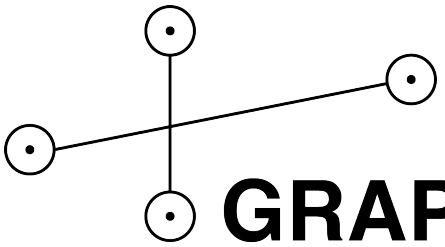


Software Design Document

Version 1.0
04. Juni 2013

ISGCI – Information System on Graph Classes and their Inclusions
Team *Graph Maga*

Inhaltsverzeichnis

1 Änderungsverlauf

Version	Datum	Beschreibung der Änderung
0.4	26.05.2013	Arbeitsversion
0.5	07.06.2013	1. Einleitung verbessert, Ausdruck
0.5	07.06.2013	2. Designziele verbessert, Ausdruck
0.5	07.06.2013	3. ProblemsFunktionen des bestehenden Systems: Formulierungen
0.5	07.06.2013	Help, About, Zeichenfläche Kontextmenüs: Kommasetzung, Quoted Text (Gänsefüßchen), Format angepasst
0.5	07.06.2013	Information − Verweis auf Browse Database → erledigt, Ausdruck verbessert.
0.5	07.06.2013	4. Wichtige Klassen des bestehenden Systems JGraphT: Formulierungen angepasst Layout: Formulierungen, Kommata, Quotes GUI: Formulierungen, Kommata, Quotes
0.5	07.06.2013	5. Vorgeschlagene Software Architektur: 5.1 Übersicht: Kommata, Ausdruck, Formulierungen, Quotes 5.2 ISGCIGraph.java: Kommata, Ausdruck 5.3 ISGCI.gui: Kommata, Ausdruck
0.5	07.06.2013	6. HW/SW Mapping Ausdruck, Kommata, Formulierungen
0.5	07.06.2013	7. Testing: Kommata, Formulierungen verbessert.
0.6	07.06.2013	Gesamtes SDD: Rechtschreibung verbessert.

2 Einleitung

Das gelieferte Produkt ist eine Aktualisierung des bestehenden ISGCI. Das bestehende ISGCI wird mit Funktionalitäten der Graphenzeichenbibliothek JGraphX erweitert. Außerdem werden der neuen Version verschiedene neue Features hinzugefügt, die zur Verbesserung der Benutzerfreundlichkeit beitragen sollen. Zweck des Updates ist es, nicht nur die Verfügbarkeit des Tools in Zukunft zu gewährleisten und neue Funktionen zu implementieren, sondern auch die Instandhaltungskosten zu senken und das Tool für interessierte Entwickler und Nutzer ansprechender zu gestalten.

2.1 Designziele

- **Benutzerfreundlichkeit**
  Das bestehende System wird vor allem deshalb erweitert, um die Benutzerfreundlichkeit zu steigern. Zwar besteht ein Teil der Zielgruppe für die Software aus technisch interessierten und fähigen Personen, dennoch gibt es auch eine breite Nachfrage im Bereich der Computer-Laien, weshalb die Software auch für diese leicht durchschaubar und selbsterklärend sein soll. Intuitiv angeordnete Reiter in der Menüleiste, sowie ein interaktives Kontextmenü sollen dazu beitragen, dass benötigte Funktionen von Benutzern leicht gefunden werden können. Zudem sorgen Funktionen wie „Grab&Pull“ oder Zoom für einen interaktiven Umgang mit dem System. Um die Übersichtlichkeit über komplexere Graphen zu gewährleisten, gibt

es zwei Varianten des „Expanding/Collapsing“, mit denen man einen komplizierten Graphen in einen überschaubaren Teilgraphen verkleinern kann.

- **Integrität**
  Alle Datensätze werden zentral auf einem Server gehalten. Die Software bietet ausschließlich lesende Funktionen an. Somit ist die Datenbank durch negative Manipulationen geschützt.

- **Flexibilität**
  Nutzer können ihren Wunschgraphen zeichnen lassen, diesen auf verschiedene Arten manipulieren (z.B.: neu anordnen, reduzieren, Knoten-Hierarchien anzeigen lassen) und den entstandenen Graphen exportieren. Dazu können zu jedem Graph-Knoten (Graphklasse) Informationen aus der Online-Datenbank eingesehen werden.

- **Portabilität**
  ISGCI ist eine Java Anwendung. Das bedeutet, dass sie plattformunabhängig ausgeführt werden kann, solange eine Java-Laufzeit-Umgebung installiert ist (als Standard vorausgesetzt).
  Die Online-Datenbank ist unabhängig von unserem System.

- **Wartbarkeit**
  Dadurch, dass die Datenbank des Programmes online, zentral gelagert wird, lassen sich Datensätze leicht und ohne Updates im Programm manipulieren oder hinzufügen.

2.2 Referenzen

Tabelle 1: Verweise

Name	**Quelle**	**Datum**
ISGCI	Graphclasses.org - ISGCI Homepage	04.06.2013
JGraphT	JGraphT Website	04.06.2013
JGraphX	JGraph Website	04.06.2013
	JGraph Dokumentation	04.06.2013
	JGraphX Library Quellcode	04.06.2013
Swing	Swing Dokumentation	04.06.2013
Java	Java 6 Dokumentation SE	04.06.2013

2.3 Übersicht

- Funktionen des bestehenden Systems
  Beschreibung des bestehenden Systems mit allen Funktionen. Systematische Aufgliederung aller Reiter und Interaktionsmöglichkeiten.

- Wichtige Klassen des bestehenden Systems
  Kurze Erläuterung der Aufgaben benötigter Packages und deren Klassen. Heraushebung der Zusammenhänge innerhalb der Packages.

- Vorgeschlagene Software Architektur
  Definition des Aufbaus und der neuen, hinzuzufügenden Funktionen im erweiterten System.

- Hardware/Software Mapping
  Beschreibung des notwendigen Software-Umfeldes der Software und Hardwarevoraussetzungen.

- Integrationstests
  Aufstellung aller Integrationstests.

3 Funktionen des bestehenden Systems

Beschreibung der aus dem GUI ableitbaren Funktionen

3.1 File

Der Menüreiter „File“ enthält drei grundlegende Funktionen zur Bedienung des Programms. Es lassen sich weitere Fenster der ISGCI Applikation öffnen, falls man beispielsweise mehrere Graphen gleichzeitig zeichnen lassen und betrachten möchte.
Zudem lässt sich das Dialogfenster „Export drawing...“ öffnen um einen gezeichneten Graphen in eine Datei zu exportieren. Dabei stehen drei Dateiformate zu Verfügung: Postscript, Structured Vector Graphics und GraphML. Zuletzt lässt sich mit „Exit“ das aktuelle Fenster schließen.

3.2 View

View stellt beim Klicken auf den Menüreiter „View“ in der Navigationsleiste verschiedene Funktionen zur Optimierung der Ansicht zur Verfügung. Darunter fällt das Suchen nach einer Graphklasse, die Einstellung der Namensanzeige und das Markieren von „unproper“ (müsste eigentlich improper heißen) inclusions.

Search in drawing...
Beim Auswählen des Punktes „search in drawing“ durch einen Mausklick, öffnet sich das Fenster „Search for a graphclass“. Aus einer Liste kann in diesem Feld eine Graphklasse aus allen Graphklassen ausgewählt werden und durch das Drücken von „Search“ gesucht werden. Dabei werden stets nur die Graphklassen angezeigt, die im Drawing auffindbar sind. Durch „cancel“ kann der Suchvorgang abgebrochen werden. Sucht man mittels Drücken auf „search“ nach einer aus der Liste gewählten Klasse, dann wird der Canvas so verschoben, dass die Graphklasse sichtbar ist.

Naming preference...
Beim Auswählen des Punktes „Naming preference..“, durch einen Mausklick, öffnet sich das Fenster „Naming preference“. Dieses stellt drei Möglichkeiten durch Radio-Buttons zur Auswahl. Dabei kann immer nur ein Radio-Button aktiv sein. (Eigenschaft der Radio-Buttons)

Basic e.g. threshold

Forbidden subgraphs e.g. $(P_4, 2K_2, C_4)$-free

Derived e.g. cograph $\cap$ split

Das Auswählen einer dieser drei Optionen und das Bestätigen durch „OK“, hat eine Umbenennung aller Knoten im Drawing entsprechend angegebenen Konvention zur Folge.

Mark unproper(improper) inclusions
Beim Auswählen des Punktes „Mark unproper inclusions“, durch einen Mausklick, wird dieser Menüpunkt mittels eines ein-/ausblendbaren Häkchen als aktiv / inaktiv gekennzeichnet. Wird dieser Menüpunkt als aktiv gekennzeichnet, so werden alle „unproper inclusions“ durch verblasste (geringere Opacity, Grauwert anstelle von Schwarz) Pfeile an einer Kante dargestellt. Wobei der Pfeil auf die Graphklasse mit „unproper inclusions“ zeigt. Nach Deaktivieren dieser Funktion verschwindet dieser graue Pfeil.

3.3 Graph classes

Der Reiter „Graph classes“ in der Menüleiste stellt verschiedene Funktionen zum Finden von Klassen anhand derer Eigenschaften und Browsen durch die Graph-Klassen-Datenbank zur Verfügung, die im Folgenden näher erläutert werden.
Bevor folgende Schritte getan werden können, muss der bereits genannte Reiter per Mausklick (egal ob rechts oder links) ausgewählt(geöffnet) werden.
Ein wichtiger Punkt ist, dass egal welche Fenster geöffnet sind, der Benutzer nicht im Hauptfenster blockiert wird, auch wenn er über dieses bereits ein anderes geöffnet hat. Dies ist für den Benutzer sehr hilfreich, da er sich keine Informationen zur aktuellen Einstellung (z.B. im „Browse Database“) merken muss und gleichzeitig nach etwas anderem schauen kann.

Browse Database
Durch Klicken auf die Auswahlmöglichkeit „Browse Database“ öffnet sich das Fenster „Graph Class Information“. Nun hat man die Möglichkeit Graphklassen anzuwählen, um über diese Informationen zur Komplexität der Probleme zu Graphklassen zu erhalten (Eine Tabelle mit 2 Spalten: Problem und Complexity; nicht alphabetisch geordnet). Außerdem steht ein Filter zu Verfügung, mit dem man durch Eingeben des Namens oder nur ein Teil davon, dabei ist egal, an welcher Stelle er im Namen der Graphklasse steht, die Informationen zu Klassen, welche die Eingabe beinhalten, angezeigt bekommt.
Zu diesen Informationen gehören nicht nur die Komplexität von Problemen. Unter der bereits genannten Tabelle sind 3 weitere, nebeneinanderstehende Listen mit gleicher Breite angeordnet. Diese Listen beinhalten (von links nach rechts erklärt) alle Superklassen, Äquivalenzklassen und Subklassen (zur in der ersten Liste ausgewählten Graphklasse). Durch Verändern der Breite und Höhe des Fensters verändert sich dynamisch die Größe der Listen. Während sich bei der Tabelle(Probleme) nur die Breite verändert, da diese eine feste Anzahl an Zeilen hat, verändert sich die Höhe der Listen ebenfalls relativ zur Fenstergröße. Die Breite der „Graph Class“-Liste erstreckt sich über die Breite des gesamten Fensters(entsprechend der 3 Listen im unteren Teil zusammen). In jeder Liste sind die Klassen alphabetisch angeordnet (Groß- vor Kleinbuchstaben; Sonderzeichen werden ignoriert), sodass man seine gewünschte Graphklasse auch darüber filtern kann. Wichtig ist, dass in jeder der 4 Listen, in denen die aktuellen Graphklassen bezüglich ihrer Eigenschaft zur Auswahl, angewählt werden können, sodass man sich durch die Zusammenhänge der Klassen „klicken“ kann. Zu beachten ist, dass durch einfaches Anwählen die Klassen in den unteren 3 Listen nur markiert werden und erst deren Informationen nach einem Doppelklick(Linksklick) angezeigt werden, während man durch einfaches Anwählen einer Klasse in der oberen Liste, sofort dessen Informationen zu entsprechenden

Problemen erhält (ohne Doppelklick).
Ganz unten im Fenster werden zentriert verschiedene Buttons angezeigt:

Class details: Ist der 1. Button (von links). Durch einfachen Linksklick wird die entsprechende Detail-Seite im Standard-Browser geöffnet und zeigt ausführlichere Informationen zur aktuell ausgewählten Graph Klasse der oberen Liste im Fenster. Dazu gehören „References, Related classes, Inclusions, Problems(alphabetisch geordnet)“

Inclusion info: Ein Linksklick auf diesen Button (2.) öffnet ein weiteres (kleines) Fenster(„Relation“), welches Informationen zu Inklusionen zu anderen Graphklassen anzeigt. Im unteren Teil dieses Fensters befinden sich 3 Buttons:

- View references: öffnet eine Seite im Standard-Browser mit Referenzen auf Literatur bezüglich der angegebenen Inklusion(en) („Relation“).
- Draw: Zeichnet im Hauptfenster(„ISGCI“) einen Graphen inklusiv der Inklusionen zu den aktuell im Relation-Fenster angezeigten Inklusion.
- OK: Einfacher Button, dessen Betätigung zur Schließung des Relation-Fensters führt.

Draw: Durch Betätigung dieses Buttons(3.) wird ein weiteres Fenster („Select Graph Classes“) geöffnet, welches wie im vorigen einen analogen Filter zur Limitierung der angezeigten Klassen in der darunter befindlichen Liste hervorruft. In diesem Fenster steht im Gegensatz zum vorigen Fenster („Graph Class Information“) nur eine Liste, welche als Standard-Auswahl immer die Graphklasse des vorigen Fensters ausgewählt hat, damit der Benutzer nicht erneut die Klasse suchen muss, welche er im Hauptfentser („ISGCI“) anzeigen möchte. Der Benutzer hat zusätzlich über zwei Checkboxen, die sich unter der Liste befinden, die Möglichkeit, zu der ausgewählten Graphklasse auch deren Super- bzw. Subklasse durch Aktivieren der Checkboxen für die jeweiligen Klassen mit in dem zu zeichnenden Graphen anzuzeigen.
Im unteren Teil dieses Fenster befinden sich der Button „New drawing“ über den der Benutzer seine Auswahl bestätigen kann, sich das Fenster schließt und im Hauptfenster das Ergebnis (den gezeichneten Graphen mit/ohne seine Super-/Subklassen) betrachten kann. Rechts neben dem eben genannten Button befindet sich ein „Cancel“-Button, welcher das Fenster schließt, ohne dass sonst etwas im Programm geschieht und der Benutzer wird zu dem Fenster zurückgeführt, von dem aus er dieses geöffnet hat („Graph Class Information“).

Close: Wenn der User diesen Button (ganz rechts; 4.) klickt (Linksklick), wird das Fenster „Graph Class Information“ geschlossen und der Benutzer gelangt zum Hauptfenster „ISGCI“ zurück, in dem die letzte ausgewählte Zeichnung noch angezeigt wird.

Find Relation...
Über diesen Auswahlpunkt öffnet sich durch Klicken (rechts/links) das Fenster „Find Relation“. In diesem hat der Benutzer die Möglichkeit über 2 untereinander dargestellte Listen, welche alle Graphklassen beinhalten, oder über einen

Filter (je Liste), zwei Graphklassen anzuwählen und sich über einen darunter befindlichen Button „Find Relation“ deren Verbindung anzusehen. Die Filter Funktion ist dabei analog zu der Filter-Funktion in dem Auswahlpunkt „Browse Database“ umgesetzt. Nach Betätigung des eben genannten „Find relation“ Buttons öffnet sich ein weiteres Fenster, welches nun Informationen zur Relation der angewählten Klassen anzeigt (wird im links oben auf dem Bildschirm geöffnet). Die angezeigten Verbindungen der beiden Klassen werden begründet, in welchem Zusammenhang sie stehen und nicht nur, dass sie es tun („by definition, forbidden subgraphs ...“). Dabei zu beachten ist, dass sich die Größe des Fenster an der Menge des zu anzeigenden Textes orientiert, sodass das Fenster möglich wenig Platz auf dem Bildschirm einnimmt. Zusätzlich zu diesen Relation-Informationen kann man über den Button „View references“ (links unten im Fenster) Referenzen zu Literatur über die graphclasses.org - Homepage anzeigen lassen. Über einen weiteren Button (rechts von „View references“) kann der Benutzer diese Relation im Hauptfenster zeichnen lassen (mit dazwischenliegenden Klassen), analog zur „Draw“-Funktion des bereits erklärten Auswahlpunktes. Ein dritter Button „OK“ (rechts von „Draw“) führt (ebenfalls analog) zur Schließung des „Relation“ Fensters und der Benutzer kommt zum „Parent-Fenster“ zurück.
Rechts neben dem Button „Find relation“ befindet sich der Button „Close“ (beide ganz unten im Fenster horizontal zentriert), der ebenfalls analog zur schon bereits genannten Funktion dieses Fenster schließt und zum Hauptfenster zurückführt.

Draw...
Dieser letzte Auswahlpunkt im Reiter „Graph classes“ der Menüleiste führt zu dem exakt identischen Fenster, wie beim Klicken des „Draw“ Buttons über den Auswahlpunkt „Browse Database“ (nur eben nicht über diesen Umweg), falls der Benutzer schon weiß, welche Klassen er gezeichnet bekommen möchte. Dieses Fenster hat den selben Namen „Select Graph Classes“ und beinhaltet ebenso einen Filter und eine Liste zu allen Graph-Klassen, in dem die gewünschte Klassen angewählt und mit/ohne deren Super-/Subklassen gezeichnet werden kann. Mit der Bestätigung über den Button „New drawing“ schließt sich das Fenster und die Auswahl wird in dem Hauptfenster graphisch dargestellt.

3.4 Problems

Der Menüpunkt Problems stellt Funktionen zur Verfügung mit denen man untersuchen kann ob und in welcher Laufzeit ein Graphproblem auf den verschiedenen Graphklassen gelöst ist
http://www.graphclasses.org/help.html#problems

Boundary/Open Classes
Wird benutzt um herauszufinden, welche Probleme in P oder in NP liegen und wo die Grenzen sind. Man kann mit dieser Funktion neue Graphen zeichnen.
Implementierte Probleme:

- Recognition
- Treewidth
- Cliquewidth
- Cliquewidth expression
- Weighted independend set
- Independend set
- Clique
- Domination
- Colourability
- Clique cover
- 3-Colourability
- Cutwidth
- Hamilton circle
- Hamilton path
- Weighted feedback vertex set
- Feedback vertex set

Color for Problem
Zeigt an, in welcher Laufzeit ein Algo in der jeweiligen Klasse zu lösen ist. Bei einigen Algo's gibt es spezielle Regeln, wie die Einfärbung funktioniert. Beschreibung siehe http://www.graphclasses.org/help.html#problemdefs
Implementierte Probleme: Siehe oben.

3.5 Help

Small Graphs
Beim Auswählen des Punktes „Small Graph" wird die URL
http://www.graphclasses.org/smallgraphs.html
aufgerufen.

Help
Beim Auswählen des Punktes „Help" wird die URL
www.graphclasses.org/help.html
aufgerufen.

About
Beim Auswählen des Punktes „About" öffnet sich ein Fenster mit dem Namen „About ISGCI". Darin steht die Bedeutung von ISGCI (Information System on Graph Classes and their Inclusions). Es ist außerdem die Versionsnummer aufgeführt, dass es von H.N. de Ridder et al. geschrieben wurde und dass es das JGraph Library nutzt. Es wird weiterhin aufgeführt, wie viele Klassen und Inklusionen in der Datenbank sind und wann diese erstellt wurde. Es ist ein Link zur Website angegeben. Zum Schließen des Fensters, kann man auf „OK" klicken.

3.6 Zeichenfläche/Kontextmenüs

- veränderbare Fenstergröße
- bei leerem Fenster: keine Interaktion möglich (über die Navigationsleiste natürlich schon) - Rechts-/Links-Klick ohne Auswirkungen.
- Fenstergröße und Position(absolut) verändert sich durch neues Zeichnen nicht automatisch. (Ausnahme: falls der alte Sichtbereich nicht mehr im Sichtbereich der neuen Zeichnung liegt, wird die nächstmögliche Ansicht gewählt.)
- Klicks in den leeren Raum haben keine Auswirkungen.
- Kontextmenü (Rechtsklick auf Kanten/Knoten):
  - Knoten:
    - Information:
      - Es öffnet sich ein neues Fenster („Graph Class Information“). Aktuell ausgewählt ist die Graphen-Klasse, über die man Information angeklickt hat. Es ist möglich andere Klassen auszuwählen.
      - Filter: Über ein Filter-Eingabefeld kann man bestimmte Graph-Klassen suchen und auswählen.
      - Von der ausgewählten Klasse werden folgende Dinge gelistet: Probleme („Problem“), die bei dieser Graph-Klasse vorkommen, und jeweils deren zugehörige Komplexität („Complexity“); Superklassen („Superclasses“); Äquivalente Klassen („Equivalent Classes“); Unterklassen („Subclasses“).
      - Über Doppelklick auf eine Klasse in einer der Listen Superklassen, Äquivalente Klassen oder Unterklassen wird die aktuell ausgewählte Klasse neu gesetzt und die angezeigten Informationen entsprechend angepasst.
      - „Class-details“-Button:
        Standard-Browser wird geöffnet.
        Weiterleitung zur http://www.graphclasses.org/ - Detailseite zur aktuell ausgewählten Graph-Klasse.
      - „Inclusion-info“-Button:
        Funktioniert nur, falls eine Klasse in Superklassen, Äquivalenten Klassen oder Unterklassen markiert (ausgewählt) ist.
        Es öffnet sich ein Pop-Up, in dem die Relation zwischen der aktuell ausgewählten Graph-Klasse und der unten markierten Klasse dargestellt wird (Zusammenhang - Mengenbeziehungen).
        Es sind drei Buttons zu finden: „View references“ - Öffnet im Standardbrowser die Referenz-Seite von GraphClasses.org; „Draw“ – Zeichnet alle in der Relation beteiligten Graph-Klassen und deren Zusammenhang;
        „OK“ – Pop-Up schließen. Während der Pop-Up offen ist, wird die restliche Anwendung blockiert.

        - „Draw“-Button:
          Ruft Draw-Pop-Up auf. Gleiches Pop-Up, wie unter Reiter „Graph classes“−„Draw“.
        - „Close“-Button:
          Schließt das „Graph Class Information“-Fenster.
      * „Change Name“:
        Im Kontextmenü erscheint eine weitere Ebene, in der alle äquivalenten Klassen zu der jeweiligen Klasse aufgelistet werden. Durch Anklicken einer dieser Klassen wird der Name des Knoten entsprechend geändert.
  - Kanten:
    * Information:
      Klickt man mit Rechtsklick auf eine Kante und dann auf „Information“, erscheint ein Pop-Up, das die Relation zwischen den beiden verbundenen Knoten zeigt (gleiches Pop-Up bei Knoten/„Information“/„Inclusion info“ - vgl. oben)

- Knoten-Position: Knoten können verschoben werden (halten Linksklick, ziehen). Da die Knoten Nach einem Algorithmus gezeichnet wurden, der hierarchisch alle Ebenen durchgeht, bleiben Kanten, die von dem verschobenen Knoten ausgehen/ankommen in der Ebene darunter/darüber an einem bestimmten Punkt oder Knoten fixiert (feste virtuelle Knoten können nicht verschoben werden).

- Pfeile: „normale“, schwarze Pfeile deuten eine Inklusion in Richtung der Pfeilspitze an. Befindet sich am anderen Ende des Pfeiles noch entgegengesetzt eine graue Pfeilspitze, dann bedeutet es, dass die eine Graph-Klasse nicht notwendigerweise eine Teilmenge der anderen ist - es könnte auch Gleichheit herrschen (Unbekannt/„Datenbank-Lücke“).

3.7 Information Fenster

Das „Information“ Fenster enthält eine Liste aller enthaltenen Graph-Klassen, aus welcher eine Klasse mithilfe der Maus ausgewählt werden kann. Entsprechend der Auswahl wird unterhalb eine Tabelle angezeigt, welche alle Probleme enthält und die zur ausgewählten Klasse zugehörige Komplexität angibt. Im Unteren Bereich werden zusätzlich Listen für Superklassen, Äquivalente Klassen und Subklassen angezeigt, innerhalb dieser drei Listen kann eine weitere Klasse ausgewählt werden. Es ist ein Textfeld vorhanden, welches bei bestehender Internetverbindung eine WebSearch (Packagebeschreibung GUI) ausführt, um die Liste der Graph-Klassen zu filtern. Neben einem Close Button um das Information Fenster zu schließen existieren außerdem ein „Draw“ Button der zum Zeichnen des gewählten Graphen auf dem GraphCanvas dient, ein „Inclusion“ Info Button der, falls eine Graph-Klasse und innerhalb der unteren Listen eine weitere Klasse ausgewählt wurde, ein Fenster mit Informationen zur Inklusion dieser beiden Klassen öffnet, sowie ein „Class details“ Button, welcher einen Hyperlink zum entsprechenden Datenbankeintrag auf www.graphclasses.org/classes/... enthält.
„Inclusion info“ Fenster: Das Fenster enthält Informationen zur gewählten Inklusion in textueller Form, sowie die Buttons „OK“, zum Schließen des Fensters,

„Draw“, zum Zeichnen der Inklusion auf dem GraphCanvas und „View references“,
welche einen Hyperlink zum entsprechenden Datenbankeintrag auf http://www.graphclasses.org/classes/...
enthält.

4 Wichtige Klassen des bestehenden Systems

4.1 ISGCI-Package JGraphT

Allgemein:
JGraphT wird benötigt, um die Graph-Struktur zu verwalten und Funktionen darauf auszuführen. Der Schwerpunkt liegt auf Funktionen zur logischen Reduzierung des Graphen und Filterung bestimmter Zusammenhänge (Nachbarn bestimmen etc.). Daneben werden noch nützliche Algorithmen zur Verarbeitung von Graphen bereitgestellt (z.B. Walkers).

Annotation.java
Zuständig für Informationen(Kommentare/Anmerkungen), die an Knoten oder Kanten angehängt werden (Benutzt z.B. von Walkern zur Markierung bereits besuchter Knoten).
Verschiedene Funktionen ermöglichen in einem Annotation-Objekt bestimmten Knoten/Kanten Informationen (Data) zuzuordnen und auszulesen.

AsWeightedDirectedGraph.java
Erweitert AsWeightedGraph und implementiert DirectedGraph
Notwendig zur gewichteten und gerichteten Darstellung des Graphen.
Mithilfe dieser Klasse können Algorithmen, die für ungewichtete Graphen ausgelegt sind, auch auf gewichtete angewendet werden (Mapping von Kanten und Gewichten).

BFSWalker.java
Erweiterung von GraphWalker
Beinhaltet BFS Algorithmus(Breitensuche). Durchläuft einen Graphen, ausgehend von einem gegebenen Startknoten (benutzt nur Kanten).
Umsetzung: Alle Knoten kommen in eine Warteschlange, werden nach und nach (via. BFS) besucht und aus der Queue gelöscht.

DFSWalker.java
Erweiterung von GraphWalker
Beinhaltet DFS-Algorithmus(Tiefensuche). Durchläuft alle Knoten eines Graphen, ausgehend von einem gegebenen Startknoten.
Umsetzung: Über ein Annotation-Objekt können besuchte Knoten markiert werden. Durchlauf mittels DFS-Algorithmus.

GraphWalker.java
Abstrakte Klasse
GraphWalkers werden benötigt, um einen Graphen zu durchlaufen – vgl. BFSWalker/DFSWalker.

CacheGraph.java
Erweiterung von ListenableDirectedGraph
Ermöglicht die Erstellung eines gerichteten Graphen mit Caching von Kanten und Knoten.
Umsetzung: HashMaps jeweils für Knoten und Kanten.
Überschreibt verschiedene Methoden/stellt zur Verfügung: Suchen von Knoten/Kanten, Feststellung ob Knoten/Kanten vorhanden sind und Konsistenz-Checks.

ClosingDFS.java
Erweiterung von DFSWalker
Definition von Closed Graph: „In a directed graph G = (V, A), a set S of vertices is said to be closed if every successor of every vertex in S is also in S. Equivalently, S is closed if it has no outgoing edge".
Beinhaltet DFS Algorithmus zur rekursiven Bestimmung der transitiven Abgeschlossenheit eines Graphen (Menge an Knoten/Kanten, so dass der Graph abgeschlossen ist).

Deducer.java
Stellt einen Algorithmus zur Entfernung aller trivialen Inklusionen in einem Graphen zur Verfügung.
Beinhaltet verschiedene Funktionen: Statistik − Auskunft über Relationen, Inklusionen und deren Beschaffenheiten; Suche und Entfernung trivialer Inklusionen (Umformungen, teilweise Umstrukturierung des Graphen)

GAlg.java
Beinhaltet verschiedene Algorithmen und andere Funktionen für Graphen.
Funktionen: Aufteilen des Graphen in Zusammenhangskomponenten; Selektion aller Nachbarn eines Knoten; Bestimmung eines Pfades zwischen zwei Knoten; Angabe der topologischen Ordnung eines Graphen(falls möglich); Transitive Reduzierung des Graphen.

Inclusion.java
extends org.jgrapht.graph.DefaultEdge implements Relation
Diese Klasse hält die Informationen einer Kante, nämlich Super- und Subklasse und ob es sich um eine unklare Inklusion handelt oder nicht.

ISGCIVertexFactory.java
implements VertexFactory<Set<GraphClass> >
Klasse, um virtuelle Knoten zu erstellen.

RevBFSWalker.java
extends BFSWalker<V,E>
Durchläuft einen Graphen mittels gerichteter Breitensuche.

TreeBFSWalker.java
extends UBFSWalker<V,E>
Durchläuft den Spannbaum eines Graphen mittels ungerichteter Breitensuche. Anmerkung: Markiert nicht selbstständig den Spannbaum.

TreeDFSWalker.java
extends UDFSWalker<V,E>
Durchläuft die Knoten des Spannbaums eines Graphen mittels ungerichteter Tiefensuche.

UBFSWalker.java
extends BFSWalker<V,E>
Durchläuft den Graphen mittels ungerichteter Breitensuche.

UDFSWalker.java
extends DFSWalker<V,E>
Ungerichtete Tiefensuche wird verwendet, um den Graphen zu durchlaufen.

WalkerInfo.java

Diese Klasse wird benutzt um Knoten Informationen zu geben, während der Graph mittels Tiefen- oder Breitensuche durchlaufen wird.

4.2 ISGCI-Package Layout

Alle Klassen, die in dem `./layout`-Package stehen, sind für das Layout des letztendlich zu zeichnenden Graphen zuständig. Dabei stehen diese stark mit den Klassen aus dem `./grapht`-Package in Zusammenhang. Die Klassen und deren Methoden werden über die Klassen des `./gui`-Packages aufgerufen.

Im Folgenden werden alle Klassen aus dem `./layout`-Package grob beschrieben und deren Funktionweise und Zuständigkeit näher erläutert. Konkrete Details, wie Methodendeklarationen, Methodenaufrufe und Variablen werden spärlich erläutert. Dies sollte der Verständlichkeit nicht im Wege stehen, ferner sogar dienen.

GraphDrawInfo.java
Erweiterung von WalkerInfo.java
Diese Klasse erweitert die Klasse `WalkerInfo.java` aus dem Package `./grapht`, welche dafür zuständig ist, ob Kanten zu einem zu berechnenden/anzuzeigenden Graphen gehört oder nicht. Bei Knoten wird auf Adjazenz geprüft (näheres in der Beschreibung bei `grapht`), ob der Knoten zum dem zu zeichnenden Graphen gehört. Sie fügt weitere Variablen, die

Werte für Knoten und Kanten beinhalten, welche zur Graphzeichnung benötigt werden, ein. Dazu gehören auch virtuelle Knoten, die keinen Inhalt haben und nur zur Kantenzeichnung benötigt werden. Außerdem werden Variablen zur Ranking-Berechnung deklariert, um die spätere richtige Position in der Zeichnung zu bestimmen. Alle benötigten Variablen werden im Konstruktor mit Default-Werten instantiiert und überschrieben, sobald diese benötigt werden. (i.e. Breite des Knotens, Koordinaten, Position im Ranking usw.)

HierachyLayout.java

Diese Klasse ist zur Berechnung der Hierarchie im Layout zuständig. Jedoch muss berücksichtigt werden, dass eine konsistente Hierarchie nur erreicht werden kann, wenn der Graph, zu dem die Hierarchie berechnet werden soll, ein gerichteter nicht-zyklischer Graph ist. Es muss beachtet werden, dass dieser transitiv reduziert ist. Erneut spielt die Berechnung des Rankings und der virtuellen Knoten eine entscheidende Rolle, wobei diese Berechnungen mit JGraphT stattfinden. Dazu gehört die Beschreibung des zum Knoten gehörenden Vertex (Weite, Rank..). Um das Layout der Hierarchie zu berechnen, wird die eben beschrieben Klasse `GraphDrawInfo.java` verwendet. Bei der Berechnung werden dabei die 4 Prinzipien berücksichtigt, die in der Dokumentation zur Zeichnung (`./doc/1993drawing.pdf`) näher erläutert und bei den 3 Durchläufen des „Network Simplex Algorithm“ beachtet werden. Der konkrete Ablauf, welche Schritte getan werden müssen, werden dort näher erläutert. (Machbarkeitsbaumberechnung mit entsprechendem Ranking der den „length-constraints“ genügen muss). Es werden verschiedene Algorithmen ausgeführt, welche mit einigen Hilfsmethoden u.a. Spannbäume mit festen Kanten, machbare Rankings berechnet [...]. Hierbei muss flüchtig erwähnt werden, dass die Berechnung in den Grundzügen auf Ersetzung von Graphkanten und -knoten durch Nichtgraphkanten und -knoten, um die Machbarkeit mit einem optimalem Ranking zu erreichen, basiert. Es wird versucht ein minimalen Graphen zu finden (optimales Ranking), der den 4 Prinzipien genügt. Dabei werden die Variablen der Klasse `GraphDrawInfo.java` benötigt.

LLWalker.java

Erweiterung von TreeDFSWalker.java

Traversiert die Knoten eines Baumes mithilfe von Tiefensuche, indem die Methoden der Oberklasse verwendet werden.

Ranks.java

In dieser Klasse werden die Knoten eines Graphen über Rankings verwaltet. Einer der Hauptaspekte für die Verwendung von Rankings ist, dass die Kantenkreuzungen soweit wie möglich minimiert werden. Ein initiales Ranking wird durch Zuweisung eines Rankings für jeden Knoten über Breitensuche (durch Traversierung), wobei bei dem niedrigsten Ranking begonnen wird (in der Anwendung von oben nach unten), erstellt. Das ist notwendig, um sicherzugehen, dass dieses Ranking ein Baum ohne Kantenkreuzungen(möglichst) ist. Um dies zu erreichen wird `HierarchyLayout.java` verwendet, indem über sie virtuelle Knoten erstellt und gesetzt werden, um ein optimales Ranking für die Kantenkreuzungen zu erreichen. Um die Knoten innerhalb eines Rankings effektiv zu sortieren werden Heuristiken (siehe `./doc/1993drawing.pdf`) verwendet (z.B. Median, Barycenter, Minimum, Sifting..). Es werden iterative heuristische Berechnungen entsprechend der bereits genannten Dokumentation ausgeführt, um ein bestmögliches in einem akzeptablen Rahmen des Berechnung-Aufwandes Ergebnis zu erzielen (Vertauschung von Knoten, falls dadurch Kreuzungen vermieden werden).

TightTreeWalker.java
Erweiterung von UBFSWalker.java;
wird in `HierarchyLayout.java` *für den „Network Simplex Algorithm“ verwendet*
Diese Klasse wird benötigt um einen gerichteten Graphen mithilfe von Breitensuche *ungerichtet* zu traversieren, dabei werden nur Kanten, die fest(„tight“$\hat{=}$ slack = 0) sind, berücksichtigt. Es wird ein Startknoten definiert, mit dem diese Traversierung ausgeführt wird. Die zu traversierenden Kanten sind als Baumkanten markiert (Information aus `GraphDrawInfo.java`). Als Ergebnis erhält man einen Spannbaum für den Ausgabe-Graphen.

TreeReranker.java
Erweiterung von TreeBFSWalker
Mithilfe dieser Klasse werden neue Rankings auf einem Layout-Graphen (als Baum übergeben) erstellt, um ein optimales Ranking zu erhalten (durch Breitensuche). Sie wird eigentlich nur verwendet, um eine semantische Trennung zum `TreeBFSWalker.java` zu erhalten, denn jede verwendete Methode ruft eine entsprechende Methode aus der Oberklasse auf.

4.3 ISGCI-Package db

Hier liegen viele Algorithmen; z.B. beinhaltet die Klasse Algo.java Algorithmen, um Subklassen/Superklassen eines Knoten zu finden und diese auszugeben. Diese werden benötigt, um die Inklusionsinformationen zu speichern und abzurufen.

4.4 ISGCI-Package gc

„GC“ steht für GraphClass. In diesem Package sind alle „GraphClass“ Klassen enthalten. Jede enthaltene Java Klasse beschreibt eine GraphClass und deren besondere Eigenschaften.

4.5 ISGCI-Package iq

Hier finden ISGCI Queries statt. „db“ und „gc“ nutzen dieses Package, um Queries auszuführen.

4.6 ISGCI-Package problem

„Store complexity information for graph problems“, in diesem Package sind alle Funktionen enthalten, die dazu genutzt werden, ein Problem zu beschreiben.

4.7 ISGCI-Package util

In dem „util“ Package liegen alle für spezielle Funktionalitäten benötigte Klassen. Dazu gehört eine Reihe von Klassen, die die Verwendung von „LaTeX“ innerhalb des ISGCI ermöglicht (d.h. das Verwenden einiger LaTeX-Symbole).

4.8 ISGCI-Package xml

Das Package „xml“ ist zum Input/Output von XML Dateien; hier werden Informationen aus XML Dateien gelesen und ausgegeben. Das Package „xml“ greift dabei auf das Package „sax“ zu und nutzt deren Filterfunktionalität. Die Daten aus einem XML input/output werden den Klassen „Grapht“, „ISG“ und „Util“ zur Verfügung gestellt.

4.9 ISGCI-Package isq

In „isg“ werden die Graph Familien, Zusammenhänge aus einer XML gelesen und zum Zeichnen vorbereitet, also an JGraphT weitergereicht. Dabei nutzt „isg“ Funktionen des „util“ Packages.

4.10 ISGCI-Package sax

Das Package „sax“ beinhaltet nur die Klasse XMLWriter.java. Diese ist ein Filter, um ein XML Dokument aus einem SAX (Simple API for XML) event stream zu erzeugen. Diese Klasse wird vom XML Parser benötigt.

4.11 ISGCI-Package gui

AboutDialog
Zeigt den „About“−Dialog an, der erscheint, wenn man unter „Help“ auf „About“ klickt.

CheckInclusionDialog
Zeigt den Dialog an, bei dem man zwei Klassen selektieren kann und prüfen kann, ob eine Relation besteht. Dabei kann nach Namen gefiltert werden.

EdgePopup
Erstellt das Popup Menu, auf das man zugreifen kann, wenn man Rechtsklick auf eine Edge(View) macht. Dabei wird Information angezeigt.

EdgeView
Erzeugt einen View von einer Edge. Zeichnet die Edge und sorgt dafür, dass Pfeilspitzen angezeigt werden.

ExportDialog
Eine Klasse, die einen Export Dialog erzeugt, in dem der Nutzer „Fileformat“, „postscrip options“ und „GraphML“ Optionen auswählen kann. Es kann die Zieldatei ausgewählt werden. Bei Bestätigung wird die Datei mit den Einstellungen erzeugt. Andernfalls wird ein Error erzeugt.

GraphCanvas
Das Display auf dem der Graph gezeichnet wird. Dabei kann der Canvas gelöscht werden und dem Canvas kann ein übergebener Graph hinzugefügt werden.

GraphClassInformationDialog
Zeigt den Dialog „Graph Class Information“ an. Dabei werden von der selektierten „GraphClass“ die Probleme, deren Komplexität, Superklassen, Äquivalentklassen und Subklassen angezeigt.

GraphClassSelectionDialog
Zeigt den Dialog „Graph Classes Selection“ an, der erscheint, wenn man bei „Graph Class Information“ auf „Draw“ klickt. Bei Doppelklick wird der Canvas mit der Klasse, die markiert wurde, neu gezeichnet. Kann eine Collection der Classes zurückgeben, die der Benutzer selektiert.

GraphView
Zeigt den Graphen auf dem Canvas an. Es wird eine neue „GraphView“ mit den Knoten(Nodes) aus „DataSet.inclGraph“ gezeichnet. Dabei werden die Nodes und der GraphCanvas, der den Graphen anzeigt, übergeben. Kann überprüfen, ob ein Knoten virtuell ist oder nicht. Im „layout“ werden „EdgeView“s erstellt und an die richtige Position bewegt. Dabei werden jeweils immer die Bounds neu berechnet.

InclusionResultDialog
Zeigt nach Klick auf „Find Relation“ an, ob eine Relation zwischen zwei Graph Klassen besteht. Wenn nein, werden die minimal ähnlichen Superklassen und die maximal ähnlichen Subklassen angezeigt. Gibt auch an, warum keine Relation existiert.

IQDialog
In dieser Klasse sind die Dialogfenster für die drei Menüpunkte im Menü „Graphclasses“, mit denen man auswählen kann, welche Graphen man sich zeichnen lassen möchte.

ISGCIGraphCanvas
Klasse für den Canvas („Leinwand“, sprich der weiße Bereich in der Mitte) auf der der Graph gezeichnet wird. Bekommt einen Graphen der Klasse „SimpleDirectedGraph<Set<GraphClass>,DefaultEdge>“ übergeben und zeichnet diesen. Einfärbung ist ebenfalls in dieser Klasse implementiert.

ISGCIMainFrame
Der „Main Frame“ der Anwendung. Besteht aus der „JMenuBar“, bekommt den Canvas übergeben und fügt Scrollbars, sämtliche Buttons und Event Listeners hinzu.

LatexGraphics
Diese Klasse ist zuständig für die Zeichnung von Latex Strings, die in der Bezeichnung einiger Graphen in der Datenbank enthalten ist.

LatexLabel
In dieser Klasse werden zusammen mit den LatexGraphics die gesamten Labels gezeichnet, die in einen Graphen hinein kommen.

ListGroup
Kommentar:„A group of lists in which only one item can be selected.“
Wird für einen gezeichneten Graphen benutzt, damit man nur einen Knoten anwählen kann und es somit keine Probleme für die Event Handler gibt.

MessageDialog
Einfache Klasse für Ja/Nein Dialogfenster, die öfter vorkommen.

NamingDialog
Der „Naming“ Dialog im Menü unter „View → Naming preference“. Damit kann man einstellen, nach welchen Kriterien die Knoten benannt werden sollen.

NodeList
„JList“ für Knoten mit Standard Listenmethoden, die in HTML ausgegeben werden können.

NodePopup
Klasse des Popup Menüs, das bei Rechtsklick auf einen angewählten Knoten erscheint. Hat Buttons für die Funktionen „Change Name“ und „Information“

NodeView
Klasse der gezeichneten Knoten-Objekte inklusive Label, die sich bewegen lassen.

NulGraphics
Eine Klasse, die zur Evaluierung der Größe dargestellter Strings dient. Die Klasse erbt von Graphics, implementiert jedoch nur die Methoden, welche zur Schriftverwaltung dienen.

OpenProblemDialog
Eine Klasse, die einen Benutzer Dialog erzeugt, in welchem eine Auswahl der Graph-Klassen, bezüglich des gewählten Problems, zur Verfügung gestellt wird. Die Graph-Klassen sind unterteilt in drei „JLists“ für „Minimal (co)NP-vollständig“, „offen“ oder „Maximal P“. Der Benutzer kann innerhalb dieser Listen Graphklassen auswählen und über Buttons entweder die gewählten Klassen zeichnen oder weitere Informationen erhalten. Dazu wird mithilfe des „ActionListeners“ entweder ein „GraphClassInformationDialog“ zur gewählten Graph-Klasse geöffnet oder mithilfe des „GraphCanvas“ die Auswahl gezeichnet.

ProblemsMenu
Stellt das unter „Problem > Colour for Problem“ dargestellte Menu mit Radio-Buttons für jedes Graphproblem zur Verfügung. Die Liste der Probleme wird über teo.isgci.db.DataSet zur Verfügung gestellt.

PSFontMetrics
Diese Klasse dient zum Verwalten der genutzten Schriftarten Helvetica und ISGCIFont.

PSGraphics
Eine Klasse, welche die grafische Verarbeitung für Postscript verwaltet. Methoden, welche nicht für die Nutzung in ISGCI relevant sind, erzeugen eine „RuntimeException“. Es sind Methoden zum Zeichnen von Elementen (Nodes, Arrows) und zum Schreiben von Strings enthalten. Außerdem Verwaltungsmethoden zur Steuerung der Postscript Befehle.

ScaleMenu
Auskommentierte Klasse ohne relevanten Inhalt. Sollte voraussichtlich zur Skalierung der Zeichnung verwendet werden.

SColor
Eine Klasse, welche mithilfe der Methode „brighter(Color c)“ eine, um einen festen Faktor hellere, Version der Farbe „c“ erzeugt und zurückgibt. Außerdem ist eine Methode enthalten, welche überprüft, ob eine Farbe eine Graustufe ist und eine weitere, um diese Graustufe zu erhalten.

SearchDialog
Stellt den Dialog „Search for a graphclass“ unter „View > Search“ in „Drawing“ zur Verfügung. Innerhalb des Konstruktors wird durch Abfrage des „GraphCanvas“ eine Liste aller gezeichneten Knoten erzeugt und als JList im Dialog dargestellt. Der Nutzer kann innerhalb dieser Liste eine Graph-Klasse auswählen und mithilfe des „Search“ Buttons innerhalb der Zeichnung nach dieser Suchen. Dazu werden im „GraphCanvas“ die Methoden „findNode(...)“, „markOnly(...)“ und „centerNode(...)“ genutzt, welche die gewählte Graph-Klasse markieren und im Canvas zentriert darstellen.

SmartGraphics
Eine abstrakte Klasse, welche von Graphics erbt und Methoden zum Zeichnen von Graphen enthält. (Zeichnen von Nodes: „drawNode(...)“ und Zeichnen von Pfeilen:„ drawArrow(...)“)

SVGGraphics
Klasse, die von SmartGraphics erbt und zum Erzeugen von SVG Grafiken zuständig ist. Hierzu implementieren die, von SmartGraphics geerbten Methoden, Funktionen zur Erzeugung einer SVG Datei.

View
Ein Interface, welches Methoden zur Implementierung von Objekten, welche auf einem „GraphCanvas“ gezeichnet werden können, zur Verfügung stellt.

VirtualNodeView
Klasse, welche von NodeView erbt und zum Erzeugen eines leeren Knotens dient. So kann innerhalb einer Ebene ein leerer Knoten erzeugt werden, um die hierarchische Darstellung beizubehalten.

WebSearch
Eine Klasse, welche von „JTextField“ erbt und ein Textfeld zur Suche innerhalb der „ISGCI Online Datenbank“ dient. Hierzu wird eine Verbindung zur Webseite hergestellt und die Suche über den Browser realisiert, indem der eingegebene Suchbegriff als Parameter in die URL hinzugefügt wird. Die dabei entstehenden Suchergebnisse werden zeilenweise eingelesen und der Ergebnisliste hinzugefügt. Anschließend wird aus „DataSet“ die entstandene Liste als Liste von auswählbaren Elementen umgewandelt und in der Liste der Graph-Klassen angezeigt.

5 Vorgeschlagene Software Architektur

5.1 Übersicht

Um neue Funktionalitäten von JGraphX hinzufügen zu können, werden grundlegende Änderungen am ISGCI vorgenommen. Dies umfasst das Einführen einer neuen Struktur, in der die Graphinformationen (Knoten (Graphklassen) und Kanten (Inklusionen)) gespeichert werden, Veränderungen des Canvas, auf dem der Graph gezeichnet wird und Veränderungen am GUI, um die neuen Funktionen für den User sichtbar zu machen.

5.2 ISGCIGraph.java

Wir erstellen eine neue Klasse ISGCIGraph im neuen Package .teo.isgci.jgraphx, welche die Klasse mxGraph(JGraphX) erweitert und als Schnittstelle zwischen ISGCI und JgraphX fungiert.
Zusätzlich erhält dieses Package eine weitere neue Klasse „ISGCIGraphWrapper“, die das Interface „DirectedGraph“(JGraphT) implementiert und auf den „ISGCIGraph“ zugreift, den sie als Instanzvariable hält. Dieser „Wrapper“ ist die Schnittstelle zwischen JGraphX und JGraphT. Dadurch bleibt die Schnittstelle JGraphT−ISGCI unangetastet. Die Erweiterung der Klasse „mxGraph“ ermöglicht uns die Benutzung der Layout-Funktionen der JGraphX-Bibliothek. Für das Layout wird die Klasse „mxHierarchicalLayout.java“ verwendet, die das Aussehen des Graphen(über Ranking) verbessert. Durch die Zusammenführung des „DirectedGraph“(JGraphT) und „mxGraph“(JGraphX) sorgen wir dafür, dass keine doppelte Datenhaltung vorliegt, denn alle Knoten und Kanten werden in JGraphX-Cells abgelegt. Sie sind weiterhin wie in einem „DirectedGraph“ erreichbar. Bei der Implementierung ist besonders darauf zu achten, dass der Zugriff auf Kanten und Knoten des Graphen effizient umgesetzt wird, damit „Userexperience“ und „Usability“ nicht unter langen Wartezeiten leidet.

Durch die Implementierung des „DirectedGraph“-Interfaces und setzen der Instanzvariablen `protected boolean multigraph = false` und `allowLoops = false` des „mxGraph“s können wir den neuen „ISGCIGraph“ wie den „SimpleDirectedGraph“ der bisherigen Implementation verwenden, da sie die selben Eigenschaften (gerichtet, keine Kreise und keine Multikanten) haben. Diese Funktionen beinhalten unter anderem:

- Erstellen des Graphen aus der heruntergeladenen „ISGCI.xml“
- Mittels Walkern, aus dem „.teo.jgrapht“-Package, den Graphen traversieren

- Suche nach Problemklassen mittels des „.teo.problem“-Packages
- Export des Graphen
- Zeichnen des Graphen

Dazu müssen folgende Klassen angepasst werden, die einen „SimpleDirectedGraph“ verwenden und jetzt einen gleichwertigen „ISGCIGraphWrapper“ verwenden werden:

FindISG.java

Algo.java

DataSet.java

ForbiddenClass.java

BFSWalker.java

Deducer.java

DFSWalker.java

GAlg.java

GraphCanvas.java (hier finden noch weitere, grundlegende Änderungen statt, s.u.)

GraphView.java (hier finden noch weitere, grundl. Änderungen statt, s.u.)

ISGCIGraphCanvas.java (hier finden noch weitere, grundl. Änderungen statt, s.u.)

ISGCIMainFrame.java (hier finden noch weitere, grundl. Änderungen statt, s.u.)

LandMarks.java

ISGCIWriter.java

Durch das Erweitern des „mxGraph“ erhalten wir zudem folgende Funktionalität:

- Automatisches Layout mittels „mxGraph.layout.mxHierarchyLayout;“ dies ersetzt das „teo.layout“−Package der bisherigen Implementation.
- Zeichnen des Graphen mittels JGraphX-Funktionen auf eine „mxICanvas“
- Effizienter Zugriff auf alle Elemente des Graphen
- Verschieben von Knoten

Die JGraphX-Cells (mxGraph.model.mxCell) werden in einer Klasse ISGCImxCell im teo.isgci.jgraphx-Package so erweitert, dass sie die „GraphClass“ und „Inclusion“ Information von ISGCI speichern können und für alle Klassen, die den ISGCIGraphWrapper benutzen, als „GraphClass“ oder „Inclusion“ erscheinen.

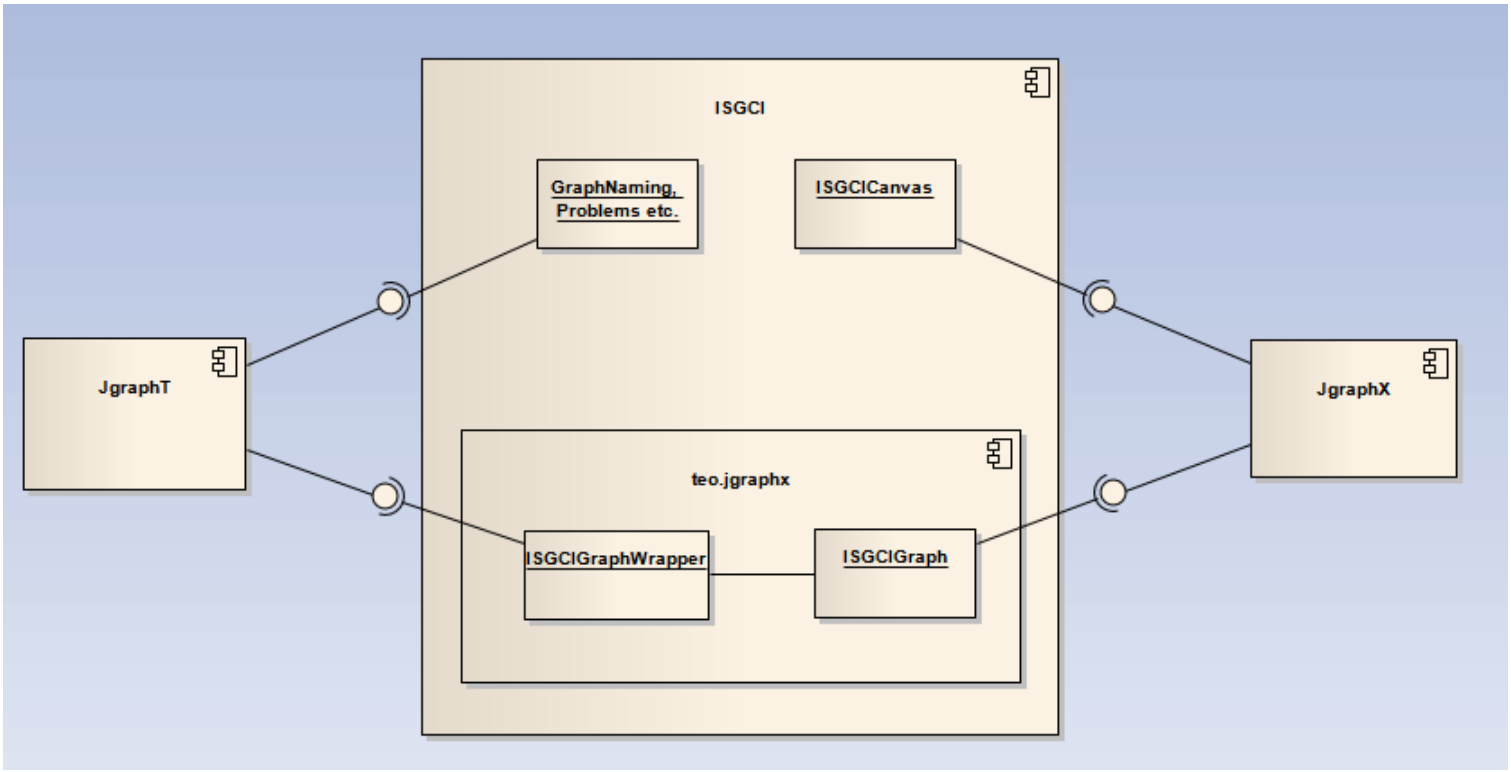


Abbildung 1: Aufbau des neuen Systems

5.3 ISGCI.Gui

Grundlegende Änderungen Die bereits existierende Klasse ISGCIGraphCanvas.java aus dem ISGCI Projekt wird so erweitert, dass sie das Interface mxICanvas.java implementiert, sodass dem aktuellen ISGCIGraphCanvas die Funktionalitäten hinzugefügt werden, die der ISGCIGraph (abgeleitet von mxGraph) benötigt um den Graphen in der Darstellung, wie JGraphX einen Graphen interpretiert, zu zeichnen. Dadurch entfallen einige Klassen aus dem GUI-Package (z.B. NodeView, EdgeView), die momentan zur Speicherung der Daten, wie ein Knoten oder eine Kante gezeichnet wird, beinhalten. Diese werden beim Erstellen des ISGCIGraph (extends mxGraph) entsprechend in der JGraphX-Darstellung gespeichert (Mapping von NodeView und EdgeView auf mxCellState). Die „isProper“ Information des EdgeView können wir hierbei vernachlässigen, da sie schon in den „Inclusions“ gespeichert ist, zum Zeichnen wird hierauf zurückgegriffen. JGraphX übernimmt mit dem ISGCIGraph, den angehängten mxCells, mxCellStates und der ISGCICanvas selbstständig das Zeichnen des Graphen. Die setProblem(Problem p) Methode in der ISGCIGraphCanvas wird angepasst um die Farben in den mxCellStates zu ändern.

Knoten Verschieben

Wird wie bisher in der ISGCIGraphCanvas.java Klasse realisiert.

Zoom (Mausrad)

Wird über Veränderung der „Scale“-Variable im ISGCIGraph assoziierten GraphView und auswerten eines MouseWheelListeners gelöst.

Zoom (Menu-Bar)

In der ISGCIMainFrame.java Klasse werden die Menüs und dazugehörigen Items erzeugt. Hier werden wir die Methoden, `createMenus()` und `registerListeners()` anpassen. Anschließend wird die interaktive Funktionalität der neu hinzugefügten Komponenten durch eine Anpassung der `actionPerformed(ActionEvent event)` gewährleistet. Hier wird auch die „Scale“-Variable im ISGCIGraph assoziierten GraphView verändert.

Tooltip

Wird in der ISGCIGraphCanvas.java realisiert. Es wird über den MouseListener implementiert, der falls er über dem Knoten verweilt den Tooltip, dessen Form in der neuen Klasse ISGCINodeTooltip beschrieben wird.

Tooltip Inhalt

Der ISGCINodeTooltip greift auf Daten in der Klasse DataSet.java aus dem Package teo.isgci.db zu. Er enthält alle alternativen Namen des Nodes (der aktuelle Name inbegriffen) in Listenform.

Scrolling

In der ISGCIMainFrame.java Klasse existiert bereits eine Methode `createCanvasPanel()`, welche den Canvas und die Scrollbalken unten und rechts des Canvas erzeugt. Um sich darin auch per click and pull bewegen zu können erstellen wir eine weitere Methode in dieser Klasse, die das MousePressed Event auswertet, wenn es nicht auf einem Knoten liegt und den Viewport anpasst.

Kontextmenü

NodePopup.java ist eine bereits existierende Klasse die das Kontextmenü bereitstellt, das bei einem Rechtsklick, auf einen Knoten, aufgerufen wird. Dieses wird von uns um die folgenden Punkte erweitert.

Kontextmenü Superklassen

In NodePopup.java werden zwei neue Items („Show subclasses“ und „Hide subclasses“) durch anpassen des Konstruktors der NodePopup-Klasse implementiert und die dazugehörige Funktionalität durch anpassen der `actionPerformed(ActionEvent event)` Methode aufgerufen. `actionPerformed(ActionEvent event)` ruft dabei Informationen über den angewählten Knoten und dessen Superklassen ab und zeichnet das Ergebnis (die Knoten) auf den Canvas, hierfür erstellen wir eine Methode `private Object[] getSupernodes(Object cell)`. Dafür

rufen wir `mxGraph.getAllEdges(Object cell)` mit selbigem Knoten auf, suchen aus dem Ergebnis alle Vorgängerknoten heraus (rekursiv für Kanten, die eine Kante als Vorgänger haben) und wenden auf diese `mxGraph.toggleCells( boolean show, Object[] cells)` an.

Kontextmenü Subklassen

In NodePopup.java werden zwei neue Items („Show superclasses“, „Hide superclasses“) durch anpassen des Konstruktors der NodePopup-Klasse implementiert und die dazugehörige Funktionalität durch anpassen der `actionPerformed (ActionEvent event)` Methode aufgerufen. `actionPerformed(Action Event event)` ruft dabei Informationen über den angewählten Knoten und dessen Subklassen ab und zeichnet das Ergebnis (die Knoten) auf den Canvas, hierfür erstellen wir eine Methode `private Object[] getSubnodes(Object cell)`. Dafür rufen wir `mxGraph.getAllEdges(Object cell)` mit selbigem Knoten auf, suchen aus dem Ergebnis alle Nachfolgerknoten heraus (rekursiv für Kanten, die eine Kante als Nachfolgerknoten haben) und wenden auf diese `mxGraph.toggleCells( boolean show, Object[] cells)` an.

Kontextmenü Nachbarn

In NodePopup.java werden zwei neue Items („Show neighbours“, „Hide neighbours“) durch anpassen des Konstruktors der NodePopup-Klasse implementiert und die dazugehörige Funktionalität durch anpassen der `actionPerformed(Action Event event)` Methode aufgerufen. `actionPerformed(ActionEvent event)` ruft dabei Informationen über den angewählten Knoten und dessen Nachbarn ab und zeichnet das Ergebnis (die Knoten) auf den Canvas, hierfür erstellen wir eine Methode `private Object[] getNeighbour(Object cell)`. Dafür rufen wir `mxGraph.getAllEdges(Object cell)` mit selbigem Knoten auf, und wenden auf diese `mxGraph.toggleCells( boolean show, Object[] cells)` an.

Topbar In der Topbar wird der neue Menüpunkt „Selection“ eingeführt der die Funktionen „Show neighbours“, „Hide neighbours“, „Show superclasses“, „Hide superclasses“, „Show subclasses“ und „Hide subclasses“ bereitstellt, die die gleiche Funktion haben wie im Kontextmenü auf einem Knoten. Dafür wird in der ISGCIMainFrame−Klasse die `protected JMenuBar createMenus()` und die `actionPerformed(ActionEvent event)` angepasst.

6 Hardware/Software Mapping

6.1 Architektur

ISGCI benötigt als Hard-/Software Umgebung eine lauffähige Java-Laufzeit-Umgebung. Für weitere Informationen zu den Hard-/Software-Anforderungen besuchen Sie bitte Java 6 - Anforderungen (06.06.2013).
Zusätzlich wird für die Filterfunktion eine ständige Internetverbindung vorausgesetzt.

6.2 Management der persistenten Daten

Alle zu speichernde Daten werden in der Datenbank online gespeichert ,werden beim Start der Anwendung heruntergeladen und liegen im XML-Format vor.

6.3 Randbedingungen

Starten der Anwendung Wenn keine Internetverbindung: Fenster "Anwendungsfehler: Anwendung kann nicht gestartet werden." dies tritt nicht immer auf; das Verhalten ist nicht spezifiziert.

Beenden der Anwendung Es wird kein Zustand gespeichert, die Anwendung beendet sich sofort.

7 Testing

Es wird ein „Grey Box testing" Ansatz verwendet, d.h. der Testende hat gegebenenfalls Zugriff auf den Sourcecode, kennt allerdings nicht unbedingt die vollständige Implementierung der Funktion. Dabei wird gegen die Beschreibung des bereits bestehenden Systems (siehe: Abschnitt 3) getestet.
Wir gliedern das Testing in folgende Testphasen:

- Unit Testing
- Integration Testing (Modul Testing)
- Software Testing
- Functionality Testing
- Acceptance Testing

Diese sind folgendermaßen aufgebaut:

Unit Testing
Für jede angepasste bzw. selbst geschriebene Klasse wird bereits während deren Implementation ein JUnit Testfile entworfen. Dieses beinhaltet Tests zu jeder Methode, welche mit fiktiven Parametern aufgerufen werden, um deren korrekte Ausführung zu testen. Die Testfiles werden jeweils nicht von den Programmierern der Methoden bzw. Klassen geschrieben, sondern von einem anderen Teammitglied. Das Feedback hat Änderungen am Programmcode zur Folge, falls Abweichungen auftreten.

Integration Testing
Sind alle JUnit Tests erfolgreich durchgeführt worden, werden die einzelne Module (Packages) auf ihre Funktionalität, alleine und in Kombination, getestet. Hierbei wird das angehängte Testprotokoll (siehe: Unterabschnitt 8.1) verwendet. Beim Test eines Moduls werden benötigte Parameter, die über Schnittstellen zur Verfügung gestellt werden sollten, durch sinnvolle fiktive Eingabewerte ersetzt. Ist ein Modul auf diese Weise erfolgreich getestet worden, so werden inkrementell benötigte andere Module, sofern diese auch erfolgreich getestet wurden, hinzugefügt. Die zusammengesetzten Module werden nach dem selben Prinzip auf ihre Funktionalität überprüft, bis dabei das gesamte System zusammengefügt wurde. Das Feedback hat Änderungen am Programmcode zur Folge, falls Abweichungen auftreten.

Software Testing
Hierbei wird der zur Verfügung stehende Prototyp auf die Umsetzung der funktionalen Anforderungen überprüft. Dazu werden spezifische Testszenarien (Template siehe: Unterabschnitt 8.2) festgelegt, die Benutzerinteraktion darstellen. Die Testszenarien werden jeweils von einem Teammitglied durchgeführt und das Ergebnis wird im Template protokolliert. Das Feedback hat Änderungen am Softwaredesign zur Folge, falls Abweichungen auftreten.

Functionality Testing
Für die Beta Version und den Release Candidate wird ein Functionality Testing durchgeführt. Dabei wird das System auf einem Endbenutzergerät ausgeführt, auf dem bisher keinerlei Entwicklung stattgefunden hat, wobei ein reales Nutzungsszenario simuliert wird, der Tester versucht das System zum Zusammenbrechen zu bringen. Dieses wird von der durchführenden Person protokolliert, insbesondere Fehler und der Weg zu ihnen. Es wird das gleiche Template (Unterabschnitt 8.2) wie für Szenarien verwendet. Das Feedback hat Änderungen am Softwaredesign zur Folge, falls Abweichungen auftreten.

Acceptance Testing
Nachdem der Release Candidate fertig gestellt wurde, wird dieser dem Kunden zum Acceptance Testing zur Verfügung gestellt. Das Feedback des Kunden hat Änderungen an der Spezifikation zur Folge, insofern sie im Zeitplan umsetzbar sind. Sind die Änderungen umfangreicher wird der Releasetermin und das Budget neu verhandelt.

8 Anhang

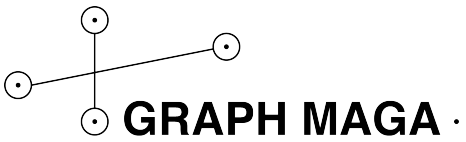
Anbei das Protokoll Layout zur Nutzung während der Modultests und ein Beispielszenario zur Nutzung während dem Software Testing.

8.1 Testprotokoll

8. Juni 2013

Testnummer _____

Testmanager _____



Software Modultest

ISGCI - Information System on Graph Classes and their Inclusions

Getestete Module: _____

Testbeschreibung	*Testergebnis*

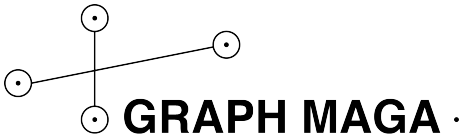
Methode	Parameter	Rückgabewert	Zulässigkeit

8.2 Testszenario

8. Juni 2013

Szenarionummer _____

Testmanager _____



Software Testszenario

ISGCI - Information System on Graph Classes and their Inclusions

Szenariobeschreibung
Auswählen eines Knotens und Einblenden seiner Superklassen mithilfe des Kontextmenüs.

Schrittnummer	**Ereignis/Aktion**
00	Vorbedingung: Ein Graph wurde gezeichnet.
01	Der Benutzer klickt mit der rechten Maustaste auf einen Knoten.
02	Es erscheint ein Kontextmenü.
03	Der Benutzer klickt auf Show Superclasses.

Erwünschte Reaktion
Superklassen zum ausgewählten Knoten sind eingeblendet und Kontextmenü ist wieder geschlossen.

Tatsächliche Reaktion

Wertung