmista tapaa dokumentoida tai korostaa näitä riippuvuuksia.
- Näkymättömät riippuvuudet voivat muodostaa ongelmapesäkkeitä, joihin kajoaminen aiheuttaa mittavia ja vaikeasti ennustettavia virheitä ohjelman toiminnassa. Ongelmapesäkkeiden tunnistaminen on hankalaa.

Intressi-perusteinen lähestymistapa

- Jaotellaan ohjelmisto ohjelman eri sidosryhmien intressien pohjalta (esim. softainsinöörit, projektijohto, asiakas)
- Kun eri intressit on selvitetty, ohjelmiston yksiköt merkataan kuuluviksi eri intresseihin. Puhutaan intressien *mappaamisesta* ohjelmistoon. (engl. *concern representation*)
- Koska intressit perustuvat eri sidosryhmien näkymiin ohjelmistosta, on todennäköistä, että yksittäinen ohjelmiston yksikkö kuuluu useaan intressiin; ts. intressit ovat käytännössä päällekkäisiä. Tämä tarkoittaa sitä, että intressijaottelu on *moniulotteinen*.
- Intressijaottelussa tekemisen kohteena olevaan järjestelmään ei kajota "fyysisesti" vaan jaottelu on *non-intrusiivinen*.

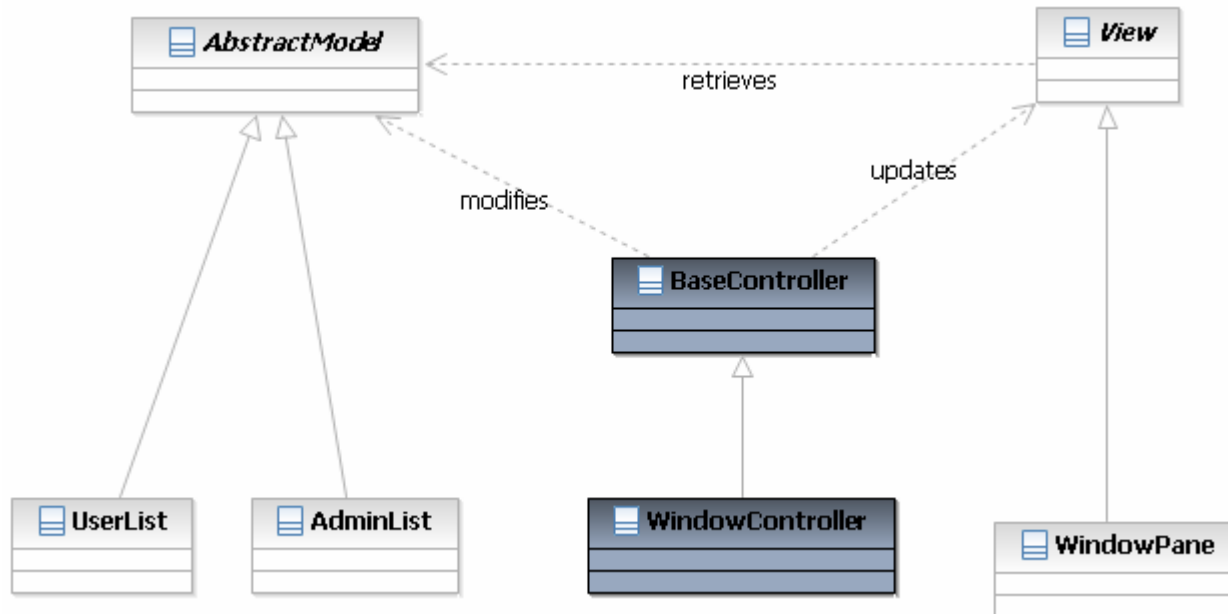


Intressi-perusteinen lähestymistapa

- Hyötyjä:
 - Mahdollistaa erilaiset näkökulmat järjestelmään.
 - Tietoa ei enää häviä, koska jaottelu ei ole enää yksiulotteista.
 - Järjestelmän ymmärrettävyys paranee.
 - Saadaan uusia tapoja analysoida järjestelmää.
 - Jaottelu voidaan ulottaa pelkistä kooditason yksiköistä kattamaan koko projektin tuotoksia: dokumentteja, malleja, kaavioita jne.
 - *Heterogeeniset artifaktit*

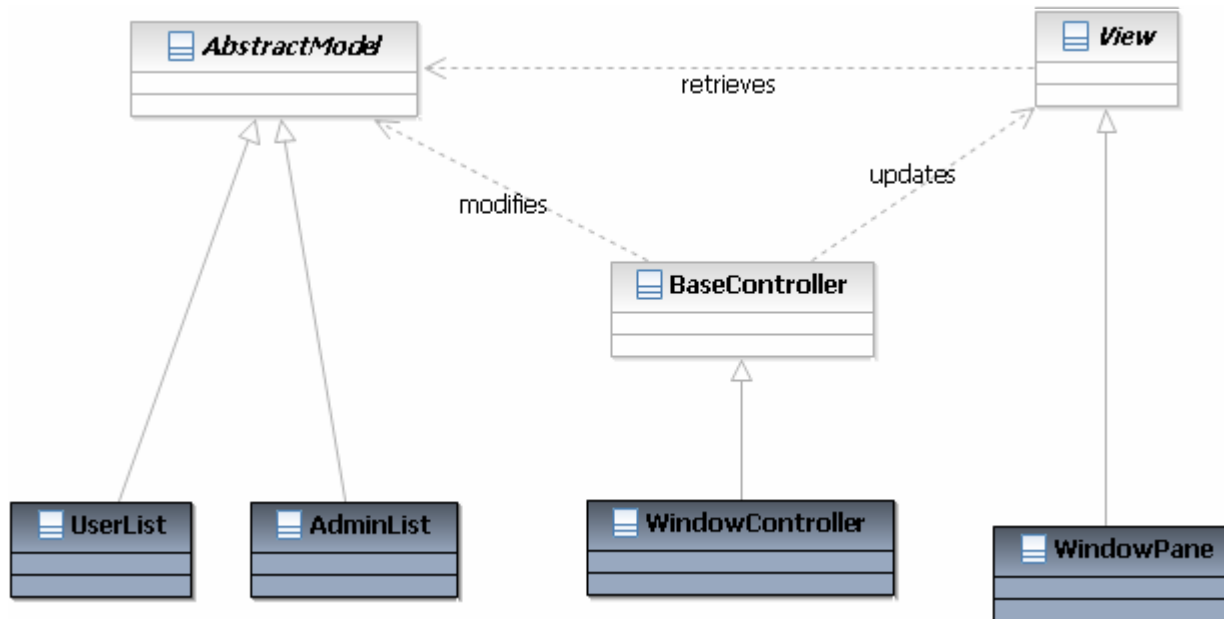
Esimerkki Intressijaosta

- Suunnittelumalli *Model-View-Controller* voidaan jakaa intresseihin *Models*, *Views* ja *Controllers*



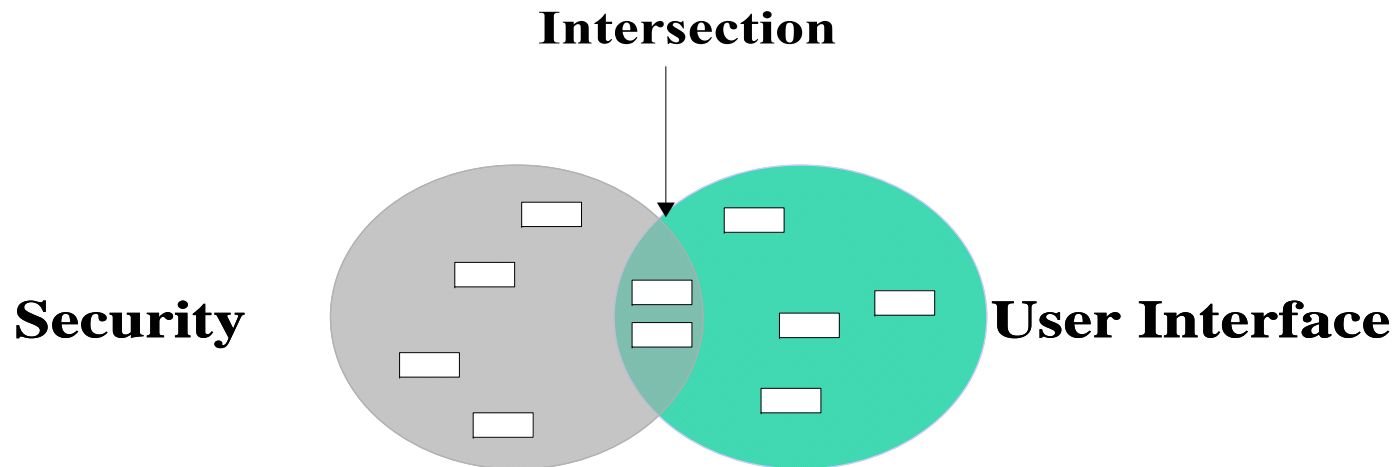
Esimerkki jatkuu

- Voidaan lisäksi määritellä intressit
 - *DesignPattern*
 - *Implementation*



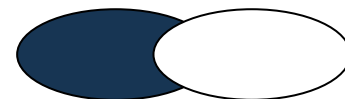
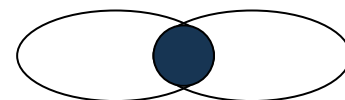
Intressien Kyselykieli

- Intressimappausta voidaan pitää joukkona ohjelmiston artefakteja. Miksipä siis ei käytettäisi matemaattisia joukko-operaatioita intressien keskinäiseen analyysiin?
- **Esimerkki:** Järjestelmästä on tunnistettu ja määritelty intressit *Security* ja *User Interface*. Molemmat intressit sisältävät useita artefakteja. Jos haluamme löytää järjestelmän kaikki käyttöliittymäartefaktit, jotka jollakin tavalla käsittelevät tietoturvaa (esim. kirjautumisruutu, turvallisuusasetukset), voimme ottaa näiden kahden intressin (ts. joukon) välisen **leikkauksen**.



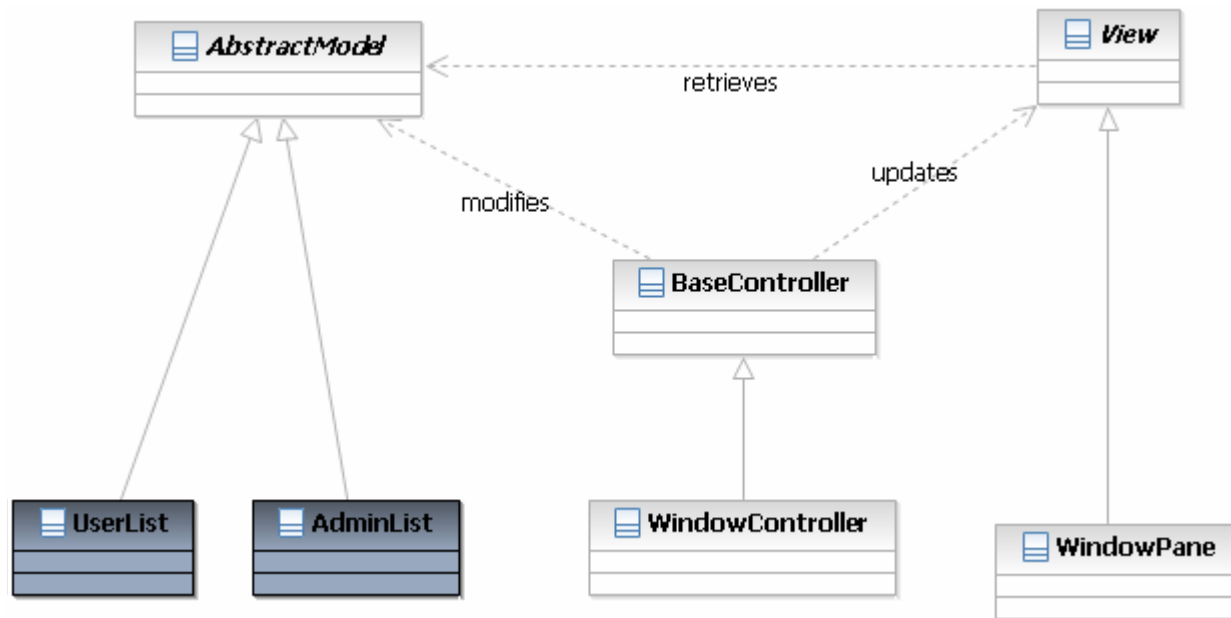
Intressien Kyselykieli

- Matemaattisia joukko-operaatioita
 - Leikkaus ($A \& B$)
 - Kahden intressin yhteiset artifaktit
 - Unioni ($A + B$)
 - Molempien intressien kaikki artifaktit
 - Erotus ($A - B$)
 - A:n artifaktit, jotka eivät kuulu B:hen
- Muita operaatioita
 - "Naapurusto" ($|A$)
 - Kaikki artifaktit, joilla on yhteys johonkin A:n artifaktiin (poislukien A:n omat artifaktit)



Esimerkkejä Intressien käytöstä

- Controllers & Implementation
- (View + Model) – Implementation
- (| DesignPattern) – (Controllers + Views)



Käytännön käyttökohteita

- Intressien keskinäisten suhteiden analyysi
 - Evoluutioaspekti
- Uusien intressien löytäminen
- Keskeisimpien artifaktien tunnistaminen (ts. artefakti, joka löytyy jokaisesta intressistä)
- Riippuvuusanalyysi
- Ohjelmiston rakenteen opettelu itsenäisellä analyysillä.

Työkalutuki

- Concern Manipulation Toolset (CMT), tämänkin esityksen perusta, on kokoelma työkaluja joka mahdollistaa:
 - Intressimappausten avustettu luominen
 - Intressikyselyjen suorittaminen määritetyille intresseille
 - Ohjelmiston evoluutioon regointi
- CMT kehitettiin diplomityönä vuonna 2006 osana Practise-tutkimusprojektia
 - <http://practise.cs.tut.fi>
- CMT kuuluu INARI:n
 - INtegrated ARchitecting envIronment
 - Toteutettu Rational Software Architect –ohjelmiston päälle
 - Monipuolinen suunnitteluympäristö, joka on tarkoitettu käytettäväksi malliperusteisessa ohjelmistotuotannossa.

Yhteenveto

- Intressipohjainen ohjelmistajaottelu on tehokas ohjelmistosuunnittelun ja ylläpidon työkalu, joka tarjoaa monipuolisemman ja joustavamman tavan jaotella ohjelmistoa, kuin modulaarinen jaottelu
- Intressien mukaan tehty jaottelu on ihmismielelle intuitiivinen.
- Intressit nostavat järjestelmän abstraktiotasoa, mutteivät kuitenkaan hukkaa tietoa järjestelmästä.
- Intressien analyysi paljastaa ohjelmistojen ongelmakohtia ja helpottaa entisestään monimutkaisen ohjelmiston sisäistämistä.
- Saatavilla olevat työkalut ovat lupaavia ja niitä on käytetty teollisuudessa tehdyissä kokeiluissa onnistuneesti.

A woman with dark hair, wearing a light grey suit, stands centrally between two white, modern-style chairs. She is holding a light-colored folder or book in front of her. The background is a plain, light grey wall.

Kiitos!

tommi.reinikainen@sysopendigia.com
www.sysopendigia.com