Toteutusdokumentti

Kaapo - Kaavioiden piirto-ohjelma

Helsinki 1.9.2005

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Ilari Heikkinen
Allan Holsti
Tero Kallioinen
Kristian Ovaska
Mikko Paltamaa
Hannu-Pekka Rajaniemi

Asiakas

Inkeri Verkamo

Johtoryhmä

Juha Taina
Sampo Yrjänäinen

Kotisivu

`http://www.cs.helsinki.fi/group/oops`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
1.0	1.9.2005	Julkaistava versio
0.1	23.8.2005	Dokumenttirunko luotu

Sisältö

1 Johdanto

Oops on Helsingin yliopiston tietojenkäsittelytieteen laitoksella toteutettava ohjelmistotuotantoprojekti. Kaapo, eli Kaavioiden piirto-ohjelma on geneerinen, käyttäjän tarpeisiin mukautuva piirtotyökalu. Projektiin liittyvä materiaali on saatavissa ryhmän kotisivulta osoitteesta

`http://www.cs.helsinki.fi/group/oops`

Tässä dokumentissa kuvataan toteutuksessa käytetyt ratkaisut, esitetään toteutuksen jälkeen päivitetyt luokkakaaviot ja tuodaan esille jatkokehityksen kannalta hyödyllistä tietoa, kuten toteuttamatta jääneet toiminnot ja ideoita mahdollisista hyödyllisistä lisätoiminnoista. Ohjelmakoodi on dokumentoitu Javadoc-kommentein ja dokumentaatio on saatavissa ryhmän kotisivulta sekä ohjelman lähdekoodi-jakelupakkauksesta.

2 Toteutusratkaisut

Tässä luvussa on esitetty toteutusratkaisut, jotka ovat syntyneet tai muuttuneet suunnitteluvaiheen jälkeen.

2.1 Arkkitehtuuri

Ohjelma on jaettu osiin, joista kullakin on oma tehtävänsä. Kyseessä ei ole varsinaiset itsenäiset osajärjestelmät, mutta osat on pyritty pitämään mahdollisimman itsenäisinä, jotta kutakin olisi helppo muuttaa koskematta muihin. Osia kutsutaan tässä komponenteiksi.

Kuten editoreissa yleensä, kaikki ohjelman toiminta käsittelee yhtä isoa tietorakennetta, projektia. Projekti sisältää kaavioita, jotka puolestaan sisältävät elementtejä ja yhteyksiä.

Ohjelman arkkitehtuuri perustuu pitkälti observer-suunnittelumallin käyttöön. Siinä esitys ja sisältö on eriytetty niin, että tietorakennetta tarkkailee yksi tai useampia tarkkailijoita. Tarkkailun kohde sisältää listan tarkkailijoistaan ja ilmoittaa niille muutoksista sisällössään. Tarkkailijat rekisteröivät itsensä kohteensa tarkkailijoiksi. Observer-suunnittelumalli muistuttaa pitkälti Javan listener-mallia.

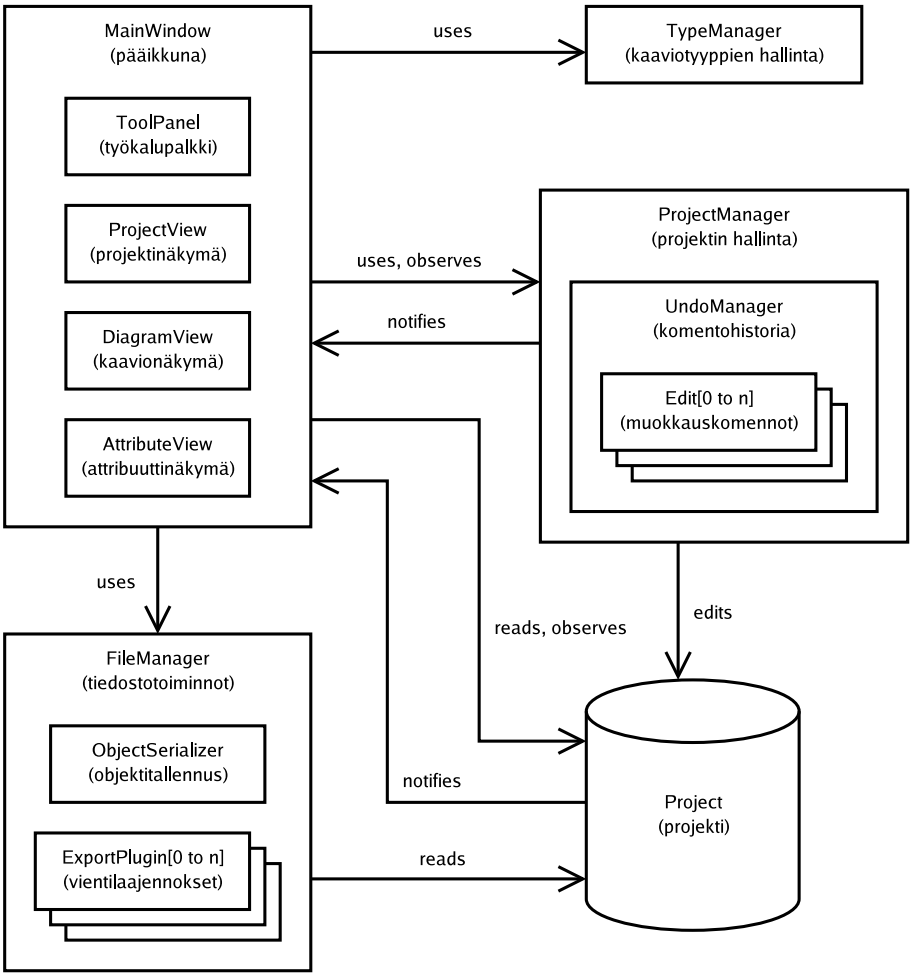
Käyttöliittymä rekisteröi tarvittavat käyttöliittymäelementit projektin kuuntelijoiksi ja saa siten tiedon muutoksista tietorakenteessa ja voi päivittää muutokset näkyville käyttäjälle.

Projektin hallintaan on oma komponenttinsa, joka pitää yllä komentohistoriaa sekä valintoja ja muuta projektin muokkaamiseen liittyvää tietoa.

Käyttöliittymä ei koskaan suoraan muokkaa projektia, vaan kaikki muutokset tapahtuvat luomalla ja lähettämällä edit-olioita projektinhallinnalle suoritettavaksi. Näin käyttöliittymän ei tarvitse tietää projektin toteutuksen yksityiskohdista eikä myöskään ylläpitää komentohistoriaa.

Projektien tallennus ja avaus sisältyy tiedostonhallinta-komponenttiin. Se on toteutettu Javan oliotallennuksella, mistä seuraa se, ettei kyseisten metodien ei tarvitse tuntea projektin toteutusta.

Seuraavassa päivitetty arkkitehtuurikaavio.

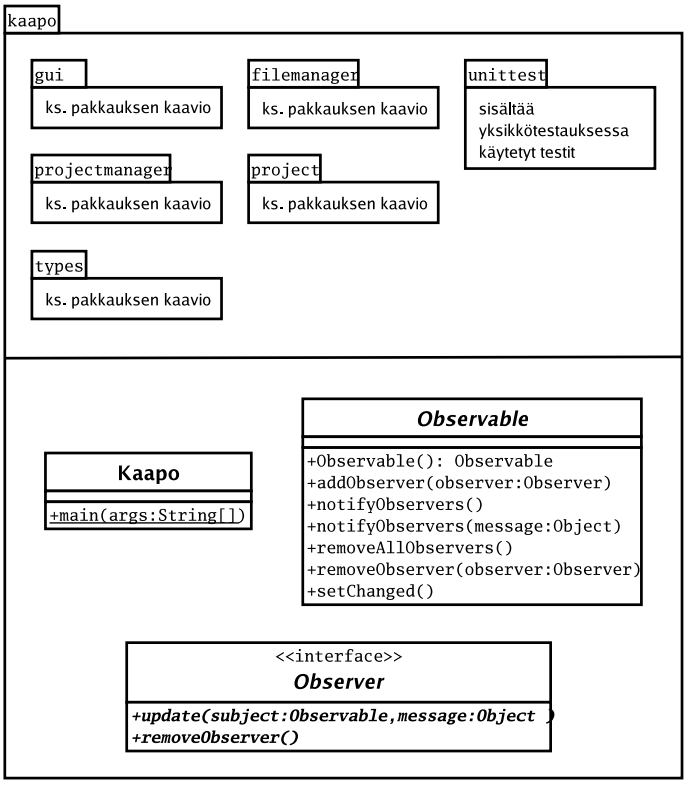


Kuva 1: Päivitetty arkkitehtuurikaavio

3 Päivitetyt komponenttikuvaukset

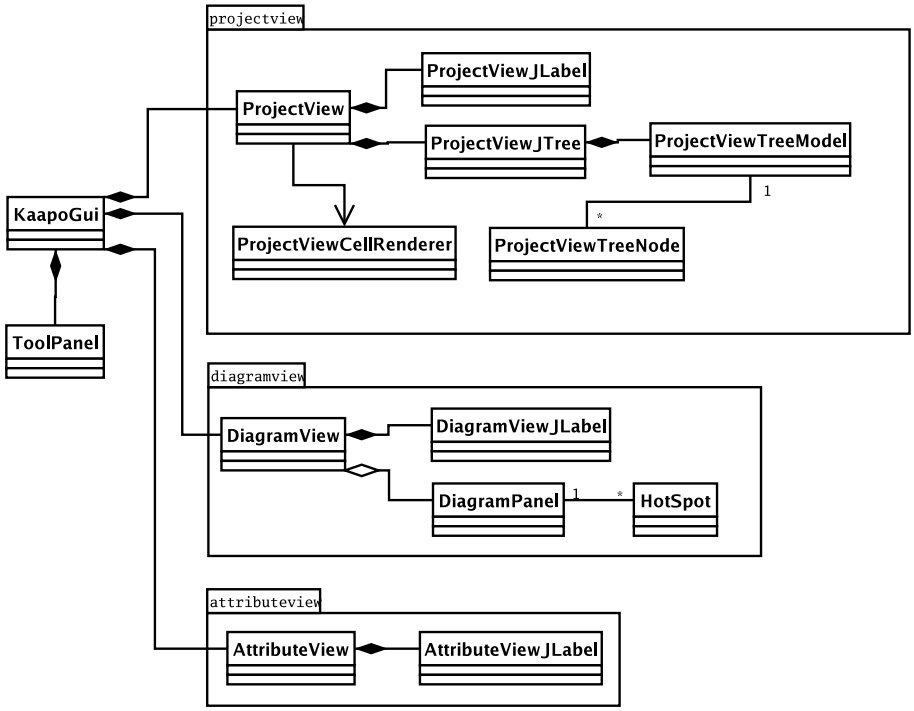
Tässä luvussa esitellään ohjelman luokkakaaviot pakkauksittain. Luokkakaaviot eivät ole täydellisiä, vaan sisältävät vain olennaiset osat pakkausten sisällöstä ja luokkien toteutuksesta sekä yhteyksistä.

3.1 Kaapo



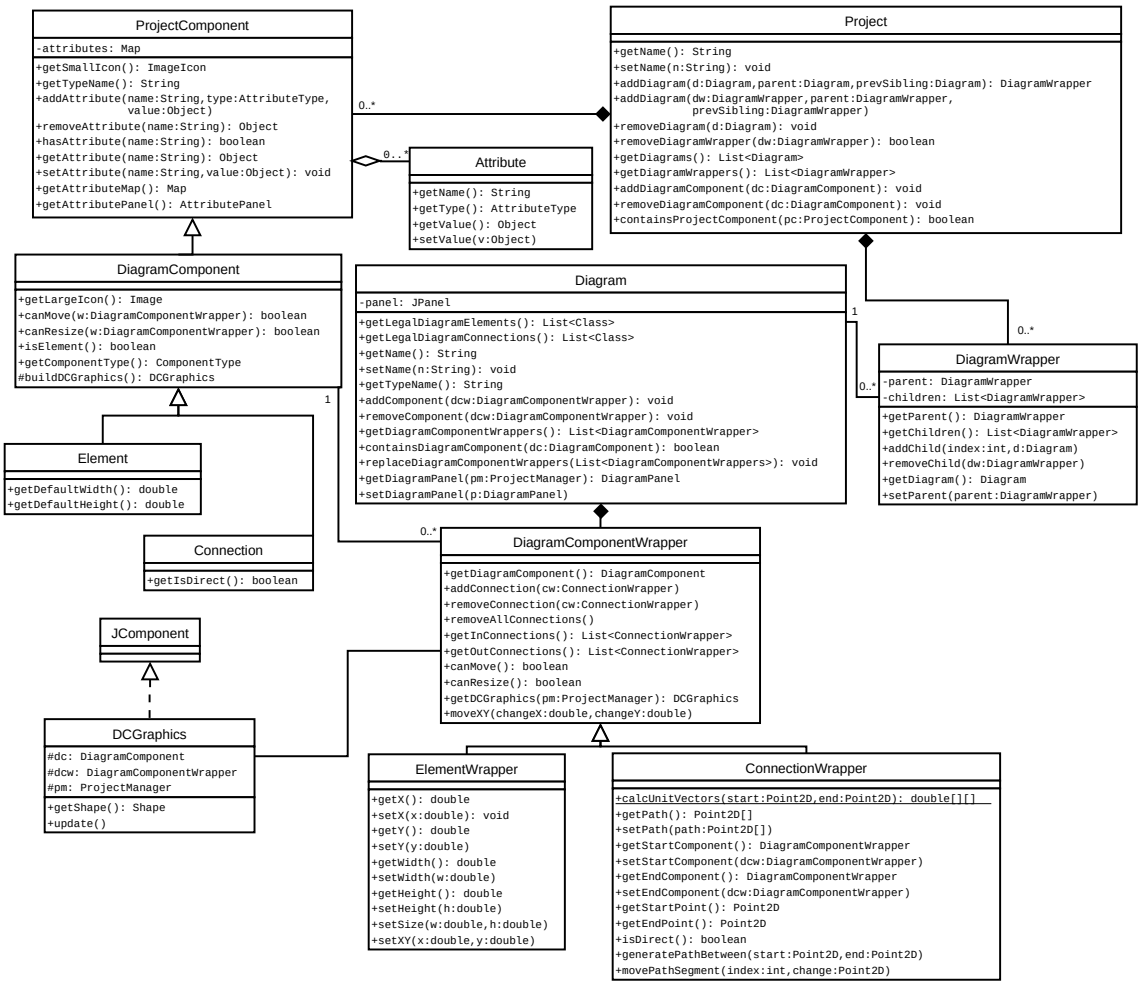
Kuva 2: `kaapo`-pääohjelmapakkauksen luokkakaavio

3.2 Gui

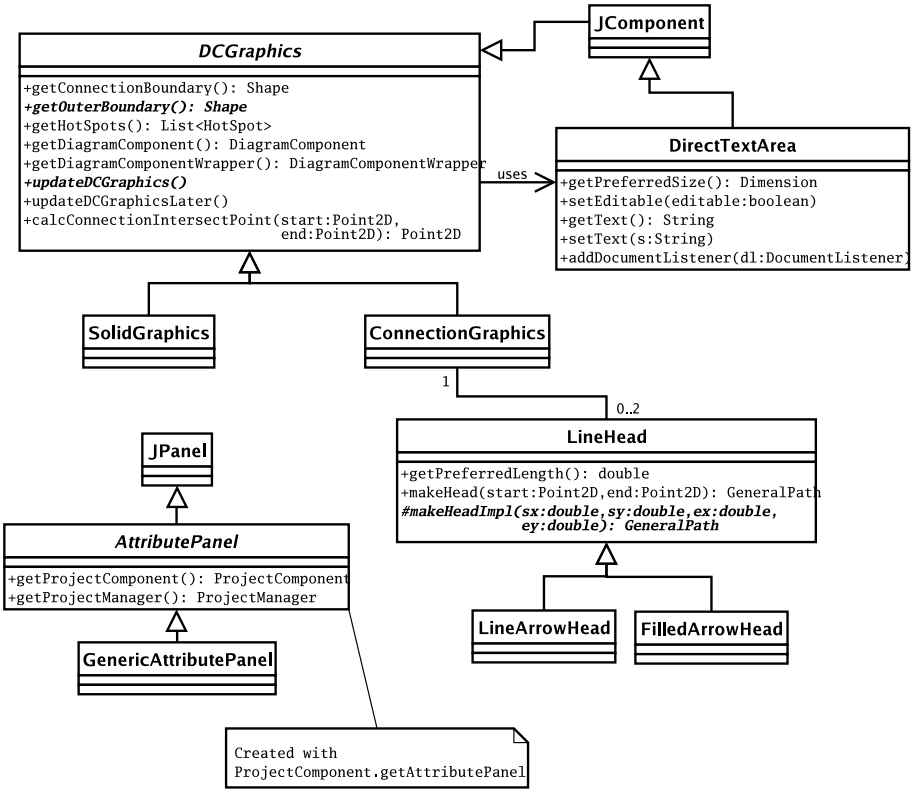


Kuva 3: `gui`-pakkauksen luokkakaavio

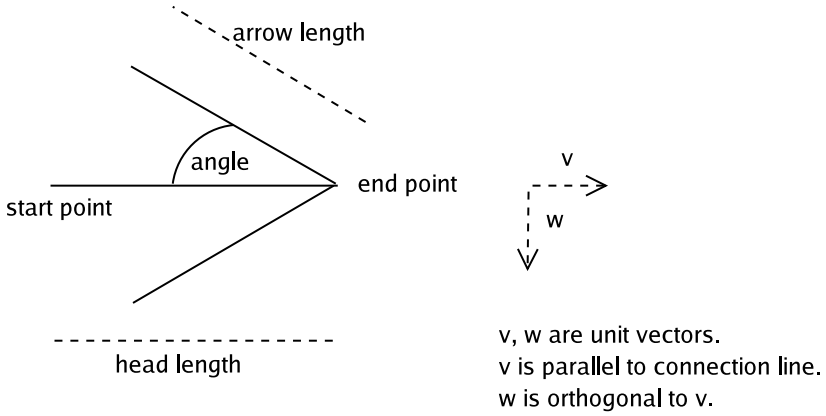
3.3 Project



Kuva 4: `project`-pakkauksen luokkakaavio

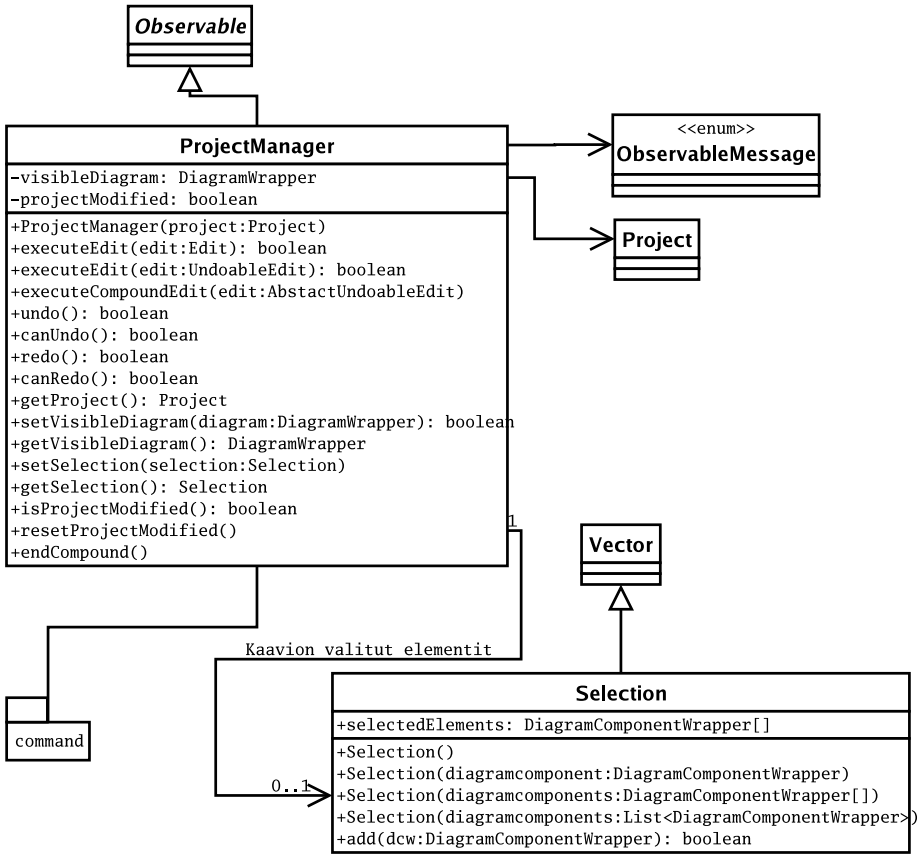


Kuva 5: `project.graphics`-pakkauksen luokkakaavio

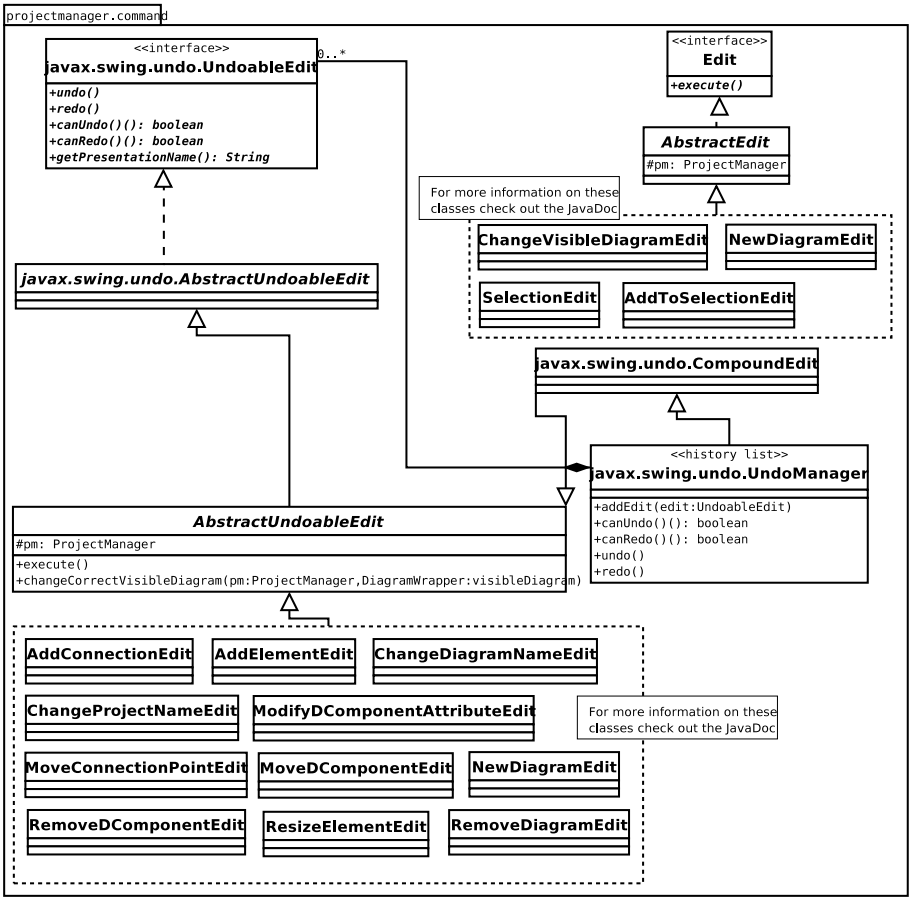


Kuva 6: Nuolenpääluokkien (luokka `kaapo.project.graphics.LineHead` ja sen aliluokat) käyttämät geometriset abstraktiot. (Orthogonal=kohtisuora)

3.4 Projectmanager

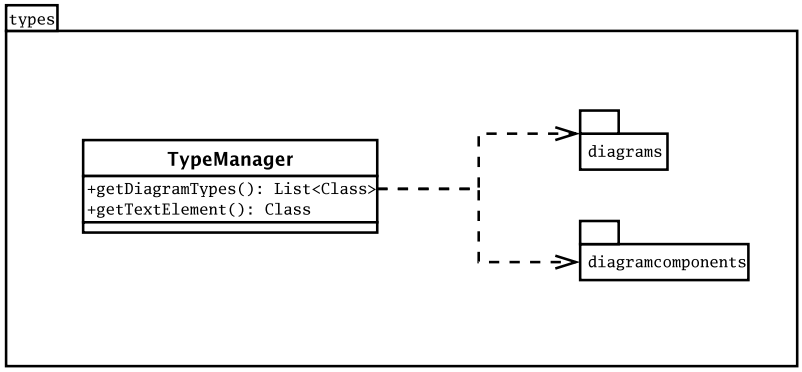


Kuva 7: `projectmanager`-pakkauksen luokkakaavio



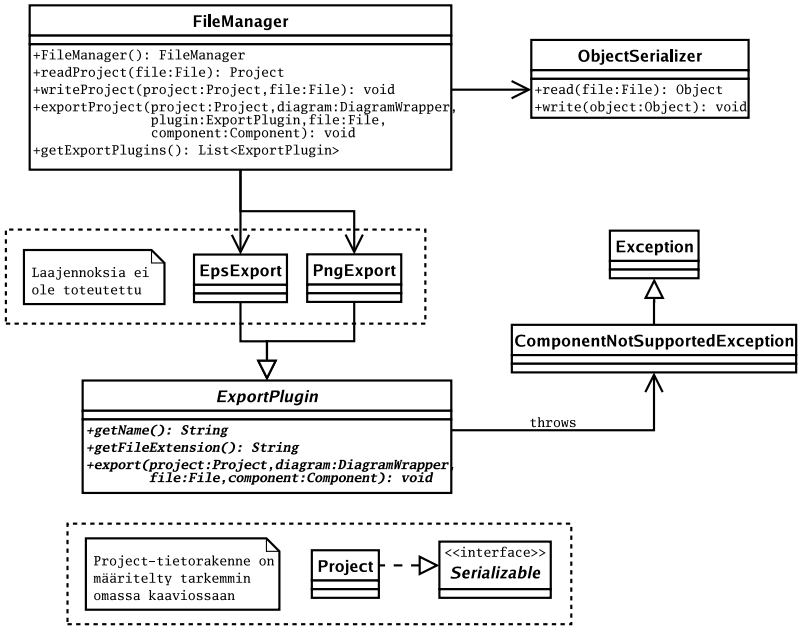
Kuva 8: `projectmanager.command`-pakkauksen luokkakaavio

3.5 Typemanager



Kuva 9: `typemanager`-pakkauksen luokkakaavio

3.6 Filemanager



Kuva 10: `filemanager`-pakkauksen luokkakaavio

4 Jatkokehitys

Tämä luku on tarkoitettu lähtökohdaksi ohjelmiston jatkokehitykselle. Luvussa on parannusehdotuksia jo toteutettuihin ominaisuuksiin sekä luettelo toteuttamatta jääneistä omi-

naisuuksista.

Erillisessä testausdokumentissa on lueteltu ohjelmiston tunnetut virheet. Joidenkin virheiden kohdalla testausraportissa on myös korjausehdotuksia.

4.1 Toteuttamatta jääneet ominaisuudet

4.1.1 Syntaksisäännöt

Elementtien ja yhteyksien syntaksisääntöjä ei ole toteutettu. Ainoa olemassaoleva syntaksisääntö on se, mitä elementtejä ja yhteyksiä kaaviotyyppiin voi kuulua. Muille syntaksisäännöille ei myöskään ole valmista paikkaa luokkahierarkiassa. Luultavasti ne tulevat joko `Diagram`-luokkaan tai kokonaan pakkaukseen, mikä saattaa olla selkeämpi vaihtoehto.

Muita syntaksisääntöjä olisivat ainakin:

- mihin kaaviokomponentteihin yhteys saa olla yhteydessä
- pitääkö elementillä olla tietynlaisia yhteyksiä (esim. UML-käyttötapauselementillä oltava yhteyksiä actor-elementtiin)
- luokkakaaviossa ei ole kahta samannimistä luokkaa. Yleistettynä: tietyssä elementtien joukossa jokin attribuutin arvo on uniikki.

Käyttöliittymässä syntaksivirheet on tarkoitus näyttää visuaalisesti värjäämällä virheelliset kaaviokomponentit punaisella. Osa syntaksitarkistuksesta on tarkoitus tehdä reaaliajassa, mutta jos löytyy sellaisia syntaksisääntöjä, joiden tarkastus on mielekästä vasta kun kaavio on lähes valmis, tämä tehdään vain eri komennosta.

4.1.2 Kaavioiden ohjelmallinen analyysi ja muokkaus

Ominaisuutta ei toteutettu eikä sille ole valmista paikkaa. Mahdollisia analyysejä ovat esim. jonoverkkoihin liittyvät laskutoimenpiteet. Esimerkki muokkauksesta olisi UML-sekvenssikaavion (puoli)automaattinen muuttaminen UML-yhteistyökaavioksi (kaaviot kuvaavat samat asiat hieman eri tavalla).

4.1.3 GUI: elementti useaan kaavioon

Tietorakenne tukee saman elementin lisäämistä useaan kaavioon, mutta käyttöliittymässä ei ole sille toiminnallisuutta. Toiminnon voisi tehdä esim. kaaviopuussa siten, että elementin voi raahata hiirellä toisen puussa olevan kaavion kohdalle. Tällöin elementin paikka uudessa kaaviossa määräytyy automaattisesti ja käyttäjä joutuu itse siirtämään sen oikeaan paikkaan. Toinen vaihtoehto on jonkinlainen copy-paste -ominaisuus.

4.1.4 Yhteydet yhteyksien välillä

Ominaisuus on toteutettu osittain, mutta ei ole käytännössä käyttökelpoinen. Tietorakenne tukee ominaisuutta ja kälissä yhteyden lisääminen on mahdollista, mutta kun pääteyhteksiä liikutetaan, niiden väliset yhteydet eivät liiku oikein. Myöskään yhteyspisteen laskeminen ei toimi oikein. `DCGraphics`-luokan metodi `calcConnectionIntersectPoint` on tarkoitettu vain laatikkotyyppisille elementeille, mutta tällä hetkellä sitä käytetään (ilmeisesti?) myös yhteyksille. Luokan `ConnectionGraphics` tulisi korvata metodi omalla toteutuksellaan.

4.1.5 Kooste-elementit

Kooste-elementti on elementti, joka koostuu useasta alielementistä. Vaikka ei ole varsinaisesti toteutettu on tietorakenne suunniteltu ominaisuutta silmälläpitäen ja sen toteuttamisen pitäisi olla suhteellisen suoraviivaista. Kaavio (`Diagram`) muistuttaa huomattavasti kooste-elementtiä, joten siitä voi ottaa mallia. Mahdollisesti kaavio ja kooste-elementti tulisi periä samasta yliluokasta tai niillä olisi sama rajapinta.

4.1.6 Linkit kaaviokomponentista toisiin komponentteihin

Näkymättömät, loogisen tason linkit kaaviokomponentista toiseen komponenttiin (tai komponenttiryhmään/kooste-elementtiin) voi mahdollisesti toteuttaa attribuutti-mekanismin kautta, jolloin varsinaista uutta koodia ei tarvita. Toisena vaihtoehtona voisi olla luokasta `DiagramComponent` tai `Connection` peritty uusi linkkityyppi.

4.1.7 Linkki elementistä tiedostoon

Elementistä voi olla linkki esim. lähdekooditiedostoon. Tätä ei ole varsinaisesti toteuttu missään elementtityypissä, mutta se onnistuu attribuuttimekanismin avulla. Tätä varten on hyvä lisätä uusi attribuuttityyppi, *FILENAME*. Tiedostonimen muokkaaminen attribuuttipaneelissa (esim. `GenericAttributePanel`) voisi avata tiedostonvalintaikkunan.

4.1.8 Projektipuu: alemmat tasot, ikonit

Tietorakenteen tasolla on toteutettu vain puun ylin tason, alemmat tasot eivät käytännössä toimi. Arkkitehtuuri on kuitenkin suunniteltu mielivaltaisille puille, joten ainakin tietorakenteen tasolla tarvitaan vain pienehköjä muutoksia.

Käli: projektipuussa tulisi kunkin kaavion, elementin ja yhteyden vieressä näyttää asiaankuuluva ikoni, joka saadaan metodilla `project.ProjectComponent.getSmallIcon`. Tätä ei ole toteutettu lainkaan.

4.1.9 Kaavion kasvaminen vasemmalle ja ylös

Tällä hetkellä kaavion koordinaatit ovat positiivisia, mikä tarkoittaa sitä, että kaaviota ei voi laajentaa vasemmalle eikä ylös. Ongelman voi korjata kahdella tavalla: a) kaavio sallii negatiiviset koordinaatit, tai b) kaaviossa on vain positiivisia koordinaatteja, mutta kaavion kaikkia elementtejä siirretään tarvittaessa oikealle ja alas, jolloin vaikuttaa siltä, että kaavio on kasvanut vasemmalle ja ylös. Vaihtoehto a) tuottaa ongelmia Swingin kanssa, sillä Swing ei salli negatiivisia koordinaatteja. Tällöin täytyisi pitää kaavion koordinaatit erillään Swingin koordinaateista (saattaa olla muutenkin hyvä idea). Vaihtoehto b) on hieman virhealtis.

4.1.10 Vienti- ja tuontilaajennokset ja niiden rajapinnat

Vienti- ja tuontilajennosten rajapinnoista ainoastaan vienti on määritelty abstraktin rajapinnan avulla. Ohjelmaan tarvittaisiin vähintäänkin laajennokset kaavioiden viemiseksi PNG- ja EPS-muotoon. Myös monille muille vienti- ja tuontityypeille löytyi käyttöä käyttäjähaastattelussa.

EPS-laajennoksen toteuttamiseksi ja oikean vektorigrafiikan tuottamiseksi pitäisi todennäköisesti tehdä joitakin muutoksia kaavioiden piirtometodeihin ja piirtotapaan. Nämä olisi hyvä tehdä varhaisessa vaiheessa, koska silloin myös toteutettuja kaavio-, elementti- ja yhteystyyppejä pitäisi muuttaa.

4.1.11 Tulostus

Kaavioiden tulostaminen olisi käyttäjien kannalta hyvin oleellinen toiminto, mutta sitä ei ole ehditty vielä lainkaan toteuttaa. Tulostusta koskevat samat asiat kuin EPS-laajennoksia.

4.1.12 Yhteys elementistä itseensä

Tietorakenne tukee sellaista yhteyttä, jonka alku- ja loppukomponentti on sama elementti, mutta käyttöliittymässä ominaisuus ei käytännössä toimi. Huom. tietorakenteessa tällaisten yhteyksien yhteysviivan reitin määrittämisen toimivuutta ei ole tarkistettu.

4.2 Parannuksia olemassaoleviin ominaisuuksiin

4.2.1 Elementtien piirto

Graafisesti yksinkertaisille elementtityypeille (laatikko, ellipsi ym.) löytyy luokka `SolidGraphics`, jonka avulla elementtien piirto onnistuu helposti, lähes ilman omaa koodia. Kuitenkin monimutkaisemmat graafiset esitykset joudutaan johtamaan alusta lähtien luokasta `DCGraphics`, mikä on virhealtista.

`SolidGraphics` koostuu vain yhdestä tekstialueesta. Elementtigrafiikkaluokille voisi tehdä monipuolisen yliluokan, joka osaisi käsitellä useista alueista koostuvia elementtejä. Esimerkiksi elementti Actor koostuu loogisella tasolla kahdesta osasta: tikku-ukkografiikasta ja alla olevasta nimikentästä. UML-luokkakaavio (ei toteutettu) koostuu kolmesta alueesta.

4.2.2 Yhteyksien piirto

Yhteysgrafiikoille on melko monipuolinen yliluokka `project.graphics.ConnectionGraphics`, joka osannee piirtää useimmat ”normaalit” yhteydet.

Nuolenpäistä puuttuvat tärkeimpänä UML:n salmiakkinuolenpäät.

Mahdollisia parannuksia:

Tekstikenttä on `TextLayout`-instanssi, joka ei osaa helposti näyttää monirivistä tekstiä eikä tue suoraeditointia. Luokan `DirectTextArea` käyttö korjaisi molemmat viat.

Tekstikenttiä tulisi olla useampia kuin vain yksi: ainakin alku- ja loppupään kentät lisäksi. Ehkä myös erikseen viivan ylä- ja alapuoliset kentät, jolloin kenttiä olisi yhteensä kuusi. Tämä vaatii jonkinlaisen yleistetyn tekstikenttämekanismin; ehkä `HotSpot`-toteutuksesta voi ottaa mallia.

4.2.3 Yhteydet: tyhjät segmentit

Luokalla `ConnectionWrapper` on path-kenttä, joka sisältää yhteysviivan polun taulukkona koordinaattipisteitä. Joissakin tapauksissa yhteydellä voi olla nollan pikselin pituisia segmenttejä, eli kaksi pistettä polku-taulukossa ovat samat. Tämä aiheuttaa ongelmia polun läpikäymisessä, ellei tyhjiin segmentteihin ole varauduttu. Niihin on pyritty varautumaan joka paikassa, mutta parempi ratkaisu olisi estää tyhjien segmenttien syntyminen. Luokan `ConnectionWrapper` metodi `setPath` voisi poistaa tyhjät segmentit automaattisesti.

Toinen segmentteihin liittyvä ongelma on, että muutamat osat ohjelmassa saattavat virheellisesti olettaa tietyn pituisen polkutaulukon, esim. kaksi tai neljä pistettä. Jos näitä oletuksia löytyy, ne on poistettava.

4.2.4 GUI:n sisäinen rakenne

GUI:n sisäinen rakenne on joiltain osin turhan monimutkainen. GUI:lla on sisäinen tila (esim. mikä painike alhaalla), jonka ylläpitäminen on monimutkaista. `DiagramPanel`-luokan hiirenkäsittelijät ovat samaten monimutkaisia. GUI:sta kannattaisi piirtää selventävä tilasiirtymäkaavio.

4.2.5 Projektipuu

Työnjako projektin kaaviopuun suhteen tietorakenteessa on hieman epäselvä. `Project` hallinnoi puun ensimmäisen tason ja `DiagramWrapper` alemmat tasot. Tästä aiheutuu myös koodin duplikointia. Parempi ratkaisu voisi olla se, että `Project`-luokalla on vain puun juurisolmu.

Luokalle `DiagramWrapper` voisi parempi nimi olla `DiagramTreeNode`, koska termiä wrapper käytetään hieman toisessa merkityksessä muualla.

4.2.6 Attribuuttien muokkaus

Uutena attribuuttityyppinä tarvitaan jatkossa taulukkotyyppi, joka koostuu alkeistyypeistä (INTEGER, DOUBLE, jne). Taulukkotyypille tarvitaan tiedot sarakkeiden lukumäärästä sekä kunkin sarakkeen nimestä ja tyypistä. Ehkä taulukkotyyppi olisi `Attribute`-luokan aliluokka? Taulukkotyyppiä voidaan käyttää ainakin UML-luokkaelementissä. Kullakin luokalla on taulukko metodeita ja taulukko kenttiä.

Myös enumeration-tyyppi on tarpeellinen. Esimerkiksi UML-luokkaelementin näkyvyysmääre kuuluu joukkoon {private, default, protected, public}. Tämän voisi koodata merkkijonoina, mutta oma enumeration-tyyppi on elegantimpi.

Käyttöliittymän puolella saattaisi olla tarpeellinen yleinen attribuuttien muokkaamiseen tarkoitettu tekstikenttäkomponentti, jota voisi käyttää sekä attribuuttipaneelissa että piirtoalustalla suoraeditoinnissa. Komponentti olisi samankaltainen kuin `JTextArea` tai `JTextField`. Tekstikenttä tietäisi editoimansa attribuutin tyypin (esim. INTEGER, STRING) ja esim. kokonaislukukenttä estäisi kirjainmerkkien syöttämisen kenttään kokonaan.

Monirivinen teksti aiheuttaa tällä hetkellä ongelmia attribuuttipaneelissa (`GenericAttributePanel`). Paneelissa on yksirivisiä editointikenttiä, mutta tyypin STRING attribuutit saattavat olla monirivisiä. Saattaa olla tarpeen tehdä erilliset tyypit STRING_MULTILINE ja STRING (yksirivinen) ja näille erilliset editorikentät. Toisena vaihtoehtona attribuuttipaneelissa olisi pelkkiä monirivisiä editoreita (JTextArea).

4.2.7 Kaavion attribuuttien muokkaus

Elementtien ja yhteyksien lisäksi myös kaavioilla (luokka `Diagram`) on attribuutteja. Kaavion attribuuttien muokkaamiseen ei tällä hetkellä ole tukea Project Managerissa; `ModifyDComponentAttributeEdit` tukee vain elementtejä ja yhteyksiä. Projektitietorakenne itsessään tukee kaavioiden attribuutteja, samoin käyttöliittymä osaa näyttää kaavion attribuuttipaneelin, kun mitään elementtiä ei ole valittu. Tarvittava muutos Project Manageriin on suoraviivainen.

4.2.8 Ikonien piirto

Tällä hetkellä kullekin kaaviokomponenttityypille on piirrettävä käsin ikoni, joka on työlästä. Ohjelma voisi luoda ikonin käyttämällä kaaviokomponentin omia piirtotyökaluja. Ainakin yhteysnuolet pitäisi onnistua tällä tavalla.

4.2.9 Luokka gui.diagramview.DiagramPanel

Luokan `paintComponent`-metodissa on varauduttu siihen, että Shape-olioilla saattaa olla NaN (Not a Number)-arvoja Bounds2D:ssään. NaN aiheuttaa JVM:n hyytymisen, joten tarkastus on olennainen. Olisi kuitenkin huomattavasti kauniimpaa tarkastaa Shapeja luotaessa, että ne eivät voi olla NaN-arvoisia. Tämä koskee lähinnä luokan `Connection-Graphics` viivaa ympäröivän alueen luomista. (aluetta käytetään yhteyksien valitsemiseen hiirellä piirto- alustalta)

4.2.10 Luokka project.graphics.DirectTextArea

Puuttuva ominaisuus/bugi: kaksoisklikkauksella hiiri menee aina editorikentän loppuun eikä siihen, mihin hiiren kursori osoittaa.

Elementin raahaaminen hiirellä on hidasta, jos hiiri on DirectTextArea:n kohdalla. Tämä johtuu hiiritapahtumien välittämisen hitaudesta metodissa `dispatchEventUp`, jota voi luultavasti optimoida siirtämällä parent-komponentin etsiminen pois k.o metodista. Välittämisen voisi tehdä myös robustimmaksi: mitä jos ei ole DCGraphics-yläkomponentti ei löydy vielä kutsulla `getParent()`, vaan esim. kutsulla `getParent().getParent()`.

4.2.11 Luokka project.graphics.LineHead ja sen aliluokat

Kenttä `preferredHeadLength` on kyseenalainen, koska nuolen voi piirtää mihin kokoon tahansa. Lisäksi nuolenpääluokkaa käyttävä koodi joutuu itse laskemaan nuolen alku- ja loppupisteet. Parempi olisi, jos nuolella olisi ”oikea” length-kenttä, ja nuolenpääluokkaa käytettäisiin antamalla parametrina yhteyssegmentti. LineHead laskisi itse oikean nuolen koon. Tämä yksinkertaistaa myös luokkaa `ConnectionGraphics`.

4.2.12 Luokka project.graphics.SolidGraphics

Paranneltava ominaisuus: suoraeditoinnin tekstinsyötön undo/redo toimii kirjain kerrallaan, mikä on ärsyttävää, jos syötetään paljon tekstiä. Olisi parempi koota yksittäiset kirjainmuutokset yhteen
`ModifyDComponentAttributeEdit`-pakettiin. Tämä vaatii `FocusListener`-rajapinnan hyödyntämistä.

4.2.13 Elementin ja yhteysviivan leikkauspisteen laskeminen

Luokassa `project.graphics.DCGraphics` on yleinen metodi `calcConnectionIntersectPoint` yhteysviivan ja elementin leikkauspisteen laskemiseen. Joskus viiva on pikselin verran elementin sisä- tai ulkopuolella. Tämän voi mahdollisesti korjata tiukentamalla k.o. metodin loppuehtoa.

4.3 Uudet ominaisuudet

4.3.1 Sekvenssikaavion toteuttamisesta

UML-sekvenssikaavio on hieman muista poikkeava kaaviotyyppi aikajanansa vuoksi. Aikajana voisi olla yhteystyyppi, jonka toinen pää on kiinni olio-elementissä (tai luokkaelementissä) ja toinen pää roikkuu irti. Aikajanan muuttaminen x-suunnassa tulisi estää; aikajana menee aina kohtisuoraan alaspäin.

Aikajanan viestien kronologinen järjestys on sekvenssikaaviossa tärkeä. Viestien keskeisen yhteyden voi laskea yhteyskääreiden y-koordinaateista. Analysoinnin helpottamiseksi viesteillä voisi olla attribuutti, joka kertoo sen järjestysnumeron: 1 tarkoittaa aikajanan ensimmäistä viestiä, jne. Tässä ratkaisussa on se ongelma, että attribuutit ovat yhteisiä kaikille kääreille ja saman viesti-yhteyden voi lisätä useaan kaavioon, jolloin tulee useita kääreitä. Pitäisikö kääreillä olla omat attribuuttinsa?

Aikajanan piirtämisessä on otettava huomioon, että osa aikajanasta on paksunnettu (elämänkaari). Tämä vaatii oman `DCGraphics`-luokan.

4.3.2 Yleistetyt yhteydet

Tällä hetkellä yhteyksiä voi olla vain kahden kaaviokomponentin välillä. Yleistetyt yhteydet (n-ary) voivat olla hyödyllisiä.

4.3.3 Muut plugin-rajapinnat

Ohjelmassa on export-rajapinta (esim. EPS), mutta ei import-rajapintaa. Import-rajapinnalle on tarvetta, kun kaavio-ohjelmaa käytetään yhteistyössä muiden CASE-työkalujen kanssa.

Myös analysoija-plugin-rajapinnalle voi olla käyttöä. Analysoija-plugin tekisi mielivaltaista kaavioiden analysointia ja muokkausta.

4.3.4 Sisäinen XML-tiedostomuoto

Kaapo käyttää Javan oliotallennusta sisäisenä tallennusmuotona, missä on se ongelma, että tietorakenneluokkien muuttaminen saattaa rikkoa olemassaolevat kaaviotiedostot. On-

gelman voi ratkaista kirjoittamalla luokille omat `writeObject` ja `readObject`-metodit. Toinen ratkaisu ongelmaan on laatia ohjelmalle sisäinen XML-tiedostomuoto.

4.3.5 Leikepöytä

Kaaviokomponenteille olisi hyvä toteuttaa standardimallinen leikepöytä. Semantiikassa on kaksi vaihtoehtoa: copy-paste voi tehdä uuden kääreen samalle kaaviokomponentille, tai se voi luoda kaaviokomponentista kokonaan uuden kloonin.