En grafeditor för grafteoretiska frågeställningar

Kjell Winblad
E-post: kjellw@cs.umu.se

30 juni 2007
Examensarbete i Datavetenskp, 10 studiepoäng
Handledare på CS-UmU: Jürgen Börstler
Examinator: Per Lindström

Umeå Universitet
Institutionen för Datavetenskap
SE-901 87 UMEÅ
SWEDEN

Sammanfattning

Den här rapporten beskriver skapandet av en grafeditor anpassad för den matematiska grenen grafteori. Syftet med grafeditorn är att göra det snabbt och enkelt att rita grafer, utföra några enkla grafoperationer på grafer och exportera grafer till användbara filformat etc. Grafeditorn kan också användas för att studera grafers struktur genom att på ett intuitiv sätt flytta om dess delar.

Programmet har efterfrågats av lektor Klas Markström vid Umeå Universitet. Design för programmet har skapats i ett samarbete mellan utvecklaren och Klas Markström.

Programmet är utvecklat i programmeringspråket *Java* och är därför plattformsoberoende. Många mjukvarubibliotek med öppen källkod har använts av programmet. Särskilt används *JGraph* som är ett mjukvarubibliotek för grafhantering.

Det existerar mjukvaror som liknar programmet som beskrivs i den här rapporten. De flesta program har dock restriktioner som en icke fri licens vilket gör det svårt att ändra dem och restiktioner på vilken plattform det är möjligt att köra dem på etc.

Programmet går att ladda hem från hemsidan:
http://www.cs.umu.se/∼kjellw/exjobb

A graph editor for graph theoretical problems

This report describes the development of a graph editor for the mathematical field of graph theory. The purpose of the graph editor is to make it fast and easy to draw graphs, perform some simple operations on graphs and export graphs to useful file formats etc. The graph editor can also be used to study the structure of graphs by moving it's parts around in an intuitive way.

The program is requested by senior teacher Klas Markström at Umeå University. The design of the program is created in cooperation with the developer and Klas Markstöm.

The program is developed in the programming language *Java* and thereby platform independent. The program uses a lot of open source software libraries. In particular it uses the *JGraph* which is a software library for graph manipulation.

There exists software similar to the program described in this report. However, most programs have some restriction like a non free license which makes it hard to change them and restriction in which platform it's possible to run them on etc.

In appendix A there is a user's guide in English for the graph editor.

The program can be downloaded from the web page:
http://www.cs.umu.se/∼kjellw/exjobb

Innehåll

Kapitel 1

Inledning

Arbetet som beskrivs i den här rapporten är gjort som 10p examensarbete på C-nivå vid Umeå universitet. Den tid som läggs på arbetet skall motsvara tio veckors arbete.

Anledningen till att det här projektet har valts är att projektets uppdragsgivare Klas Markström har upplevt ett behov av en grafeditor speciellt anpassad för personer som arbetar med grafteori. Klas Markström har tidigare använt *Macintosh* programmet *Cabri-graph* som inte har uppdaterats till *Mac OS X* och inte utvecklats sedan 1998[1]. Ett av målen med projektet är att skapa en ersättare till *Cabri-graph* som fungerar under de flesta plattformar och som kan fortsätta utvecklas.
Den här rapporten är uppdelad i sju kapitel:

Kapitel 1 Det här kapitlet innehåller en inledning.

Kapitel 2 Här beskrivs specifikationen av programmet som togs fram innan projektet startade.

Kapitel 3 Presenterar det undersökningsarbete som genomfördes för att om möjligt hitta program som redan uppfyller specifikationerna och mjukvara som skulle kunna användas i projektet.

Kapitel 4 I det här kapitlet förklaras hur utvecklingsarbetet har gått till med bland annat information om hur programmet har designats. I kapitlet förklaras också varför vissa beslut har tagits. En beskrivning av hur programmets interna design sett ut finns också i det här kapitlet.

Kapitel 5 Beskriver resultatet av arbetet. Här beskrivs vilka av de ursprungliga målen som är uppfyllda och vilka som inte är det samt vilka möjliga vidareutvecklingar som finns.

Kapitel 6 Här beskrivs vilka slutsatser som utvecklaren har dragit under arbetets gång.

Kapitel 7 Tacksägelser till de som har bidragit mycket till arbetet.

[1] http://www-cabri.imag.fr/CabriGraphes/cabri_anglais/gb_cabri_graph.html

Kapitel 2

Problembeskrivning

2.1 Projektets mål

Projektets mål är att skapa en grafeditor som är specialiserad för den matematiska grenen grafteori. Programmet skall vara körbart på de flesta plattformar och framförallt på *Mac OS X* då det är den plattform som uppdragsgivaren använder. Uppdragsgivaren har tidigare använd programmet *Cabri-graph*, vilket inte längre stöds under *Mac OS X*. Eftersom *Cabri-graph* enligt uppdragsgivaren i stort sett har uppfyllt sin funktion, är ett av målen att skapa ett program som har en liknande funktionalitet som *Cabri-graph*, men saknar många av de brister som *Cabri-graph* har. Programmet skall t.ex. fungera under de operativsystem som är vanliga för skrivbordsdatorer som t.ex. *Microsoft Windows*, *Mac OS* och *Linux*. Ett annat mål är att programmet skall vara gjort på ett sådant sätt att det är lätt att vidareutveckla och lägga till ytterligare funktioner.

2.2 Funktionella krav

I den här sektionen presenteras de funktionella krav som togs fram för programmet. Kraven har tagits fram i samråd med uppdragsgivaren. Programmet skall minst ha följande funktionalitet:

F1 Programmet skall ha ett grafiskt användargränssnitt.

F2 Det skall finnas en yta i programmet där grafer skall kunna ses och ändras. På ytan skall hörn och kanter kunna vara utplacerade. Hörnen representeras av någon figur och kanterna är streck som går mellan hörnen.

F3 Det skall gå att lägga till hörn i grafen genom att placera ut dem med hjälp av musen. Genom att trycka på en position på grafytan med musen skall man kunna placera ut nya hörn i grafen.

F4 Det skall gå att flytta hörn och på så vis ändra utseendet på grafen. Man skall alltså genom att använda musen kunna flytta hörn. Kanterna som är sammanbundna med hörnet skall då flyttas med.

F5 Det skall gå att spara och läsa upp grafer så att grafens utseende bevars. Det skall finnas ett filformat som programmet klarar av att spara ner till och öppna upp från, på ett sådant sätt att grafens utseende och egenskaper bevaras.

F6 Det skall gå att exportera graferna till ett antal externa format. Ett bildformat som t.ex. *postscript* samt ett format som skall gå att importeras av programmet *Mathematica*. Exempel på format som går att öppna upp och exportera till i *Mathematica* är format som liknar grannlistor och grannmatriser.

F7 Det skall vara möjligt att importera grafer från ett filformat som *Mathematica* kan exportera till.

F8 Det skall gå att markera en del av grafen för att utföra operationer på den delen. Som t.ex. att ta bort eller kopiera.

F9 Det skall vara möjligt att ångra och göra om operationer man har gjort på grafen.

F10 Det skall finnas möjlighet att kopiera, klippa ut och klistra in delar av en graf.

F11 Det skall gå att på ett enkelt sätt slå ihop två hörn genom att t.ex. dra dem samman. De kanter som binder dem skall då försvinna och de andra kanter som gick ut från dem skall gå ut fån det nya hörnet.

Ovanstående funktionalitet är de krav som uppdragsgivaren har på programmet. Det finns även en del funktionalitet som är önskvärd men som inte är ett krav för att programmet skall vara användbart. För att senare kunna utökas med ytterligare funktionalitet är det viktigt att programmet är skrivet på ett sådant sett att det är enkelt att utöka funktionaliteten ytterligare. Följande funktionalitet är exempel på önskvärd funktionalitet:

WF1 En möjlighet att visa och dölja samt ändra märkning av kanter och hörn. En märkning är någon form av text som användaren kan ändra.

WF2 Att på ett enkelt sätt kunna utföra grafoperationer som t.ex. multiplikation av två grafer.

WF3 Funktionalitet som gör det möjlighet att sätta olika färger på hörn.

WF4 Att på olika sätt organisera om grafen för att bli mer lättläsliga.

WF5 Att kunna lägga till funktionalitet dynamiskt genom ett specificerat gränssnitt.

WF6 Att det skall finnas ett gränssnitt för att kunna visualisera algoritmer som utförs på grafer.

WF7 Att det skall finnas möjlighet att skapa riktade grafer.

WF8 Att det skall finnas möjlighet att skapa grafer där kanter och hörn har vikter.

2.3 Icke funktionella krav

Nedan beskrivs de ickefunktionella krav på programmet som har tagits fram och diskuterats med uppdragsgivaren:

NF1 Det är viktigt att programmet lätt kan utökas och modifieras. Det är alltså viktigt att programmet är konstruerat på ett sådant sätt att det blir lätt att lägga till nya funktioner. Det måste även vara väldokumenterat så att det är lätt att sätta sig in i programmets uppbyggnad för att sedan kunna lägga till funktioner.

NF2 Programmet skall vara lättförståeligt och lättanvänt så att användare lätt och snabbt skall kunna sätta sig in i det.

NF3 Programmet skall vara portabelt så att det kan användas på många plattformar. Detta är viktigt eftersom det inte går att veta vilken plattform alla potentiella användare använder. Programmet skall minst kunna köras på operativsystemen *Microsoft Windows*, *Mac OS* och *Linux*.

Kapitel 3

En översiktlig marknadsundersökning

Innan utvecklingen av programmet startade gjordes en snabb marknadsundersökning för att ta reda på om det finns några grafeditorer med ungefär samma egenskaper som har specificerats för den efterfrågade grafeditorn. Undersökningen genomfördes genom sökning på *Internet*. Under sökningen hittades många program för att rita grafer av olika typer. De flesta program som hittades var inte specifikt gjorda för att användas till grafteori. Ett exempel på en annan typ av grafeditorer kan vara editorer för flödesscheman.

Under marknadsundersökningen hittades också ett antal mjukvarubibliotek för att hantera grafer både som datastrukturer och visuellt. Ett mjukvarubibliotek som är väldigt intressant för just det här projektet är *JGraph*. *JGraph* är ett mjukvarubibliotek som tillhandahåller en komponent för att utveckla grafeditor med. *JGraph* är gjort i programmeringsspråket *Java* och är baserat på *GUI*[1]-biblioteket *Swing*[2]. Ett annat intressant mjukvarubibliotek för grafeditorn som hittades under den inledande marknadsundersökningen är *JGraphT*. *JGraphT* är också det skrivet i *Java* för att tillhandahålla datarepresentation av grafer med utgångspunkt från grafteori.

Under underökningen hittades tre program som är gjorda i ungefär samma syfte som det här projektet. Dessa program beskrivs i listan nedan:

JGraphEd [3] är en grafeditor skriven i *Java*. Programmet är skriven som ett examensarbete vid *School Of Computer Science* vid *Carleton University, Ottawa, Canada* av Jon Harris. Den har till stora delar all den funktionalitet som har specificerats till det här projektet. Det den saknar är framför allt möjlighet att exportera grafer till externa format. Programmet har öppen källkod. Programmet är ett bra alternativ till grafeditorn som utvecklas i det här projektet. Anledningen till att inte basera det här projektet på *JGraphEd* är att *JGraph* verkar vara ett mer genomtänkt bibliotek för visualisering och editering av grafer. Därför skulle ett projekt baserat på *JGraph* vara enklare att utveckla vidare även efter projektets slutförande. *JGraphEd* användargränssnitt kan upplevas som krångligt och ohanterligt för ovana användare, mycket på grund av att det finns en mängd snabbknappar

[1] GUI är en förkortning för *Graphical User Interface* eller grafiskt användargränssnitt.
[2] Swing följer med företaget *Sun Microsystems* standard version av *Java*s programbibliotek.
[3] http://www.jharris.ca/JGraphEd

synliga i huvudfönstret.

Grin (GRaph INterface) [4] är en grafeditor utvecklad för *Microsoft Windows*. Programmet har samma syfte som det här projektet. Programmet har utvecklats av Ph.D Vitaly Pechenkin. Det är dock en stängd programvara och går endast att använda på operativsystemet *Microsoft Windows* så det kan inte ersätta det här projektet.

Cabri-graph [5] är den grafeditor som uppdragsgivaren har erfarenhet att arbeta med. Den har också samma funktionalitet som planeras att utvecklas i det här projektet. Det har dock slutat utvecklas. Är utvecklat i programmeringspråket *C* för *Mac OS* men fungerar endast till *Mac OS 9* och tidigare och underhålls inte längre.

Under undersökningen hittades också ett antal programbibliotek och program som används i samband med grafer och som kan vara av intresse för det här projektet. Nedan listas de som anses viktigast:

JGraph [6] En *Java Swing*-komponent som kan användas för skapa grafprogram för att visualisera och editera grafer. Den är gjord för att vara väldigt anpassningsbart till olika tillämpningar. Licenserad som öppen källkod. Det finns en mängd kommersiella och öppna program som baseras på *JGraph*. Det finns ett kommersiellt tilläggsbibliotek till *JGraph* som heter *JGraph Loyout Pro* och som kan användas för att göra automatiska inbäddningar av grafer.

JGraph Pad CE [7] En generell grafeditor med öppen källkod. Den är ett bra exempel på hur man kan använda *JGraph* för att utveckla grafeditorer och liknande program. Det finns ett antal automatiska inbäddningsalgoritmer implementerade till programmet. Ett alternativ skulle ha kunnat vara att basera programmet på *JGraph Pad CE*. Anledningen till att det inte gjordes var att det verkade mer tidskrävande att sätta sig in i *JGraph Pad CE* och göra om programmet för att passa till kraven än att utveckla programmet från början.

JGraphT [8] Ett mjukvarubibliotek som är baserat på grafteori och som kan användas för att representera grafer. Det finns en koppling mellan *JGraphT*-biblioteket och *JGraph* som gör att man kan använda *JGraphT* som representation för den graf som visas med hjälp av *JGraph*.

aiSee [9] Ett annat mjukvarubibliotek som inte är öppen källkod. En mängd grafrelaterade program är även baserade på detta.

yEd [10] En grafeditor med ett intuitivt användargränssnitt som är gjord i *Java*. Den har inte öppen källkod.

En slutsats man kan dra efter marknadsundersökningen är att det finns en mängd mjukvara för att visualisera och editera grafer. Det finns även ett antal program som är specifikt anpassade för grafteori. Det har dock inte gått att hitta ett program som uppfyller alla de krav som har ställts upp i kapitel 2. Därför finns ett syfte i att utveckla ett nytt program för att uppfylla kraven.

[4] http://www.geocities.com/pechv_ru
[5] http://www-cabri.imag.fr/CabriGraphes/cabri_anglais/gb_cabri_graph.html
[6] http://www.jgraph.com
[7] http://sourceforge.net/projects/jgraph
[8] http://www.jgrapht.org
[9] http://www.aisee.com
[10] http://www.yworks.com

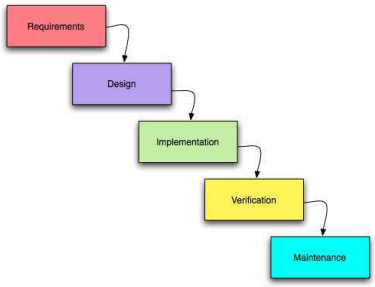
Kapitel 4

Utförande

I det här kapitlet beskrivs hur utvecklingsarbetet har genomförts. Kapitlet har i möjligaste mån strukturerats på ett sådant sätt att det följer den kronologiska ordning saker har genomförts i. Men vissa utvecklingssteg har utförts parallellt utan en bestämd ordning så det är inte hela sanningen. Till exempel så har mycket av den objektorienterade designen av programmet bestämts under själva implementationen.

4.1 Planering

Innan projektet startade på allvar gjordes en planering över hur arbetet skulle genomföras. Planeringen baserades på utvecklarens tidigare erfarenhet av programvaruutveckling. Planeringen liknar den s.k. *vattenfalls-modellen*. Programmet har dock inte utvecklats strikt efter någon utvecklingsmodell. Till exempel så har utvecklingen flera gånger gått tillbaka till tidigare utvecklingsteg för att utöka funktionaliteten eller rätta till fel, så kallad iterativ utveckling.



Figur 4.1: Sketch över *vattenfalls*-modellen. (Bilden är kopierad med tillåtelse enligt *Creative Commons Attribution ShareAlike License v. 2.5 (http://creativecommons.org/licenses/by-sa/2.5/)* från *http://en.wikipedia.org/wiki/Waterfall_model (2007-05-26))*

Se tabell A.1 i bilaga A för den ursprungliga tidsplaneringen som gjordes i början av projektet.

4.2 Design av det grafiska gränssnittet

När kravspecifikationen som beskrivs i kaptel 2 var klar fortsatte arbetet med en design av ett grafiskt gränssnitt som utformades för att möta de specificerade kraven. Designen skapades i form av sketcher på papper med penna. Syftet med sketcherna var att visa ungefär hur programmet skulle se ut och fungera och inte att vara en exakt prototyp för programmet. Eftersom sketcher med papper och penna är enkla att göra är det också lätt att ändra och korrigera designen. Sketcherna kompletterades med beskrivande texter för att kunna beskriva programmets beteende tydligt och kunna användas som mall för utvecklingsarbetet sedan. När designen var klar presenterades den för uppdragsgivaren för att på så vis verifiera att den uppfyller den funktionalitet som önskas. Se bild 4.2 för en jämförande bild mellan en av sketcherna och programmets gränssnitt.

Det kan vara mycket enklare att lära sig ett användargränssnitt som liknar något man redan känner till än ett helt nytt. Eftersom uppdragsgivaren tidigare har arbetat med programmet *Cabri-graph* analyserades dess användargränssnitt innan arbetet med designen startade för att kunna efterlikna lyckade koncept. Utvecklaren tittade också snabbt på en del andra grafeditorer, men *Cabri-graph* var det som studerades mest noggrant. De flesta av de grafeditorer som nämns i kapitel 3 testades också innan den grafiska designen konstruerades.

4.3 Val av programmeringsspråk

Programmet har helt utvecklats i programmeringsspråket *Java* p.g.a. ett antal orsaker som listas nedan:

- Krav **NF3** i problemspecifikationen säger att programmet skall gå att använda med de vanliga operativsystemen för skrivbordsdatorer. Detta blir möjligt om programmet utvecklas i *Java* eftersom *Java* är anpassat för att göra det lätt att utveckla program som går att köra på flera operativsystem. *Java*-program körs över en virtuell maskin vilket gör att man i de flesta fall inte behöver göra några förändringar över huvud taget för att få programmet körbart på många plattformar.

- Det finns ett klassbibliotek för att skapa grafiska gränssnitt inkluderat i *Java*s standardbibliotek som gör det lätt att utveckla ett grafiskt gränssnitt som går att köra på flera plattformar.

- *Java* är det programmeringspråk som utvecklaren har mest erfarenhet att arbeta med.

4.4 Integration med tredjeparts mjukvarubibliotek

Programmet använder ett antal tredjepartsmjukvarubibliotek. Alla dessa är licensierade under öppna källkod-licenser. Att den är distribuerad under öppen källkod betyder att man har fri tillgång till källkoden om man vid en eventuell distribution av ett program som baseras på koden också distribuerar den förändrade koden under samma licens. Det existerar en mängd olika typer av licenser för öppen källkod. Ett försök till en definition

av vad som är en öppen källkod licens har gjorts av organisationen *open source initiative*[1]. Att alla tredjeparts mjukvarubibliotek har öppen källkod är viktigt eftersom det icke funktionella kravet **NF1** säger att programmet skall vara enkelt att vidareutveckla. Skulle programmet använda bibliotek med licenser som inte gör källkoden tillgänglig eller tvinga användare att betala för att använda dem, skulle det vara ett hinder för vidareutveckling av programmet. En lista över vilka tredjepartsbibliotek som används i projektet finns nedan. Listan innehåller också information om vilka licenser biblioteken använder. Den främsta skillnaden mellan licenserna är att de är olika restriktiva vad det gäller om det är tillåtet att använda biblioteken med mjukvara som är licensierade under en annan typ av licens. För mer detaljerad information om varje licens hänvisas till hemsidorna som refereras i fotnoterna.

JGraph En grafisk komponent för att visualisera och ändra grafer. *JGraph* är licensierat under licensen LGPL[2].

JGraphT Innehåller Datastrukturer som baseras på grafteori för att representera grafer. Även *JGraphT* är licensierat under licensen LGPL.

JGraphpad CE En generell grafeditor som t.ex. kan användas för att rita flödesscheman. Det här programmet baseras på *JGraph*. Det som används från *JGraphpad CE* är koden för att göra automatisk inbäddning av grafer. Det finns även en variant med icke öppen källkod av programmet som heter *JGraphpad PRO*. *JGraphpad CE* är licensierat under licensen GPL[3].

XStream Ett bibliotek som används för att spara ner grafer och inställningar till filformatet *XML*. Med hjälp av *XStream* kan man generera text i *XML*-format från objekt i *Java* samt ”ladda” objekt från *XML*-filer. Det är licensierat under en modifierad BSD-licens[4].

EPS Graphics Library Ett bibliotek som har en klass som ärver av klassen *Graphics* i *Java*s standardbibliotek. Med hjälp av objekt av den klassen kan man utföra grafikoperationer som resulterar i en EPS (Encapsulated PostScript)-fil. Biblioteket används för funktionen för export till EPS-filer i programmet. Det är licensierat under GPL-licensen.

JavaHelp System Ett hjälpsystem som gör det enkelt att skapa grafiska hjälpdialoger till *Java*-program. Licensierat under GPL med ett undantag som kallas *class path*-undantaget.

BrowserLauncher2 Används för att underlätta öppnandet av externa webbläsare på ett plattformsoberoende sätt. Det är licensierat under licensen LGPL.

Det mjukvarubibliotek som har utnyttjats mest i utvecklingarbetet är *JGraph*. Ytan där grafer visas och ändras baseras på *JGraph*. *JGraph* är gjord för att för att vara en anpassningsbar grafkomponent för ändring och visualisering av många typer av grafer. *UML*-verktyg, editor för flödesscheman och kopplingsscheman är exempel på programtyper som använder *JGraph*. *JGraph* använder sig av designmönstret *Model View Controller*. Designmönstret *Model View Controller* går ut på att dela upp visuella interaktiva komponenter i de tre delarna:

[1] http://www.opensource.org
[2] http://www.gnu.org/licenses/lgpl.html
[3] http://www.gnu.org/copyleft/gpl.html
[4] http://xstream.codehaus.org/license.html

Model eller modellen är en datastruktur som representerar de data som komponenten skall visualisera,

View eller vy är den visuella delen som presenterar informationen för användaren och

Controller eller kontrollen är den del som tar hand om interaktionen med användaren.

Denna modulariserade uppbyggnad gör det enkelt att byta ut och ändra delar för att ändra funktionaliteten av komponenten. Detta har utnyttjats flitigt under utvecklandet av grafeditorn. Modell delen har bytts ut till modellen som finns i *JGraphT*-biblioteket. Detta gör att grafrepresentationerna som finns i *JGraphT* kan användas som datarepresentation för grafen. Detta är en fördel jämfört med baskomponenten som finns i *JGraph* eftersom *JGraphT* har utvecklats med grafteori i åtanke. Till exempel så finns det i *JGraphT* ett antal algoritmer för att traversera grafer, vilket kan vara användbart vid implementerade av funktionalitet för att utföra operationer på grafer. Även kontrolldelen och vydelen i *JGraph* har ändrats för att passa designen av det här programmet. Mer om hur det har gått till kommer i nästa stycke om den objektorienterade designen.

4.5 Konstruktion av objektorienterad design

Den objektorienterade designen skapades med utgångspunkt från kravspecifikationen och design sketchen över det grafiska gränssnittet. Under marknadsundersökningen som beskrivs i kapitel 3 hittades mjukvarubiblioteken *JGraph* och *JGraphT*. Dessa två bibliotek har spelat en stor roll vid den objektorienterade designen eftersom de utgör en avgörande del av programmet. För att kunna få en bild över hur programmet skulle kunna sättas samman med dessa komponenter är det en nödvändighet att sätta sig in i hur de fungerar och är organiserade. Till *JGraph* finns en utförlig manual med information om hur det kan anpassas för att passa olika tillämpningar[5]. Till *JGraphT* är man hänvisad till dess API-dokumentation (Application programming interface).

I figur 4.3 visas en *UML*-liknande (Unified Modeling Language) sketch över de mest grundläggande klasserna i den objektorienterade designen. Den mest grundläggande designen som beskrivs i figuren skapades innan implementationsarbetet startade. Detta är dock inte en särskilt detaljerad design. Den specificerar till exempel ingenting om vilka metoder som skall finnas med eller vad klasserna har för uppgifter mer än vad deras namn säger. Anledningen till att designen inte specificerades noggrannare än så är att *JGraph* var så pass ansträngande att sätta sig in i, så att man i vissa fall behövde testa delar av funktionaliteten för att vara säker på hur det verkligen fungerade. Det fanns två alternativ för designen. Dessa alternativ förklaras nedan med de risker och fördelar de bedömdes ha:

Alternativ 1 var att läsa på och testa *JGraph* för att få en mycket detaljerad insikt i hur det fungerar. Detta för att sedan kunna göra en detaljerad design. Det bedömdes att en risk med det här alternativet var att det skulle ta så lång tid att göra den detaljerade designen så att det inte skulle gå att hinna med implementationen. En uppenbar fördel med det här alternativet är dock att programmet troligtvis skulle få en genomtänkt design vilket skulle vara positivt för det icke funktionella kravet **NF1**.

[5] http://www.jgraph.com/pub/jgraphmanual.pdf

Alternativ 2 var att göra en snabb översiktlig objektorienterad design och börja implementationen snabbt och läsa på i dokumentationen för varje implementerad funktionalitet. Detta kan vara effektivt, eftersom det går att vara påläst om det man för tillfället håller på med under utvecklingen. Man skulle dessutom komma igång med implementationen snabbare vilket gör det lättare att bedöma hur lång tid den delen skulle ta. Risken är naturligtvis att programmet inte blir lika övertänkt som i alternativ 1 och att det i sin tur skulle leda till ineffektivt arbete och göra programmet svårare att vidareutveckla.

Alternativ 2 valdes slutgiltligen eftersom det bedömdes att designen skulle bli såpass genomtänkt att implementationen skulle bli tillräckligt effektiv. Men det var inte ett självklart val eftersom det var svårt att bedöma hur lång tid en så noggrann design skulle ta att genomföra. Alternativ 1 hade troligtvis varit ett bättre val om projektet hade haft många utvecklare. Detta eftersom en detaljerad design gör samarbetet med implementationen betydligt enklare. Är det många personer som implementerar ett program utan en detaljerad design kan det vara svårt att arbeta parallellt med olika delar av programmet eftersom man inte vet i detalj hur delarna skall passa ihop.

4.6 Implementationsarbete

Implementationensarbetet utfördes i två steg. Först implementerades alla funktionella krav som betraktades som grundläggande krav vid kravspecifikationen. Sedan utvärderades de i samråd med uppdragsgivaren. När uppdragsgivaren verifierat att de grundläggande kraven var implementerade genomfördes en ny utvärdering av vilka extra finesser som var önskvärda. Dessa sammanställdes i en ny lista med prioritetsordning. Prioritetsordningen är skapad utifrån hur viktig funktionaliteten anses vara samt hur tidskrävande implementationen av den bedöms vara. Detta eftersom tiden för arbetet är begränsad och det är omöjligt att bedöma hur många av dessa funktionaliteter som skulle hinna implementeras. Listan skiljer sig en hel del från den lista som gjordes över exempel på önskad funktionalitet som finns i problemspecifikationen. Den nya listan presenteras nedan:

EF1 Ändra form på hörn. Så att det går att välja vilken form hörn som placeras ut får samt ändra formen på utplacerade hörn.

EF2 Ändra färg på hörn på samma sätt som det går att ändra form.

EF3 Att kunna generera en komplett bipartit $K_{m,n}$ graf utifrån två delgrafer. Man markerar först en delgraf sedan den andra och därefter genereras de kanter för att grafen skall bli en $K_{m,n}$ graf.

EF4 Implementera de två vanligaste grafprodukterna *Cartesian* och *Categorial* produkterna. Man markerar först en delgraf sedan den andra. Den resulterade produkten placeras på ett tomt ställe i graffönstret.

EF5 Göra en delgraf komplett.

EF6 Markera nåbara delar av grafen från en markering. Alla hörn som är grannar med de markerade hörnen blir markerade.

EF7 Ge statistik angående grafen. Hur många hörn och kanter varje delgraf har.

EF8 Lägga markerade hörn i en cirkel.

EF9 Rotera och spegla valda markerade delar av grafen. Speglingen skall kunna göras både horisentalt och vertikalt.

EF10 Exportera grafen till formatet EPS (Encapsulated PostScript).

EF11 Exportera till ett rent grannlistformat, det vill säga en fil där varje rad representerar ett hörn och där det första numret är hörnets identifikation och resterande nummer är hörnets grannar. Numren separeras med ett mellanslagstecken.

EF12 Implementera fjäderinbäddning[6], som automatiskt organiserar grafen.

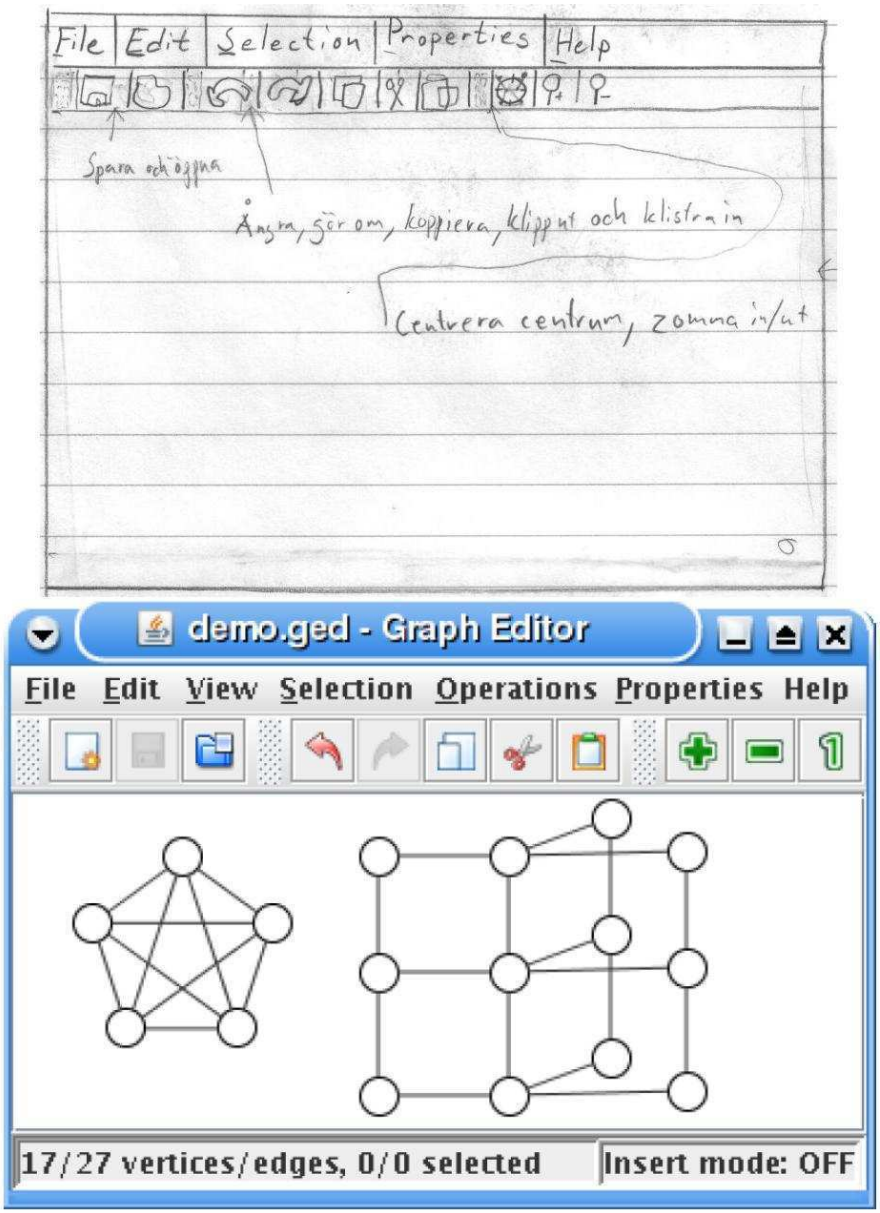
EF13 Sätta etiketer på grafer beroende på valt format. Exempel på format är att etiketten innehåller hörnens valens, att de är numrerade så att de med störst valens har lägst nummer, att etiketterna innehåller avstånd från ett visst hörn eller den traverseringsordning de har vid *bredden-först-sökning* och *djupet-först-sökning*.

EF14 Generera grafer utifrån *random key regular graph*-algoritmen.

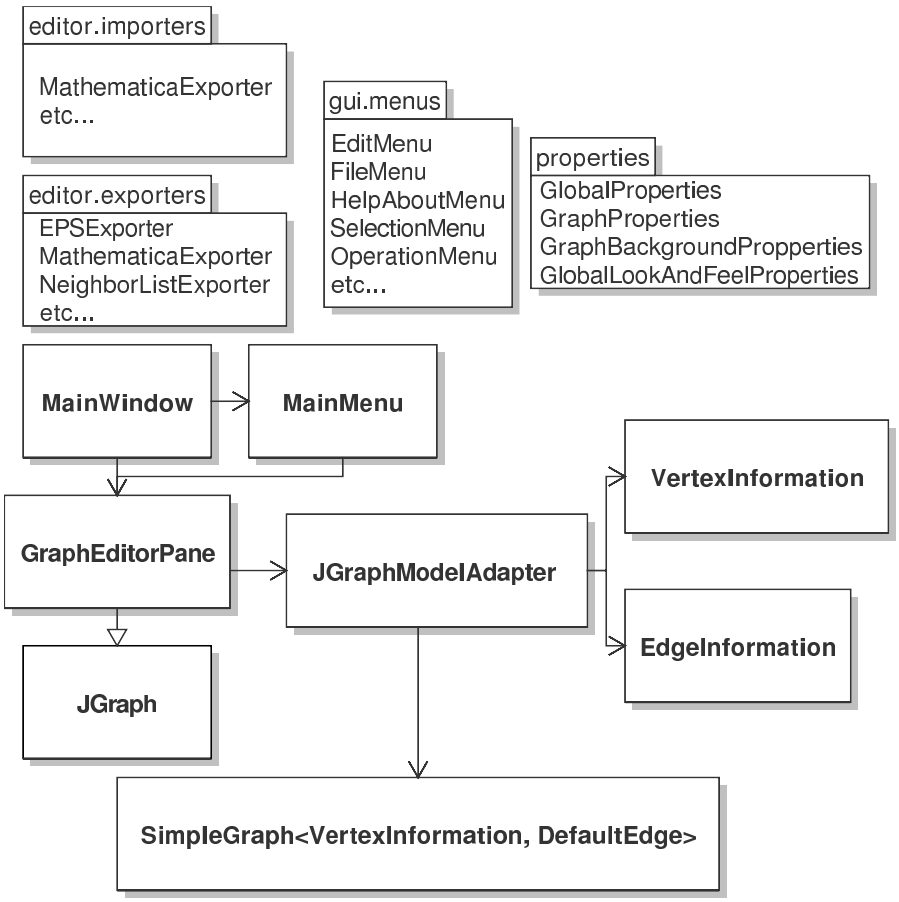
EF15 Importera och exportera grafer i Graph6 formatet.

På grund av tidsbegränsningen har inte riktigt alla extrafunktionaliteter implementerats, utan **EF13** till **EF15** har inte implementerats. Resterande funktionalitet skulle kunna vara nästa steg i en eventuell vidareutveckling av programmet.

[6] På engelska *spring-embedding*



Figur 4.2: Ett exempel på en designsketch som går att jämföra med en färdig version av programmet.



Figur 4.3: En sketch som visar de mest grundläggande klasserna i den objektorienterade designen. Klassen GraphEditorPane ärver av klassen JGraph som är huvudkomponenten i JGraph-biblioteket. GraphEditorPane är associerad till JGraphT komponenten JGraphModelAdapter som gör att man kan använda JGraphT grafer som modell för JGraph komponenten. JGraphModelAdapter är konfigurerad för att använda klasserna VertexInformation och EdgeInformation för att representera information om ett hörn samt en kant. MainWindow som representerar programfönstret har en referens till en GraphEditorPane och en MainMenu. Ovanför klassdiagrammet visas symboler för ett antal exempel på paket med exempel på klasser. Bilden är bara till för att visa den viktigaste strukturen i designen och inte för att ge en heltäckande översikt. Många klasser och paket har utelämnats för att göra huvuddragen i designen tydligare. För en utförligare dokumentation av den objektorienterade designen hänvisas till api-dokumentationen.

Kapitel 5

Resultat

Eftersom det här projektet är att skapa ett program är resultatet programmet. Det här kapitlet beskriver programmets olika delar samt ger en sammanfattning av dess funktioner.

5.1 Systemöversikt

Programmets huvudkomponent är ritytan för grafer. Alla andra delar av programmet är gjorda för att på ett eller annat sätt manipulera eller utföra någonting på ritytan. För att denna komponent inte skall bli för ohanterlig och komplex har mycket av funktionalteten flyttats utanför denna komponent. Till exempel så är funktionalitet för att exportera grafer till andra filformat utflyttade. Funktionalitet som har upplevts mer naturlig att ha integrerad i komponenten har integrerats med den. Exempel på sådan funktionalitet är de operationer som går att utföra på grafen.

Programmets klasser har delats in i så kallade paket för att göra det lättare att hitta klasser som hänger samman med en viss funktionalitet. Dessa paket listas nedan med en kort beskrivning av vilka klasser som finns i dem:

org.grapheditor.editor I det här paketet och alla dess underpaket finns klasser som har med själva grafeditorkomponenten att göra. I paketet finns klassen *GraphEditorPane* som ärver av klassen *JGraph* och representerar ytan där grafer ritas. Mång av klasserna i det här paketet ärver av klasser i *JGraph*-biblioteket för att omdefiniera deras funktion.

org.grapheditor.editor.graph Här finns klasser för att representera själva grafen. *VertexInformation* och *EdgeInformation* används för att representera information om hörn och kanter. Det finns också klasser för att spara ner och öppna grafer från filer.

org.grapheditor.editor.exporters Här finns klasser som används för att exportera grafer till externa filformat. Koden för utförandet av exporten har delats upp från koden för det grafiska gränssnittet. Detta för att på ett enkelt sätt kunna byta ut den.

org.grapheditor.editor.importers Innehåller klasser som har att göra med import av grafer från externa filformat.

org.grapheditor.editor.listeners I det här paketet finns gränssnitt för lyssnare. En lyssnare är ett objekt som kan registreras som lyssnare för någon speciell händelse och har en eller flera metoder som kan anropas när dessa händelser inträffar. De lyssnare som finns i det här paketet är de lyssnare som har skapats för att från programmets övriga delar kunna ”känna av” att händelser som t.ex. att grafen har ändrats.

org.grapheditor.editor.menus Innehåller menyer som används av *GraphEditorPane* komponenten. Menyerna som visas när man högerklickar någonstans i graffönstret. Beroende på vilka hörn och kanter som är valda visas olika menyer.

org.grapheditor.editor.vertexrenders I det här paketet finns klasser som ärver av klassen *VertexRenderer* som finns i *JGraph*-biblioteket och är ett gränssnitt för att få en komponent för att *rendera* ett hörn.

org.grapheditor.gui Det här paketet innehåller klasser som har med det grafiska gränssnittet att göra. Här finns t.ex. klassen som representerar programmets huvudfönster.

org.grapheditor.gui.menus Innehåller menyerna som finns i huvudfönstret. Till exempel menyerna *File, Edit* och *Selection* etc. Några av dessa menyer använder sig av menyerna som finns i paketet **org.grapheditor.editor.menus**.

org.grapheditor.gui.toolbars Här finns klasser för programmets verktygsfält med snabbknappar.

org.grapheditor.properties I det här paketet finns klasser som används för att hantera programmets inställningar. Klassen *GlobalProperties* använder sig av designmönstret *singelton* och representerar de globala inställningar som gäller för hela programmet. *GraphProperties* är motsvarande fil för inställningar som hör till en enskild graf som t.ex. grafens bakgrundsfärg. Det finns även klasser som representerar grafiska komponenter för att ändra inställningar. Dessa implementerar gränssnittet *PropertiesModule* för att man på ett enkelt sätt skall kunna lägga in dem i inställningsdialoger.

För en mer detaljerad förteckning av vad de olika paketen innehåller hänvisas till api-dokumentationen som finns på projektets hemsida[1].

5.2 Implementerad funktionalitet

Alla de grundläggande funktionalitetskrav som ställdes i kapitlet 2 med problembeskrivning har implementerats. Punkterna **WF1** till **WF7** som ställdes upp som exempel på önskvärd funktionalitet i problemspecifikationen användes aldrig under utvecklingsarbetet. I stället togs en ny och mer detaljerad lista med extrafunktionalitet fram. Denna lista finns i kapitel 4 under rubriken implementationsarbete. Av den funktionalitet som beskrivs där är **EF1** till **EF12** implementerade. Extra funktionalitet **EF12** som var att implementera fjäderinbäddning blev betydligt enklare än vad som hade väntats eftersom källkod från projektet *JGraph Pad CE* kunde användas för det.

För mer detaljerad information om hur den funktionalitet som finns i programmet används hänvisas till användarhandledningen. Användarhandledningen är skriven på

[1] http://www.cs.umu.se/ kjellw/exjobb

engelska och finns med som bilaga B till denna rapport. Användarhandledningen finns också integrerad i programmet som ett dialogfönster som öppnas under hjälpmenyn.

Kapitel 6

Slutsatser

I det här kapitlet diskuteras projektets resultat samt vilka framtida utvecklingsmöjligheter som finns.

6.1 Har projektets mål uppfyllts?

Den kravspecifkation och projektbeskrivning som gjordes i början av arbetet med grafeditorn måste ses som projektets målbeskrivning. Som tidigare har nämnts i kapitel 5 har många av projektets mål uppfyllts genom att alla de grundläggande kraven och mycket av den efterfrågade extrafunktionaliteten har implementerats.

De icke funktionalla målen **NF1** till **NF3** som är beskrivna i problembeskrivningen är betydligt svårare att bedöma om de är uppfyllda. Detta eftersom i alla fall **NF1** och **NF2** är beroende av subjektiv uppfattning. **NF3** att programmet skall vara användbart på de flesta vanliga plattformar borde man dock kunna säga att programmet uppfyller eftersom det har testats på *Microsoft Windows*, *Mac OS* och *Unix*-baserade system. Dessa plattformar måste utan tvekan anses som de vanligaste. Funktionalitet **NF1** som är att programmet skall vara välstrukturerat och dokumenterat på ett sådant sätt att det är lätt att bygga ut med ytterligare funktionalitet är dock omöjligt att svara på objektivt. Det är dock inte säkert att kravet var betydelselöst, eftersom att ha det i åtanke kan ha lett till att strävan har riktats åt det hållet mer än om inte målet skulle vara uppskrivet. Samma sak som för **NF1** gäller även **NF2** som är att programmet skall vara användarvänligt. Man skulle kunna få en uppfattning om det genom att testa programmet med ett antal användare och undersöka hur deras upplevelse av programmet har varit. Projektets tidsbegränsning har dock inte tillåtit något sådant att ske.

6.2 Framtida utvecklingsmöjligheter

Förutom den extrafunktionalitet som har nämnts går det att vidarutveckla programmet på många andra sett. Det går till exempel att finna förfiningar och förbättringar i nuvarande funktionalitet. Till exempel kan man genom att utvärdera användningen av programmet hitta saker i användargränssnittet som går att göra på ett mer intuitivt sätt. Det finns även en mängd funktionaliteter som t.ex. grafprodukter som skulle vara intressanta att ha med i programmet för ett visst användningsområde inom grafteorin.

Ett av målen med projektet var att skapa ett program med ungefär samma funktionalitet som programmet *Cabri-graph*. Det är fortfarande kvar funktionalitet som finns i *Cabri-graph*, men som inte är implementerad. En möjlig väg för att hitta vidareutvecklingsmöjligheter är att titta mer på den funktionalitet som finns i *Cabri-graph*. Det skulle kunna vara en smart väg att gå eftersom *Cabri-graph* har utvecklats och används under lång tid och utvecklarna av det programmet har med stor säkerhet konsulterat användarna om vilka funktioner som är eftertraktade och hur de skall utformas.

Programmet som har utvecklats i det här projektet kräver i dagsläget att man både sätter sig in i programmets uppbyggnad och komponenten *JGraph* för att på ett bra sätt kunna lägga till funktionalitet som t.ex. grafoperationer. En möjlig strategi för vidareutveckling är att göra ett gränssnitt för att låta användarna på ett mycket enkelt sätt lägga till egen funktionalitet. Exempel på sådant skulle kunna vara någon typ av *plugin*[1]-system för att lägga in grafoperationer.

[1] Någonting som går att lägga till i ett datorprogram på ett enkelt sätt utan att det övriga systemet är beroende av det.

Kapitel 7

Tack

Jag har inte gjort projektet helt på egen hand utan har fått hjälp av många under arbetets gång. De som har varit mest involverad i projektet förutom mig själv är Jürgen Börstler och Klas Markström. Klas Markström är projektets initiativtagare och uppdragsgivare. Klas har förutom själva idén till projektet bidragit med kommentarer till funktionaliteten och det grafiska användargränssnittet. Jürgen Börstler har även han bidragit med några kommentarer till det grafiska användargränssnittet samt gett goda råd och stöd angående projektets upplägg och rapportens utformade. Tack också till min opponent Peter Sundqvist och min vän Showcat Gani vilka har hjälpt mig med språket i rapporten. Jag vill också tacka Per Lindström som är examinator för kursen som det här projektet ingår i. Han hjälpte mig att få det här arbetet genom att förmedla kontakt med Klas Markström.

Jag vill också tacka alla de som har arbetat med de mjukvarubibliotek och andra program som det här projektet är beroende av och har använt. Detta för att de har gjort det möjligt, genom att licensera dem på ett sådant sätt att det är tillåtet att använda dem och distribuera dem vidare, utan att hindra programmet från att vara lika öppet och vidareutvecklingsbart.

Bilaga A

Planering

Tidsåtgång	Planerad aktivitet
1 dag	Undersökning av om det finns redan existerande program som liknar den grafeditor som efterfrågas samt om det finns andra program eller mjukvarubibliotek som kan vara interessanta vid utveckling av en grafeditor.
1 dag	Utarbeta en detaljerad specifikation av projektet. Denna specifikation finns i sin helhet under kapitel 2.
1 dag	Verifiera den konstruerade specifikationen med uppdragsgivaren. Modifiera den efter förslag och kommentarer.
1 dag	Titta mer på möjliga mjukvarubibliotek och annat som kan vara användbart till projektet. (Skillnaden mot när det gjordes första gången är att det nu finns en specifikation till projektet.
1 dag	Utarbeta en designsketch av det grafiska gränssnittet. Visa den för uppdragsgivaren för att få kommentarer och verifiera så all efterfrågad funktionalitet finns med.
3 dagar	Läs på om de mjukvarubibliotek som är interessant för projektet och gör en objektorienterad design över programmet som täcker den övergripande strukturen.
2 veckor	Implementera de grundläggande funktionerna. De funktioner som behövs för att göra programmet användbart.
3 veckor	Utveckling av ytterligare funktionalitet, finesser, förbättring av nuvarande, dokumentation och testning. Beroende på hur mycket som har hunnits med och hur lång tid saker beräknas ta vid det här laget skall en ny tidsplan skrivas för de närmaste tre veckorna.
3 veckor	Arbete med rapport, förberedelse för presentation samt hjälp och dokumentation till programmet.

Tabell A.1: Tabellen visar den ursprungliga planeringen av arbetet.

Bilaga B

User's Guide

Contents

1 Introduction

The program is a graph editor developed for the mathematical field of graph theory. The program is crated on request of Klas Markström at the mathematical department in Umeå University as a Bachelor thesis project.

1.1 Purpose

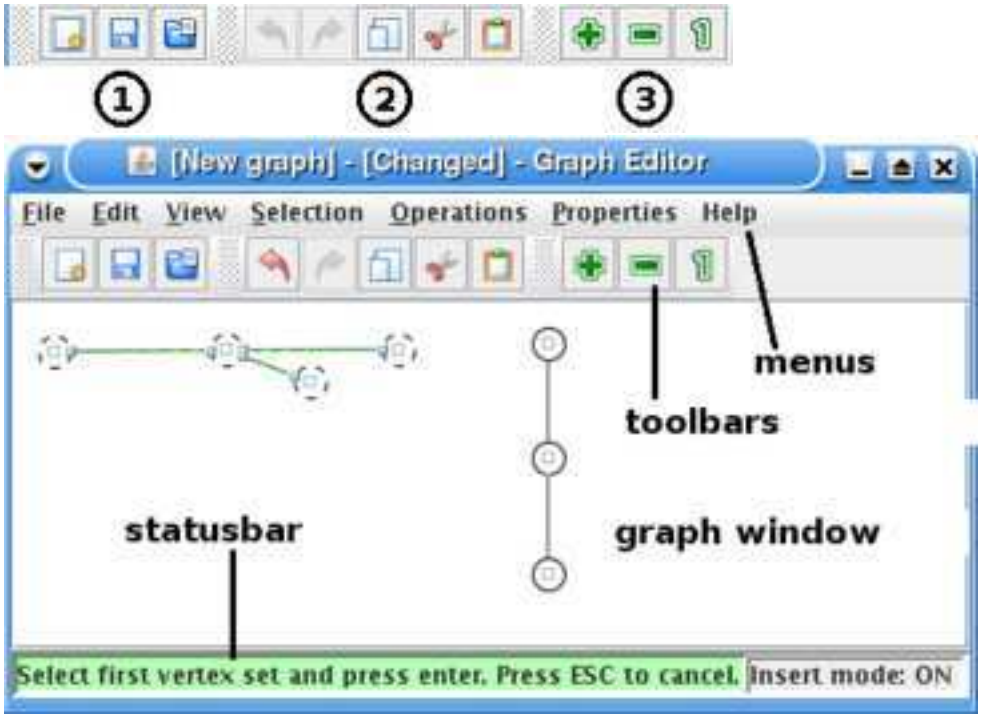
The main purpose of the graph editor is to make it easy to edit and create graphs that are used to illustrate theories and ideas in the mathematical field of graph theory. Many graph editors exist but just a few are developed with graph theory in mind and of them almost none are updated. Therefore there is a gap that this editor tries to fill.

1.2 Main functions and features

The program's main functionality is to *edit*, *save* and open *graphs*. This is described in the help section *editing graphs*. Other features are for example, graph operations and automatic embedding algorithms etc. The functions are summarized in the following list:

- Functions for copying and pasteing part of the graph
- Functions for unlimited undo and redo editing commands
- View functions for zooming in and out the view of the graph and to center the current graph
- Selection function to select parts of the graphs depending on the current selection (e.g. select neighbours)
- Graph operations and other graph manipulation functions
- Export functions to export the graph to other graph file formats:
  - Graphviz dot-file format
  - GraphML
  - Graph Modeling Language (GML)
  - File format compatible with Mathematica Combinatorica software
  - JPEG image file format
  - PNG image file format
  - *Mathematica* compatible file format
  - Neighbor lists
  - EPS image file format
- Import functions to import graphs from other file formats:
  - Random graph importer that creates a random graph from a number of vertices and a number of edges.
  - Mathematica compatible graph file format

2 Main window and toolbars

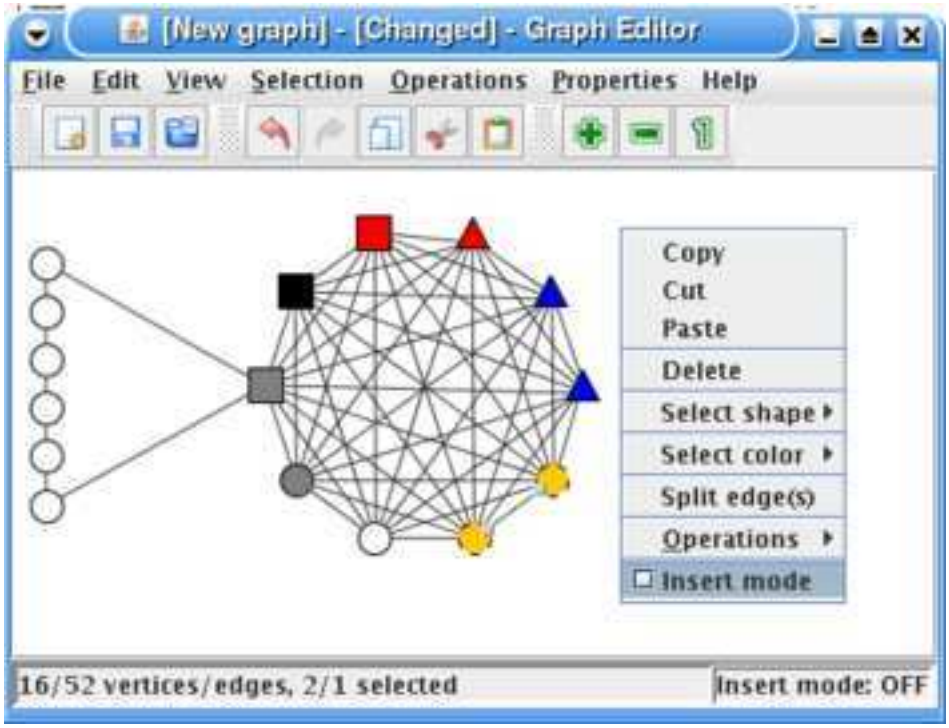


The main window has four head components that are marked in the figure above. They are described in the list below:

- The **graph window** is the place where graphs are edited and displayed. How editing is performed is described in more detail in the section called *editing graphs*.
- The **menubar** contains a number of menus to choose operations from. They are described in more detail in the section called *Menus and their operations*.
- The **toolbars** are used to fast access operations that are also in the menus. You can choose to hide the toolbars by using the *show toolbars* menu that is under the properties menu. The descriptions below correspond to the toolbar with the same number in the picture of the toolbars above.
    1. The file toolbar is used for some basic file operations. There are buttons on it to create a new graph, save the current graph and to open a graph from a file.
    2. The edit toolbar is used for some basic edit operations that also exists under the edit menu. The edit commands in the toolbar are for undo, redo, copy, cut and paste.
    3. With the view toolbar you can zoom in/out and set the zoom-level to the default.
- The **statusbar** is used to give varying information to the user. As default the number of vertices and edges and selected vertices and edges in the graph are displayed. The status bar is also used to give hints to the user of how to interact with the program. In the picture above the status bar contains a hint to the user during a graph operation that requires several interaction steps.

3 Introduction to editing

When the program starts it displays an empty graph in the *graph window*. The *graph window* is the component in the program window that is located in the center of the screen. The computer mouse is used to edit the graph.



3.1 Editing modes

There are two editing modes, one is called *insert mode on* and another is *insert mode off*. The current is indicated in the right corner of the status bar at the bottom of the window. The *insert mode on* is typically used to insert vertices and edges into the graph. The *insert mode off* is typically used to move vertices.

3.2 Ways to switch editing mode

The easiest way to switch editing mode is to use the *space* key on the keyboard. You can also click on the edit menu and select insert mode or right click in the editing window to get a menu where it is possible to change editing mode. Holding down the **A**-key will temporaily change the editing mode and it will only stay as long as you hold down the key. The **A**-key is intended for situation when you only want to do something fast in the other editing mode and after that go back to the original

3.3 Selection and related commands

Many of the commands that can be performed on the graph are dependent on which part of the graph is selected. To select one edge or vertex click on the vertex or edge you want to select. To select all vertices and edges in an area you can press the mouse button in one corner of the area and move it to the opposite corner and then release it. Press the control button on the keyboard to keep the selection when you are

selecting more edges or vertices.

3.4 Delete vertices and edges

To delete vertices or edges, first select the vertices and edges you want to delete, then use the delete or backspace key on the keyboard. You can also delete by using the delete menu item in the menu that comes up when you right click in the graph window.

4 To edit in insert mode

The intention of the insert mode is to make it easy and intuitive to put in vertices and edges in the graph.

The only action that is needed to put in a new vertex is to click with the mouse on the position where you want the vertex. The position where you click needs to be empty, if it isn't the object at the place you click will be selected and no new vertex will be created. To create an edge between two vertices, press the mouse button in the middle of one of the vertices and drag the mouse with the mouse button down, until the mouse pointer is in the middle of the other vertex and then release the mouse button.

To get a menu with commands click with the second mouse button in the graph window. The commands are different depending on the selection of vertices and edges.

5 Not insert mode

The intention of *not insert mode* (or *insert mode off*) is to make it easy to move vertices and modify a currently existing graph without adding new vertices and edges. It is also possible to add vertices in this mode but it is much less intuitive than in the *insert mode*. To add vertices click with the right mouse button on an empty area and select *Insert vertex*.

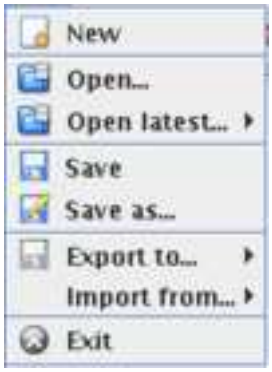
To move vertices in this mode it is just to press the mouse button when the mouse pointer is in the middle of a vertex and drag the mouse with the mouse button down until the mouse pointer is at the position where you want to place it.

To get a menu with commands click with the second mouse button in the graph window. The commands are different depending on the selection of vertices and edges.

5.1 Merge vertices

It is possible to merge two vertices in the not insert mode so that the resulting vertex is neighbour to all vertices that the vertices in the merging are neighbours to. To merge two vertices press the mouse button in the center of a vertex and drag the mouse so that the mouse pointer is at the center of the vertex you want to merge it with.

6 The file menu and its content



The file menu contains operations that has to do with files. The operations in the menu are described in the list bellow. The list is ordered in the same order as the menu items in the file menu.

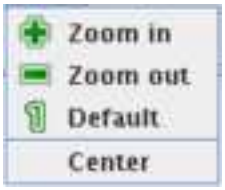
1. The **New** command opens a new window with an empty graph.
2. The **Open...** command opens a file chooser dialog to open a file from.
3. The **Open latest...** menu item has a sub menu which contains a list of recently used files. The list is ordered with the latest used file in the top of the list. If you click on one of them a new window with the corresponding file will be opened.
4. The **Save** command saves the graph if it is already associated with a file. If it is not associated with a file a dialog box where it is possible to select a file to save to will be displayed.
5. The **Save as...** command will show a dialog box where it is possible to select a file to save the graph to.
6. The **Export to...** menu item will show a menu with file formats it is possible to export the graph to.For more information see the section about export.
7. The **Import from...** shows a list of possible import alternatives. For more information see the section about import.

7 The edit menu and its operations

The edit menu contains operations that are used for the editing or effect the editing in some way. The operations in the menu are described in the list bellow. The list is in the same order as the menu items in the menu.

1. The **undo** command makes the last operation undone. The number of undo commands that is possible to perform is unlimited.
2. The **redo** command cancels the last undo command.
3. The **cut** command first saves all selected graph objects so they can be pasted with the paste command later and then removes them.
4. The **copy** command makes a copy of the selected objects in the graph so they can be pasted with the paste command.
5. The **paste** command paste the saved object from the cut or copy commands in the graph window. It is possible to paste parts of graphs that have been copied or cut from other graph windows.
6. The **Insert mode** alternative can be set to on and off.

8 The view menu and its content



The view menu contains operations that change the view of the graph in some way.

The operations that are in the menu are described in the list below. The list is ordered in the same way as the menu items.

1. The **Zoom in** command zoom the current graph. This means that the size of everything is increased. If you want to expand the graph so that the distance of the vertices increases but still have the same size of the vertices and edges then look at the *expand selected by factor* command in the operations menu.
2. The **Zoom out** command is the opposite of the *zoom in* command. It decreases the size of the graph.
3. The **Default** command sets the zoom level to the default.
4. The **Center** command centers the graph in the middle of the window. The center of the graph is the center of the least bounding rectangle that contains all vertices.

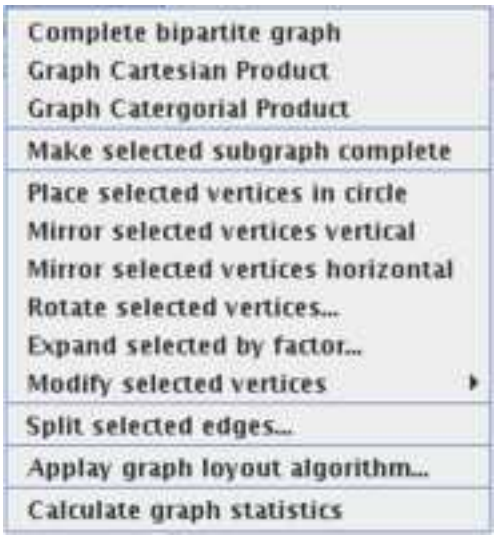
9 The selection menu and its content

The selection menu contains operations that are used to select different parts of the graph. All commands except the *select all* requires that at least one vertex is selected. Therefore they are disabled when no vertex is selected.

The operations in the menu are described below. The list is ordered in the same order as the menu items in the selection menu.

1. The **Select all** command selects all vertices in the graph.
2. The **Select neighbour** command selects all neighbour vertices to the currently selected vertices.
3. The **Select reachable** command selects all vertices that are reachable by only *walking* on the edges from the selected vertices.
4. The **Select other** command unselects all selected vertices and selects all previously not selected vertices.

10 The operations menu and its content



This menu is also available under the menus that come up when you click with the right mouse button in the *graph window*. The menu contain commands that perform some kind of operation on the graph. Most operations change the graph, but there is also one command that traverses the graph to take information from it, namely *Calculate graph statistics*. The commands have been divided by lines in the menu into categories. All menu items in the same category are similar to each other in some way. That has been done to make it easier to find a certain command. The description of the menu commands below is also divided in the same way. The description marked with the number one describes the first category from the top of the menu and so on.

1. The first category contains operations or graph products that need two subgraphs. They are selected by the user in two steps:

   When a command has been started by a click on a menu item in the category the statusbar in the bottom of the window displays a message. The message tells the user to select the vertices that is in the first subgraph and then press *enter*. When *enter* is pressed the user can select the second subgraph in the same way. Because the operations need two disjoint vertex sets it is not possible to select vertices in the first set when the second is being selected. They are therefore marked with a red cross. To cancel a started operation the user can press the *escape* key.

   The operations in this category is:

- The **Complete bipartite graph** operation creates a complete bipartite graph of two vertex sets. The operation does not remove edges that are already there. A complete bipartite graph made up by two vertex sets A with n vertices and B with m vertices is denoted by $K_{m,n}$ and is a graph where every vertex in A is connected to all vertices in B and every vertex in B is connected to all vertices in A.
- The **Graph Cartesian Product** operation creates a graph that is the *Graph Cartesian Product* of two graphs G_1 and G_2. The resulting graph is placed to the left under the orginal graph in the *graph window*. For a definition of the Graph Cartesian Product please see, `http://mathworld.wolfram.com/GraphCartesianProduct.html`.
- The **Graph Categorical Product** operation creates a graph that is the *Graph Categorical Product* of two graphs G_1 and G_2. The resulting graph is placed to the left under the orginal graph in the *graph window*.For a definition of the*Graph Categorical Product* please see, `http://mathworld.wolfram.com/GraphCategoricalProduct.html`.

2. This category contains operations that only use one graph as operand. The operand is the graph created by the selected vertex set. The menu item in this category is only enabled if one or more vertices is selected. At the moment there is only one operation in this category. The **Make selected subgraph complete** operation makes the subgraph created by the selected vertex set to a complete subgraph by adding edges so every vertex is neighbour to every other vertex in the vertex set.

3. This category contains operations that modify the position of selected vertices. The menu items is only enabled if any vertices are selected. The operations in this category is listed below with an explanation of what they do:

   - The **Place selected vertices in circle** operation reorganize the selected vertices so they are placed in the edge of a circle with the same distance between each vertex. The circle's diameter is equal to the greatest of the width and the height of the least bounding rectangle of the selected vertices. The circle's center is in the center of the bounding box.
   - The **Mirror selected vertices vertical** operation mirror all selected vertices over the x-axis through the center of the least bounding rectangle of all selected vertices. To mirror a vertex over the x-axis is to change its horizontal position so it is on equal distance from the axis but on the opposite side.
   - The **Mirror selected vertices horizontal** does the same thing as *Mirror selected vertices vertical* does but over the horizontal axis instead of the vertical.
   - The **Rotate selected vertices** operation brings up a rotation dialog where it is possible to select a rotation angle for the selected vertices. When the angle is changed in the dialog the graph is instantly updated. To keep the new angle press the *ok* button and use the *cancel* button to cancel the changes. The vertices is rotated around the center of the least bounding rectangle for the selected vertices.
   - The **Expand selected by factor...** command brings up a dialog. In the dialog it is possible to select a decimal expand factor. There are two buttons. A *cancel button* that closes the dialog and an *expand button* that expand or shrink the selected vertices by the expand factor. The name of the command is misleading in the way that it can not only be used to expand but also to shrink. When the *expand button* is pressed the vertical and horizontal distances between vertices is multiplied with the expand factor.
   - The **Modify selected vertices** menu item brings up a submenu where it is possible to change the color and the shape of selected vertices.

4. This category contains edge operations. Items in this category is only enabled when one or more edges are selected. The **Split selected edges...** command brings up a dialog where you can select a number of vertices to input for each selected edge. When the *ok button* is pressed every selected edge

is removed and then the number of vertices that have been chosen is inserted between all the vertices that previously where neighbours. New edges are also created between the newly inserted vertices and the vertices that was previously neighbours so a path with the length (number of inserted vertices + 1) are created between all vertices that previously were neighbours.

5. The **Apply graph layout algorithm...** command brings up a dialog where you can select a number of graph layout algorithms that you can execute on the graph. You can select to execute most of the algorithms on selected vertices or all vertices.

6. The **Calculate graph statistics** command first traverses the graph to catch some information from it and then displays a dialog with information about the current graph.

11 The properties menu and its content

The properties menu contains menu items used to change the properties of the program and the graph. The **Show quickmenus** alternative shows a submenu where it is possible to select which quickmenus that are displayed.

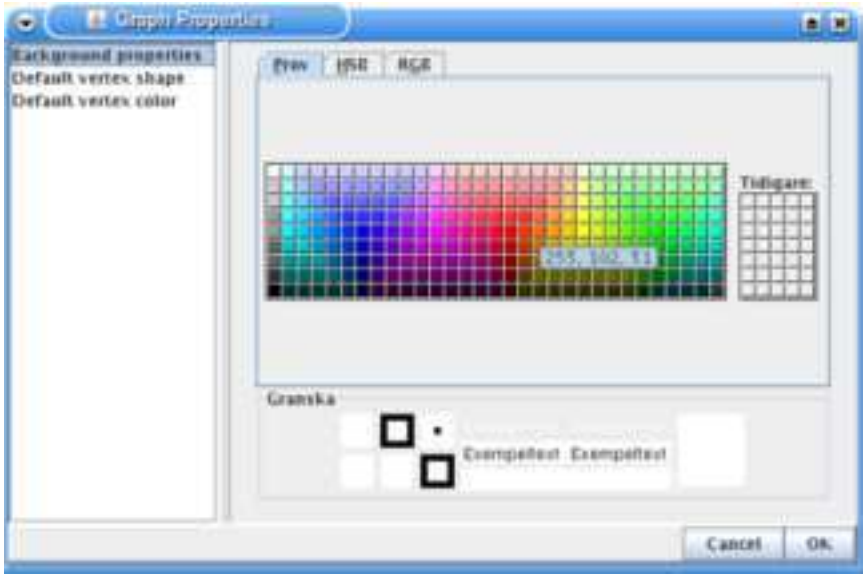
The dialogs that comes up when **Graph properties...** and the **Properties...** alternatives have been chosen, are described in the two next sections.

11.1 The graph properties dialog

This dialog comes up when you select the **Graph properties...** alternative. In this dialog it is possible to select settings that is directly related to the graph. The settings is saved to the graph file when you save a graph and loaded when you open a graph from a file.

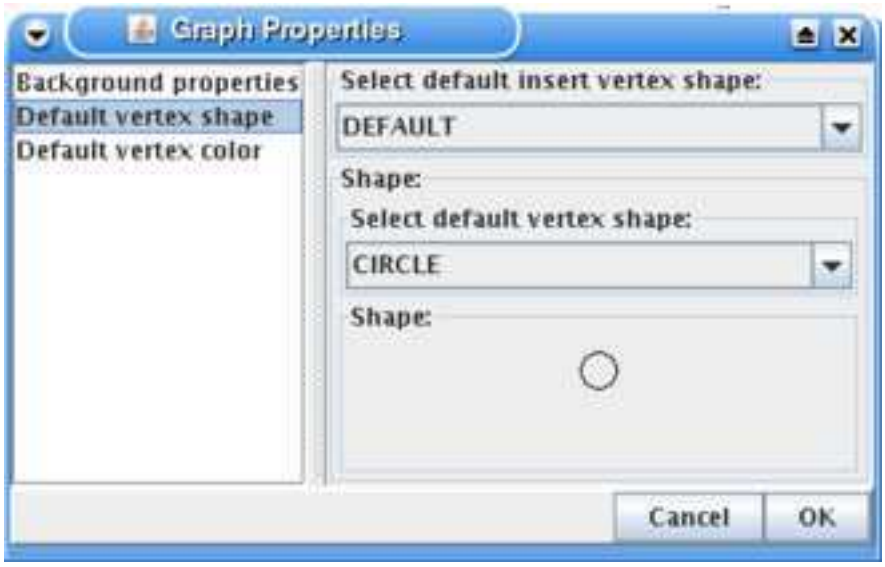
The dialog is divided in two parts. One is the list that is to the left. The list contains a number of names of settings modules. To the right, the currently selected settings module is displayed. To switch between settings module click on the name of the settings module that you want to change. To save all changed settings that have been done in the different settings modules click on the ok button in the bottom right corner. If you don't want to save the changes click on the cancel button. The different settings modules are described in the sections below.

11.1.1 Background properties



Here the background color of the graph is selected.

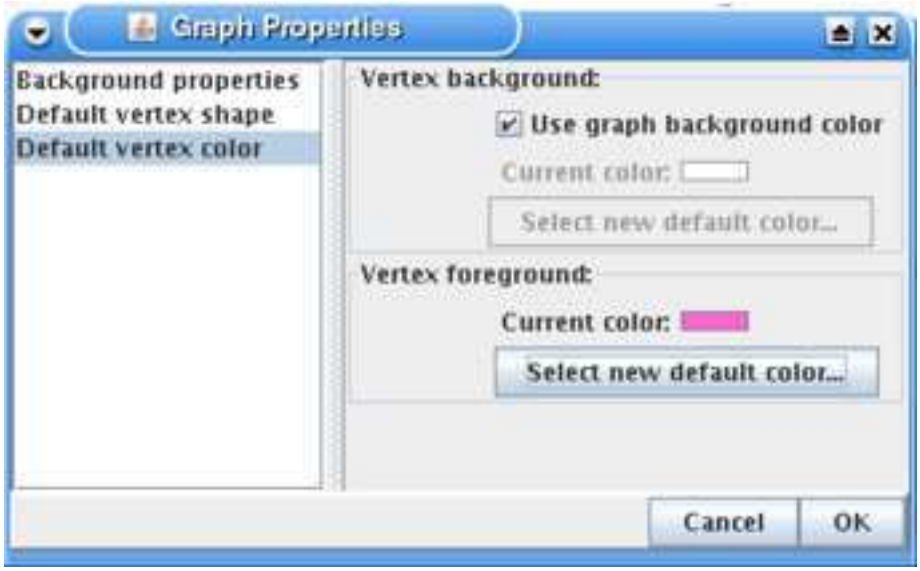
11.1.2 Default vertex shape



In this module you can select the default vertex shape. The default vertex shape is the shape of the vertex that is inserted when you insert vertices. The shape that you select in the drop down menu in the top of the module is displayed under it.

There exists one shape called *DEFAULT* that is different from the others. The *DEFAULT* shape can be any of the other shape. Therefore when you select the *DEFAULT* shape it is also possible to select which shape that shall be set as the *DEFAULT* at the moment. If you have inserted vertices with the *DEFAULT* shape it is possible to change them afterward by changing the actual shape of the *DEFAULT* shape.

11.1.3 Default vertex color



Here it is possible to select the default vertex color. That is the color of the vertices created with insert commands like clicking in the *graph window* when insert mode is on. The module is divided into two parts.

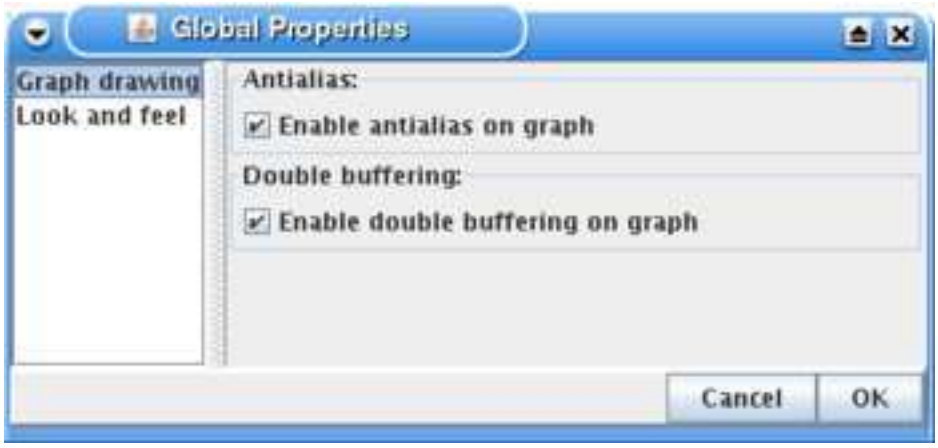
In the upper part you can select the background color of vertices. That is the color inside the selected shape that has borders with the foreground color. One option is to use the graph background color of the graph as the background color for vertices. If you unselect that option it is possible to choose a background color by clicking on the *Select new default color...* button.

You can select the default vertex foreground color in the lower part. The foreground color is the border of the shape that the vertex is made up by.

11.2 The global properties dialog

The global properties is the program properties that is used for all opened programs. The global properties is saved in the file *.grapheditor/properties.xml* that is placed in the user home directory. The user home directory is the directory given by the *java system* as the home directory. The dialog's structure is the same as in the *graph properties dialog*. The different settings modules in it are described in the sections below.

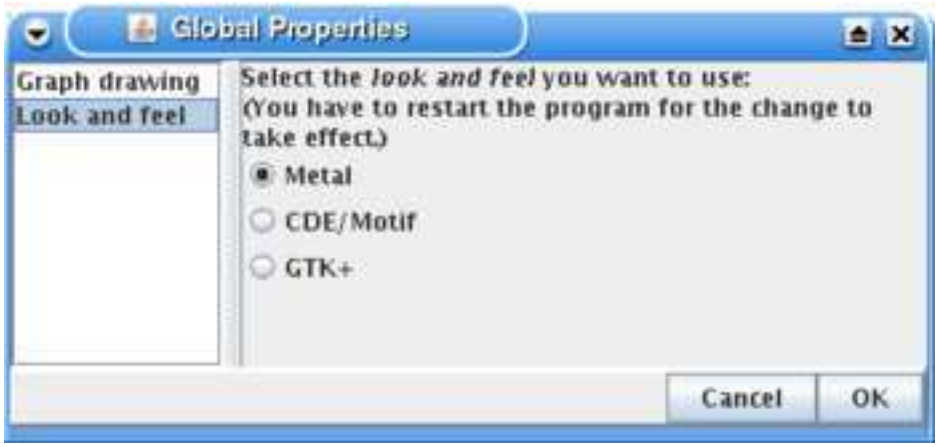
11.2.1 Graph drawing



Here you can change some drawing alternatives for the graph.

- If the **Antialias** alternative is selected the graph will be drawn with antialiasing. That means that lines and shapes get smoother edges. Turn this off if the program runs slowly
- If the **Double buffering** alternative is selected the graph will be drawn with the *double buffering* technique which means that the images is first drawn to an off screen buffer and then to the on screen buffer. This will remove flicker effects but can possible make the program consume more processor cycles and memory.

11.2.2 Look and feel



In this module you can select the *look and feel* for the program. A *look and feel* is a collection of settings that decide how the user interface will look and behave. Which *look and feel*s that is available depend on the java version the program is running on.

12 Export

The file menu contains a submenu with a number of different file formats that it is possible to export the graph to. The submenu is under the menu item called *Export to...*. Below are the different file formats

described with referenses to webpages where it is possible to find more information.

12.1 Graphviz dot-file format

This format can be used with the Graphviz tools produced in the Graphviz project. The Graphviz tools can create images of graphs from simple graph descriptions. The generated file is a description of a undirected graph created from the current graph. See the Graphviz documentation for information of how to change the files and configure the output. Below is an example of a command that produces a PNG-image file with a graph drawn from the description in the test.dot file:
neato test.dot -Teps > test.eps

http://www.graphviz.org

12.2 GraphML

GraphML is an XML based file format for describing all types of graph structures. Among other things it has support for both undirected and directed graphs. The generated file is a description of an undirected graph.
http://graphml.graphdrawing.org

12.3 Graph Modeling Language (GML)

GML was created to make a universal graph file format that could be used in all types of graph programs. It is possible to attach all kind of data to the graph objects.
http://www.infosun.fim.uni-passau.de/Graphlet/GML

12.4 File format compatible with Mathematica Combinatorica software

This file format is a simple neighbour list with position information for vertices. It can be opened with the Mathematica software.

Each row in the file format represents a vertex. The first number on each line is a vertex identification. The two floating point number next on the line is the horizontal and vertical coordinates of the vertex. The numbers coming after that is the identifications of the vertices that are neighbours to the vertex.
http://www.wolfram.com

12.5 Neighbourlist

The files exported by this exporter are exactly the same as files that are exported by the *Mathematica* exporter except that they do not contain vertex position information.

12.6 JPEG image file format

When you export to a JPEG (Joint Photographic Experts Group) image the graph is exported exactly as it looks like in the *graph window* except that the selection is cleared. This means that if you zoom in the graph the exported image will be bigger. The image exported is the same size as the least bounding rectangle of all vertices plus an empty border with 5 pixels.
http://www.jpeg.org

12.7 PNG image file format

The PNG (Portable Network Graphics) export is similar to the JPEG export but produces a PNG image.
http://www.w3.org/Graphics/PNG

12.8 EPS image file format

The EPS (Encapsulated PostScript) image files produced by this exporter uses vector graphics which makes the images scalable. It is a very popular file format for images included in e.g. scientific reports.
`http://www.prepressure.com/formats/eps/fileformat.htm`

13 Import

The file menu contains a submenu with different import alternatives. The submenu is under the menu item called *Import from...*. Below are the different import alternatives described.

13.1 Random graph importer

The random importer imports a graph with a random structure. A dialog is displayed when you choose this option. In the dialog, it is possible to select a number of vertices and a number of edges. The edges is randomly connected from one vertex to another until all edges has been connected somewhere. The vertices are placed in the circumference of a circle. It is also possible to change the distance between vertices in the dialog.

13.2 Mathematica compatible graph file format

The Mathematica compatible file format importer imports graphs from a graph file format that can be exported fromMathematica. This format is a simple neighbor list with vertex position information. See the information about *Mathematica compatible graph file format* under the section about exporters for more information.

14 Used libraries

The program uses a lot of third part libraries. In the table below they are listed with information about which license they uses and where you can find more information. The program also uses *Java* standard class library but it is not listed below.

Name	License	Website
JGraph	LGPL	http://www.jgraph.com
JGraphT	LGPL	http://www.jgrapht.org
JGraphpad CE	GPL	http://sourceforge.net/projects/jgraph
XStream	Modified BSD license	http://xstream.codehaus.org
EPS Graphics Library	GPL	http://www.abeel.be/epsgraphics
JavaHelp System	GPL with the class path exception	http://javahelp.dev.java.net
BrowserLauncher2	LGPL	http://browserlaunch2.sourceforge.net

The icons used in the program is created by David Vignoni. They are licensed under the LGPL license and can be downloaded from *http://icon-king.com*