Oliopohjaisten ohjelmien dokumentointi

Henri Leisma

26.3.2003

Joensuun yliopisto
Tietojenkäsittelytiede
Viestinnän raportti

Tiivistelmä

Tämän raportin tarkoituksena on kuvata yksinkertaisella tavalla oliopohjaisten dokumenttien laatimisessa ja käytössä huomioon otettavia tekijöitä. Näkökulma on pyritty pitämään avoimena erilaisille dokumenttien toteutustavoille. Oliopohjaisuuden soveltuvuutta sekä saavutettavia hyötyjä ja vaikutuksia on esitelty erityyppisten dokumenttien käyttäjien näkökulmista. Oliopohjainen ajattelu on tullut dokumentointiin ohjelmistojen hallinnan kautta. Oliopohjaisuutta esitetään ratkaisuksi monimutkaisuuden hallintaan, laadun parantamiseen, vähentämään ylläpidosta aiheutuvia kuluja sekä parantamaan uudelleenkäytettävyyttä.

ACM-luokat (ACM Computing Classification System, 1998 version): D.2.7, H.3

Avainsanat: oliopohjainen, dokumentaatio, järjestelmä, hallinta, ylläpito

Sisältö

1 Johdanto

Oliopohjaisella ohjelmien dokumentoinnilla tarkoitetaan tyypillisesti dokumentin sisältämien tietojen kapselointia pienempiin osiin, osien linkittämistä haluttuihin kohtiin sekä piilottamista niiltä laatijoilta ja lukijoilta, joita kyseinen kohta ei koske.

Tavoitteena oliopohjaisuuden puolesta puhujilla on saada karsittua päällekkäisyyksiä jatkuvasti monimutkaistuvien suurten ohjelmistoprojektien dokumenteista ja samalla nopeuttaa niiden valmistumista. Työvoiman suuri vaihtuvuus ja projektityössä ilmenevä tarve uusien ratkaisujen nopeaan sisäistämiseen ovat osaltaan edistäneet dokumentoinnin menetelmien ja työkalujen kehittämistä.

Oliopohjaisesti toteutetut dokumentit koostuvat luokista ja olioista, samaan tapaan kuin oliopohjaiset ohjelmistotkin. Tämä ei lähtökohtana kuitenkaan ole täysin ongelmaton. Erityisesti käyttäjille suunnatuissa dokumenteissa olion tunteminen ei riitä vaan tarvitaan tietoa myös sen esiintymisympäristöstä. Järjestelmädokumentoinnissa oliopohjaisuus taas on luonnollinen valinta, koska se on muita dokumentointitapoja lähempänä itse ohjelmointiongelman ratkaisua.

Oliopohjaisuuden nähdään tuovan hyötyä erityisesti ohjelmistoprojektien hallintaan. Muita etuja ovat laatu, helpompi ylläpito sekä mahdollisuus dokumentin osien uudelleenkäyttöön.

2 Oliopohjaisuus

Tässä osiossa esitellään eräitä oliopohjaisuuteen liittyviä keskeisimpiä käsitteitä ja periaatteita. Oliopohjaisuus on pohjimmiltaan tapa hahmottaa ja ratkoa ongelmia, joten sen kantaviin ajatuksiin tutustuminen on hyödyksi tämän raportin myöhempien osien ymmärtämistä ajatellen.

2.1 Oliopohjaisuuden käsite

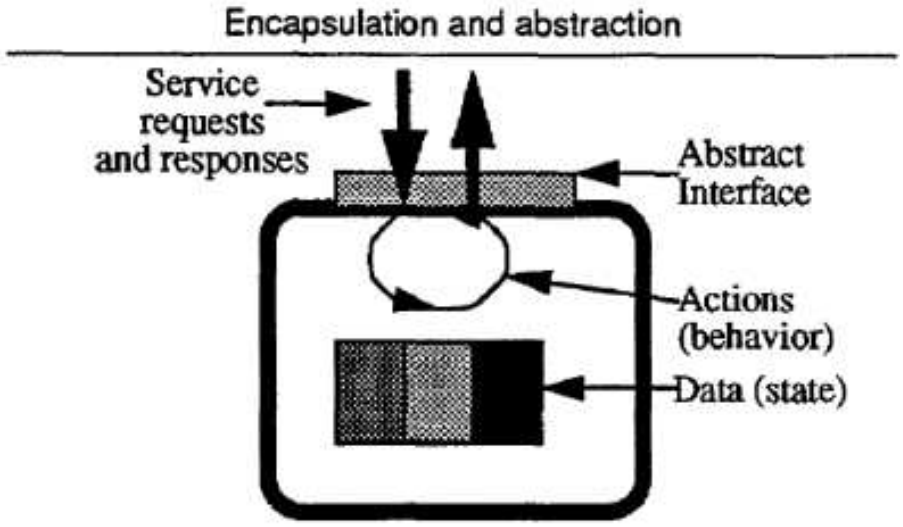
Oliopohjaisen ohjelmoinnin periaatteet ja tekniikat kehitettiin alunperin ohjelmistojen monimutkaisuuden hallintaan, laadun parantamiseen, vähentämään ylläpidosta ai-

heutuvia kuluja sekä parantamaan uudelleenkäytettävyyttä. (Matthews ja Grove, 1992)

Oliopohjainen ajattelu on jatkoa jo 60-luvulla havaitulle tarpeelle tiedon abstraktimmasta esittämisestä tietojenkäsittelyssä. Tietojenkäsittelytieteen käsitys oliopohjaisuudesta ja sen filosofiasta perustuu ensimmäisiin, usein puutteellisesti toteutettuihin oliopohjaisiin ohjelmointikieliin. Matthews ja Grove (1992) pitävät tätä valitettavana, sillä lähtökohtaisesti oliopohjaisuus on tapa esittää ja ratkaista ongelmia.

2.2 Oliopohjaisuuden periaatteita

Käyttäjän näkökulmasta tavoitellaan korkeampaa *abstraktiotasoa* ja pyritään *tiedon kapselointiin* (abstraction and encapsulation). Oliopohjaisuuden perusperiaatteita on, että kaikki ongelmakentän tiedot ja toiminnot ovat samantyyppisille ongelmille samoja ja ne esitetään yhteisessä *luokassa* (class). Luokka kuvaa siis mahdolliset tilat ja *oliot* (object) ovat ilmentymiä todellisista ongelmakentän tiloista.



Kuva 1: Tiedon kapselointi (Matthews ja Grove, 1992).

Tieto- ja *operaatioyksiköiden itsenäisyys* ja pitäytyminen sellaisina ongelmien ja tiedon muuttuessa (Independence and persistence) tarkoittaa, että oliot ovat aina yksilöllisesti tunnistettavissa ja muokkaavat tilaansa itsenäisesti.

Viestien välitys (Message passing) mahdollistaa objekteille palveluiden tarjoamisen ja pyytämisen muilta objekteilta. Ilman viestien välitystä kaikkien objektien tulisi pystyä

tarjoamaan tarvitsemansa palvelut itse.

Periytyminen (Inheritance) on erittäin olennainen mekanismi tietoyksiköiden uudelleenkäytössä. Erityisesti oliopohjaisuudessa (aliluokkien toteuttamisessa) periytyminen on pääasiallinen olemassaolevan tiedon hyödyntämis- ja muokkaamistapa. Periytyminen on tullut tietojenkäsittelytieteeseen tietämyksen esittämisen tutkimuksen kautta.

Yhtenäisyyden (Homogeneity) vaatimus vahvistaa abstraktia tiedon esittämistä ja kapselointia. Periaatteessa oliopohjaisessa järjestelmässä kaikki tulisi esittää olioina, käytännössä tarvitaan kuitenkin primitiivisiä tyyppejä, joita ei enää esitetä olioina.

Matthews ja Grove (1992) esittävät edellisten lisäksi *suunnittelun yksinkertaisuutta* (design simplicity). Sen painoarvon hän näkee yhtä suurena kuin edellä mainittujenkin ominaisuuksien. Perusteluna esitetään oliopohjaisuuden mukanaan tuoma helpotus siirtymään ongelman analyysistä oliopohjaiseen toteutukseen sekä ratkaisujen yksinkertaistuminen.

3 Oliot dokumentoinnissa

Oliopohjaista ajattelua on kehitelty aluksi muilla tieteenaloilla kuin tietojenkäsittelytieteessä. Dokumentointiin se on tullut vähitellen ohjelmistojen monimutkaistumisen ja sen hallinnan kautta. Oliopohjaisuutta esitetäänkin ratkaisuksi erityisesti monimutkaisuuden hallintaan. Muita osa-alueita joihin sen nähdään tuovan etua ovat ohjelmistoprojektien laatu, ylläpito ja komponenttien helpompi uudelleenkäytettävyys.

3.1 Dokumentointiin vaikuttavia tekijöitä

Dokumentin hallittavuus riippuu paljon sen *monimutkaisuudesta* (complexity). Selkeät ja johdonmukaiset, usein puumaista hierarkiaa seuraavat, dokumentit ovat osoittautuneet helpoiten hallittaviksi. Monimutkaisuuteen vaikuttavia tekijöitä ovat esimerkiksi kirjoittajien määrä, dokumenttityyppien määrä ja valitut kohderyhmät.

Selkeä ja hyvin suunniteltu rakenne helpottaa myös *ylläpidettävyyttä* (maintainability). Ohjelmistotuotteen virheiden korjaaminen ja toiminnalliset muutokset täytyy pystyä

siirtämään myös dokumentaatioon. Ajantasalla pysyminen ja suunnitellun rakenteen säilyttäminen ovat keskeisiä tekijöitä ylläpidettävyydessä.

Dokumentin tulee olla *käytettävyydeltään* (usability) hyvä ja asiasisällön suunnattu määritellylle kohderyhmälle. Käyttäjää ei useinkaan kiinnosta järjestelmän sisäinen rakenne ja ylläpitäjää taas yksityiskohtaiset ohjelman toimintojen käyttöohjeet. Käytettävyys on myös oliopohjaisen dokumentoinnin jonkin asteinen kompastuskivi, koska esimerkiksi käyttötapausten dokumentointi oliopohjaisesti vaatii olion kontekstin (viitekehys) tuntemista. (Matthews ja Grove, 1992)

3.2 Oliopohjaisuuden tuomat ominaisuudet

Eräs tärkeimmistä oliopohjaisuuden mukanaan tuomista ominaisuuksista on, että dokumentin rakenne on helpommin ymmärrettävissä. Navigointi dokumentin eri osien välillä helpottuu, mikäli rakenne on selkeä. Samalla uusien dokumenttien kokoaminen olemassa olevista palasista muuttuu helpommin hallittavaksi. Uusien luotavien dokumenttien rakenne myös yhtenäistyy ja käytännössä kun käyttäjät oppivat uuden ajattelutavan, niin myös tiedonhaku muista oliopohjaisesti toteutetuista dokumenteista helpottuu. (Matthews ja Grove, 1992)

3.3 Oliopohjaisuus erityyppisissä dokumenteissa

Järjestelmän dokumentoimiseen (system documentation) oliopohjaisuus sopii erinomaisesti. Dokumentti rakentuu täysin samanlaisen hierarkian mukaan kuin ohjelman tiedot ja toiminnot. Dokumentin sisältö voidaan jopa säilyttää ja poimia lähdekoodin kommenteista, jolloin se myös varmimmin pysyy ajantasalla itse koodin muuttuessa.

Käyttötapausten dokumentoiminen (user documentation) taas on merkittävästi vaikeampaa. Käyttötapaukset eivät noudata samanlaista selkeää hierarkiaa kuin järjestelmän toiminnot vaan käytännössä aina tulee tuntea myös olion viitekehys (object context), eli missä ympäristössä olio esiintyy. Tämä ei ole täysin ylitsepääsemätön este, mutta edellyttää vähintään erillistä käyttötapaushierarkiaa, jossa toisistaan muuten riippumattomia luokkia periytetään. Esimerkiksi juuston höyläämisen tulisi esiintyä voileivän teon aliluokkana, jotta se olisi helposti kytkettävissä tämän luokan tapahtumiin. (Sametinger, 1994)

4 Hyödyt käytännössä ja esimerkkejä toteutuksesta

Oliopohjaisuudella saavutettavina etuina on nähty erityisesti ohjelmistoprojektien hallinnan helpottuminen, koko ohjelmistotuotteen laadun paraneminen, selkeän ja modulaarisen rakenteen ansiosta helpottunut ylläpito sekä mahdollisuus dokumentin osien uudelleenkäyttöön.

4.1 Tiedon haku ja dokumenttien hallittavuus

Oliopohjaisuudella saavutetaan muutamia etuja tiedon haussa dokumentaatiosta. Tietoa voidaan luonnollisesti hakea manuaalisesti paperidokumenteista, kuten ohjekirjat tai käyttöohjeet, mutta vain sähköisissä dokumenteissa on mahdollista toteuttaa myös automatisoitua tiedon poimimista. Automatisointi helpottuu, kun dokumentaation rakenne on yhtenäinen ja osioiden riippuvuudet toisistaan selkeästi tiedossa. Oliopohjaisuus tukee näistä vaatimuksista kumpaakin.

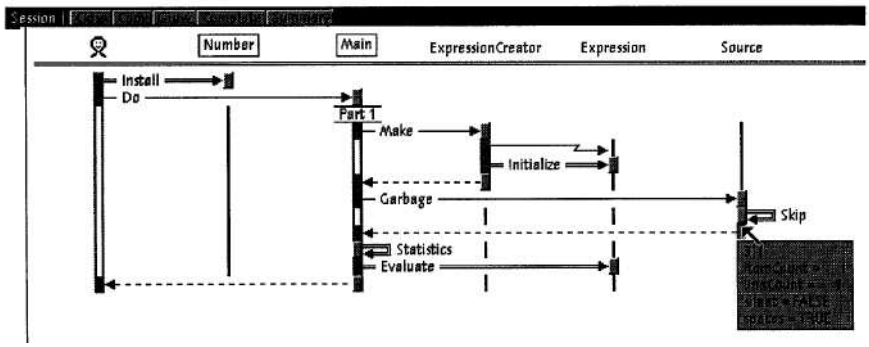
Kun dokumentin rakenne oliopohjaisissa dokumenteissa vastaa pitkälti ratkaistavan ongelman rakennetta, on ihmisen helpompi hahmottaa ohjelmistolla ratkaistu ongelmakenttä kokonaisuutena. Tämän ominaisuuden painoarvo kasvaa, mitä laajemmiksi ja monimutkaisemmaksi ohjelmistot kehittyvät.

Ohjelmiston tavoin oliopohjaisessa toteutuksessa dokumentaatioon jää myös yleensä vähemmän virheitä komponenttien uudelleenkäytön myötä. Kun virhe on kerran havaittu ja korjattu, korjaus periytyy myös muihin tätä osaa käyttäviin osadokumentteihin.

Dokumentin ymmärrettävyys ja muokattavuus paranevat niin kirjoittajan kuin käyttäjien näkökulmasta. Osa tästä tulee dokumentaation siirtymisestä sähköiseen muotoon, osa olio-ajattelun käytöstä. Dokumentin rakenteen suunnittelu ja toteuttaminen onnistuvat joskus jopa samoilla työkaluilla kuin oliopohjainen ohjelmointikin. Tiedon haku ja dokumentin selattavuus paranevat, kun eri osien riippuvuudet on otettu aiempaa tarkemmin huomioon. Myös siirtyminen dokumentin osien välillä on vaivatonta. Ylläpidettävyyden kannalta ei samaa havaittua virhettä tarvitse useinkaan muuttaa eri dokumentteihin erikseen. (Briand et al., 1997)

4.2 Työkaluja oliopohjaiseen dokumentointiin

Muutamissa oliokielillä ohjelmointiin tarkoitetuissa *kehitysympäristöissä* (integrated development environment) on toteutettu tieteellisissä julkaisuissa esitettyjä olio-ohjelmoinnin apuohjelmia. Pääasiassa työkalut ovat kuitenkin vain tutkijoiden julkaisuissa käsittelemään tiettyyn tehtävään suunniteltuja ohjelmia.

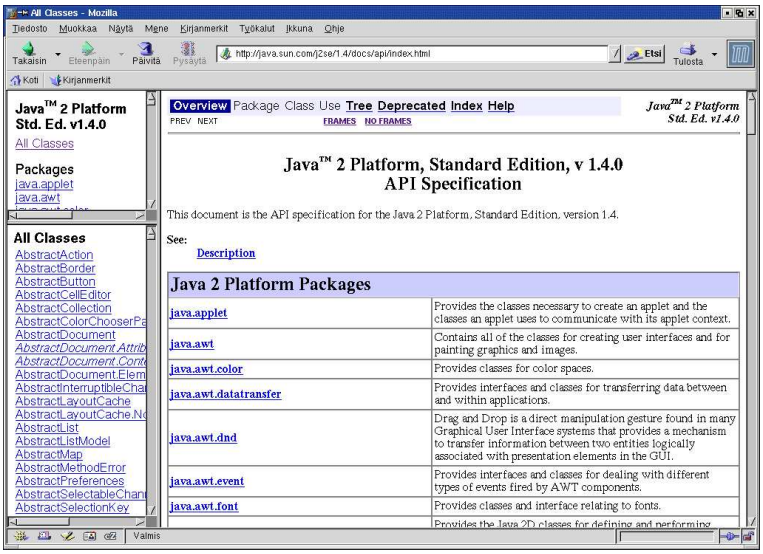


Kuva 2: Apuväline hahmottamiseen (Koskimies ja Mössenböck, 1996).

Scene on yksi oliopohjaisten ohjelmien dokumentoinnin helpottamiseen laadittu työkalu. Sillä voidaan luoda tapahtumasekvenssi-kaavioita (uimarata-kaavio), joilla olioiden dynaamista käyttäytymistä voidaan seurata. Staattisten ohjelman osien seuranta on ihmiselle merkittävästi helpompaa, mutta monen olion yhdessä luoman dynaamisen käytöksen seuranta onnistuu yleensä vain laadittavien käyttöesimerkkien avulla. Scene-ohjelmalle on mahdollista antaa parametreina oletettuja syötteitä ja se havainnollistaa olioiden tiloja ja viestien välitystä piirtämällä käyttötapauksesta kaavion. (Koskimies ja Mössenböck, 1996)

4.3 Hypertekstin ja oliopohjaisuuden suhteesta

Oliopohjaisuus ja hyperteksti-dokumentit liittyvät monilta osin yhteen ja tuovat dokumentointiin osittain samantyyppisiä etuja. Yhtenäisyyksiä on hierarkisen rakenteen ja osien välisten riippuvuuksien saralla. Hypertekstissä ei suoraan ole mahdollista toteuttaa periytyvyyttä, mutta esim. Apache www-palvelimen avulla dokumentin haluttuun kohtaan voidaan poimia toinen tiedosto - käyttäjälle täysin läpinäkyvästi. Uusimmat html-kielen muotoiluun luodut apuvälineet (css ja javascript) mahdollistavat dokumentin laatimisen kerroksittain ja osien piilottamisen ja näyttämisen vain määritellyissä yhteyksissä.



Kuva 3: Javadoc-ohjelman tuottamaa järjestelmädokumentaatiota.

Hyvä esimerkki toimivasta oliopohjaisten ohjelmien dokumentointityökalusta on Java-kielen javadoc. Se on suunniteltu erityisesti järjestelmän dokumentointiin ja toimii poimimalla ohjelmakoodin kommenteista dokumenttien sisällön. Dokumentaatio tulee kokemusteni mukaan varmimmin tehdyksi ja päivitetyksi, kun se on jatkuvasti läsnä myös koodia kirjoitettaessa. Dokumenttiin poimitaan lähdekoodin seasta funktioiden ja parametrien kuvauksia, tietoja tekijästä ja muuta kirjoittajan tuottamaa kuvausta järjestelmän toiminnasta. Työkalu luo dokumentin hyperteksti-sivuiksi ja jakaa sen luokkien jaon mukaisesti osiin. Tiedonhaku ja siirtyminen dokumentista onnistuu automaattisesti luodun luokkien metodien linkkilistan avulla.

5 Yhteenveto

Mikäli oliopohjaisuutta aiotaan soveltaa dokumentaatioon, tulee käytössä olla dokumenttien hallintaan suunniteltuja työkaluja. Dokumentin rakenteen monimutkaistuminen oliopohjaisuuden myötä on aiheuttanut tarpeita entistä laajempaan kuvien ja havainnollistavien esimerkkien käyttöön. Tämä on puolestaan parantanut ohjelmoijien ja dokumentoijien koko projektin ymmärtämistä ja siten usein parantanut lopputulosta ja keventänyt työtaakkaa.

Oliopohjaisuus on ollut merkittävä askel itse ohjelmistojen monimutkaisuuden hallinnalle ja se on luonnollisena jatkona saanut jalansijaa myös dokumentoinnis-

sa. Etuina dokumenttien kirjoittajat ovat kokeneet parantuneen ohjelmistoprojektien hallinnan. Ylläpitäjien näkökulmasta pieniin osiin jakautunut ja hierarkinen rakenne on helpottanut muutosten hallintaa. Samantyyppisiä ohjelmistoja toteuttavissa yrityksissä mahdollisuutta myös dokumentin osien uudelleenkäyttöön on arvostettu.

Viitteet

Briand, L.C., Bunse, C., Daly, J.D. (1997) *An experimental evaluation of quality guidelines on the maintainability of object-oriented design documents*, seventh workshop on Empirical studies of programmers

Koskimies, K., Mössenböck, H. (1996) *Scene: using scenario diagrams and active text for illustrating object-oriented programs*, 18th international conference on Software engineering

Matthews, S.,Grove, C. (1992) *Applying object-oriented concepts to documentation*, 10th annual international conference on Systems documentation

Sametinger, J. (1994) *Object-oriented documentation*, ACM SIGDOC Asterisk Journal of Computer Documentation, 18(1)