Entwurf
Graph von Ansicht

Nicolas Boltz
uweaw@student.kit.edu

Jonas Fehrenbach
urdtk@student.kit.edu

Sven Kummetz
kummetz.sven@gmail.com

Jonas Meier
Meierjonas96@web.de

Lucas Steinmann
ucemp@student.kit.edu

Inhaltsverzeichnis

1 Einleitung

Das Programm Graph von Ansicht soll, wie bereits im Pflichtenheft erläutert, der Visualisierung von Graphen dienen. Hierzu wurden im besagten Dokument Kriterien erarbeitet, welche zum erreichen dieses Ziels notwendigerweise umgesetzt werden sollen. Die Erarbeitung dieser Pflichtkriterien, in Form eines Entwurfs, ist Ziel dieses Heftes. Im weiteren wird aber auch versucht, die Umsetzung der Wunschkriterien möglichst früh zu beachten und durch einen allgemeineren Entwurf, nach Möglichkeit, nicht zu verhindern.
Zur Umsetzung der genannten Kriterien wurden diese zunächst getrennt voneinander betrachtet, danach die gröberen Designstrukturen, die für diese relevant sind, etabliert. So soll das bestimmte Ziel der Erweiterbarkeit (siehe Pflichtenheft 1.1.4) durch eine Pluginstruktur ermöglicht werden. Hierfür ist es notwendig zu bestimmen welche Bereiche des Programms erweiterbar sein sollen. Das Pflichtkriterium sieht hier vor, dass „Schnittstellen für Plugins in den Bereichen Import, Export, Layoutalgorithmen, Filter für Knoten- und Kantentypen und weitere Operationen auf einzelne Knoten und Kanten“ existieren, welche in Form von Interfaces in dem Paket plugin zur Implementierung zur Verfügung gestellt werden. Auch Teil dieses Pakets ist der PluginManager, welcher die Plugins verwaltet und bei dem sie ihre Funktionalität registrieren, sowie ein Interface Plugin, welches dann erweitert und geladen werden soll.
Die in Input/Output (siehe Pflichtenheft 1.1.2) sowie Steuerung (siehe Pflichtenheft 1.1.4) gestellten Anforderungen legen eine Trennung zwischen Datenmodel und Anzeige nahe, um auf der einen Seite verschiedene Graphtypen zu unterstützen, die auf der anderen Seite wiederum nicht der GUI bekannt sein sollten, da diese auch von Plugins gestellt werden können. Diese Trennung erfolgt in Form der Pakete graphmodel und gui. Das graphmodel beinhaltet grundlegende Interfaces, welche dann von spezifischen Graphen, wie beispielsweise dem in diesem Paket liegenden DefaultGraph, implementiert werden. Das Paket gui wiederum enthält die Anzeigenlogik, sowie die Kontrolle dieser Anzeige und baut stark auf javafx. Insgesamt wird so eine MVC-artige Struktur aufgebaut, was wiederum der Kapselung zugute kommt.
Ein Paket objectproperty dient der weiteren Verknüpfung zwischen Model und View, hier wird ein von javafx bereitgestelltes Visitor Entwurfsmuster verwendet, um die Daten in der Anzeige bei Veränderung anzupassen.
Zudem werden die Plugins Sugiyama, Joana und graphml importer als Pakete modeliert.
Das Sugiyama Plugin liefert eine allgemeine, erweiterbare Implementierung des Sugiyama-Frameworks. Es stellt Interfaces bereit, die von anderen Plugins genutzt werden können, um ihre Graphen mit diesem Plugin zu layouten. Zusätzlich werden die einzelnen Phasen des Frameworks in Form eines Strategy Patterns austauschbar gemacht, sodass man, beispielsweise an Stelle der heuristischen Methoden, optimale Lösungen implementieren könnte.
Das Joana Plugin liefert Algorithmen zur spezifischen Bearbeitung von Joana Graphtypen, die auch hier modeliert werden. Es stützt sich stark auf das Sugiyama Plugin, welches es erweitert. Hier werden auch joana spezifische Constraints sowie Filter geliefert.
Der GraphML Importer implementiert das Importer Interface aus plugin und soll graphml-

Dateien importieren können.
Allgemeine Kriterien(siehe 1.1.1), wie das bereitstellen von Tabs für verschiedene Graphen (siehe /FA260/), werden häufig schon von javafx ermöglicht.

2 Klassendiagramme

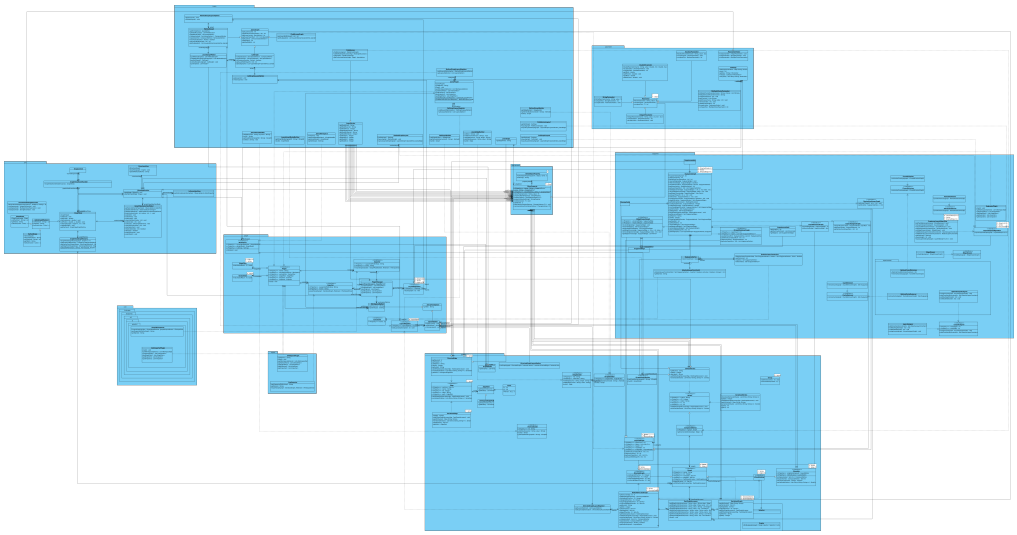

Abbildung 2.1: Diagramm aller Klassen

3 Paketeinteilung

Die Paketeinteilung der Software gliedert sich in folgende Pakete:

3.1 graphmodel

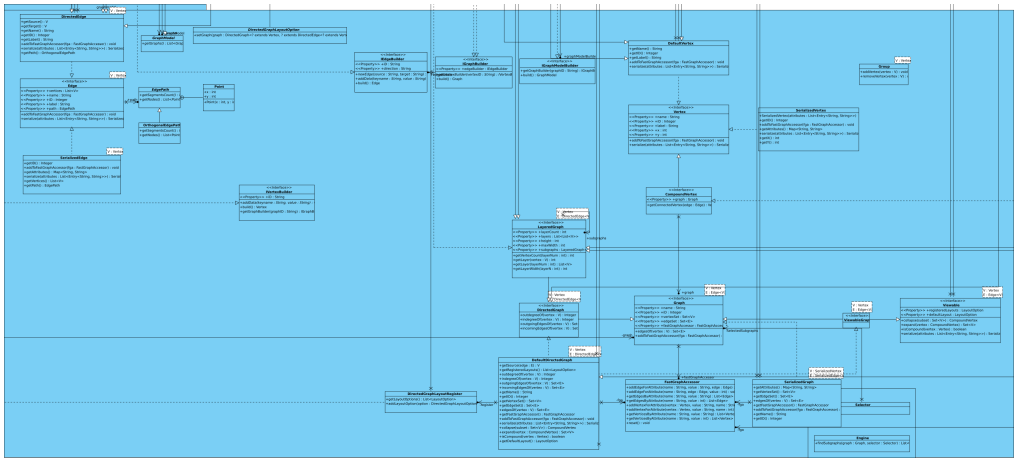

Abbildung 3.1: Paketübersicht graphmodel

In dem Paket **graphmodel** befinden sich die Klassen, die für die interne Repräsentation des Graphen benötigt werden. Die Funktionalität dieses Pakets ist es ein flexibles Graph-Modell, sowie schnellen Zugriff auf dessen Komponenten bereitzustellen.
Zum Graph-Modell gehören Interfaces wie **GraphModel**, **Graph**, **Vertex** und **Edge**, welche den groben Aufbau eines Graphen und dessen Funktionalitäten vorgeben. Das **GraphModel** enthält alle Graphen, welche bei einem Import aus der Datei ausgelesen wurden. Da es Kanten und somit auch Graphen gibt, welche gerichtet sind, gibt es dazu auch passende Interfaces, welche die zuvor genannten erweitern. Zusätzlich gibt es noch weitere Spezialisierungen der Interfaces, wie **CompoundVertex**, ein Knoten der einen Subgraphen enthält und **LayeredGraph**, ein Graph der noch Schichten speichert.
Die genannten Interfaces sind teilweise schon implementiert. Zu diesen Klassen gehören die **Default**-Klassen, welche die im Interface definierten Methoden primitiv implementieren und die **Serialized**-Klassen, welche ein jeweiliges Element darstellen, welche einheitlich für den Export vorbereitet sind.
Damit speziellere Implementierungen von **Graph**, **Vertex** oder **Edge** aus **Plugins** vom **Importer** einheitlich erstellt werden können gibt es die **Builder**-Interfaces. Sie geben eine Schnittstelle vor, über welche die speziellen Graphelemente erstellt werden können, ohne dass der **Importer** davon in Kenntnis gesetzt werden muss.

3.2 gui

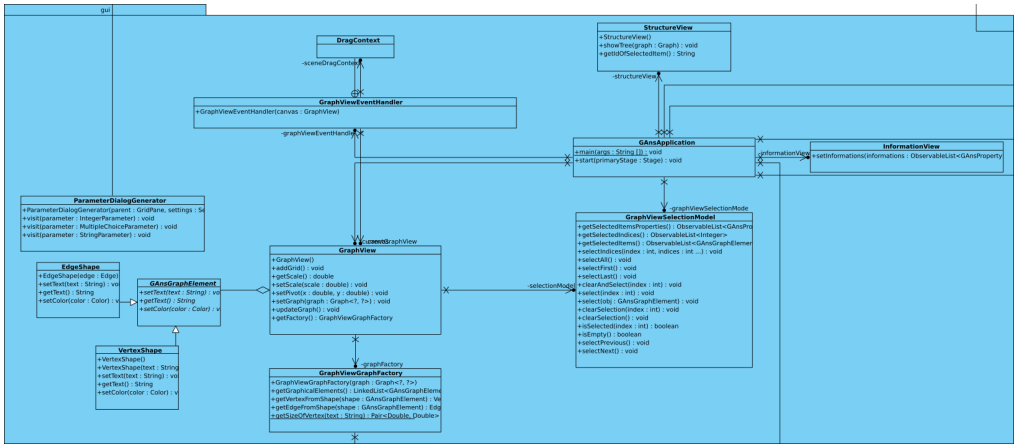

Abbildung 3.2: Paketübersicht gui

Das Paket **gui** beinhaltet die Anzeigelogik, sowie die Haupt-Applikation von GAns. Dafür werden JavaFX Komponenten verwendet, die erweitert und angepasst werden. Es werden Views und Elemente definiert, die für die Anzeige notwendig sind. Die **GAnsApplication** hat, als Klasse welche das ganze Programm aufsetzt, noch weitere Aufgaben. Sie stößt das Laden der Plugins an und behandelt zusätzlich viele Interaktionen des Benutzers mit den einzelnen GUI-Elementen unter anderem auch das Laden bzw. Erstellen des unterliegenden GraphModels(siehe 3.1). Es werden verschiedene Views definiert:

GraphView Eine View welche den gelayouteten Graphen anzeigt und dazu verschiedene Navigationsmöglichkeiten, wie Zoom, Selektion usw., anbietet. Es können mehrere GraphViews gleichzeitig geöffnet sein. Jede einzelne ist in einem eigenen Tab enthalten, zwischen denen hin und her gewechselt werden kann. Die Elemente welche in der GraphView angezeigt werden, sind **GAnsGraphElemente**. Sie bestehen aus verschiedenen JavaFX Klassen, welche einen Text und eine Farbe gesetzt bekommen können. Standardmäßig werden Rechtecke für Knoten und direkte Kanten als Kanten zwischen Knoten verwendet.

StructureView Eine TreeView, welche die geschachtelten Graphen einer importierten Datei anzeigt. Über doppelklick auf eines der Baumelemente wird der durch das Element dargestellte Graph in einer neuen GraphView angezeigt. Zum Erstellen der einzelnen Baumelemente werden normale JavaFX-Funktionalitäten verwendet.

InformationView Eine TableView, welche, je nach Selektion in der aktuell angezeigten GraphView, Informationen über die selektierten Elemente anzeigt. Jede Zeile der Tabelle enthält in der ersten Zelle den Bezeichner/Namen und in der zweiten Zelle den jeweiligen Wert der Eigenschaft aus der Selektion. Die einzelnen Zellen werden durch eine JavaFX **PropertyValueFactory** erstellt

welche mit den **GAnsProperties**(siehe 3.5) der selektierten Elemente der GraphView arbeitet.

3.3 parameter

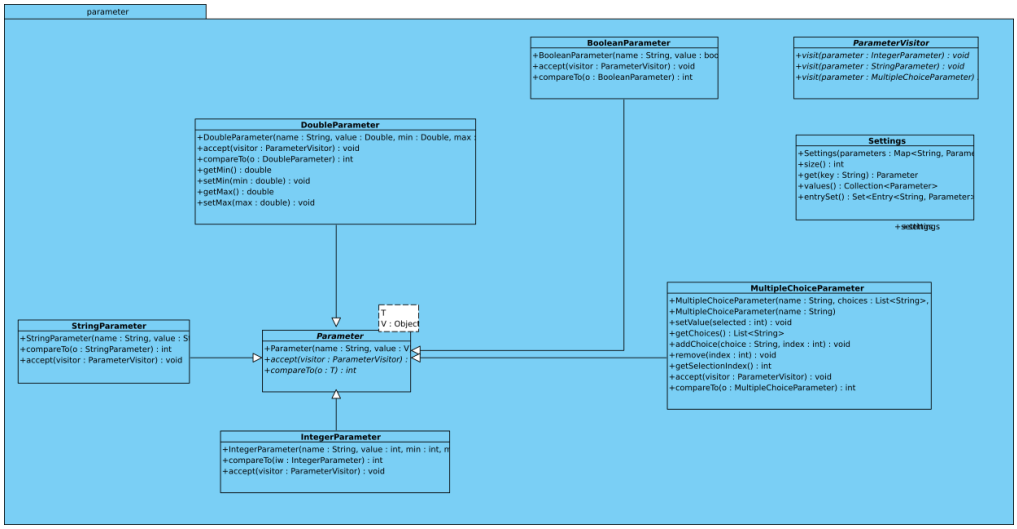

Abbildung 3.3: Paketübersicht parameter

Dieses Paket trennt Rahmenbedingungen(Parameter) von Layouts oder Constraints in eigene Klassen. Diese Wrapper-Klassen erlauben z.B. einem Layout seine Einstellungsmöglichkeiten an die GUI zu übergeben, ohne das vom Layout aus Wissen über GUI-Elemente gefordert wird. Die implementierten Parameter können von Visitors besucht werden, welche den Wert des Parameters auslesen und verändern können. Im GUI-Paket gibt es eine Generator-Klasse, welche einen solchen Visitor implementiert. Somit können die Einstellungsdialoge automatisch gefüllt werden und gemachte Änderungen im Dialog werden direkt wieder in den Parametern gesetzt. Da in gewissen Situationen von der GUI, oder dem zugehörigen Layout, Listener auf die Werte gesetzt werden müssen, erben die Parameter von GAnsProperty, da diese bereits viel benötigte Funktionalität vereinfacht anbieten und Gemeinsamkeiten aufweisen.

3.4 graphRegex

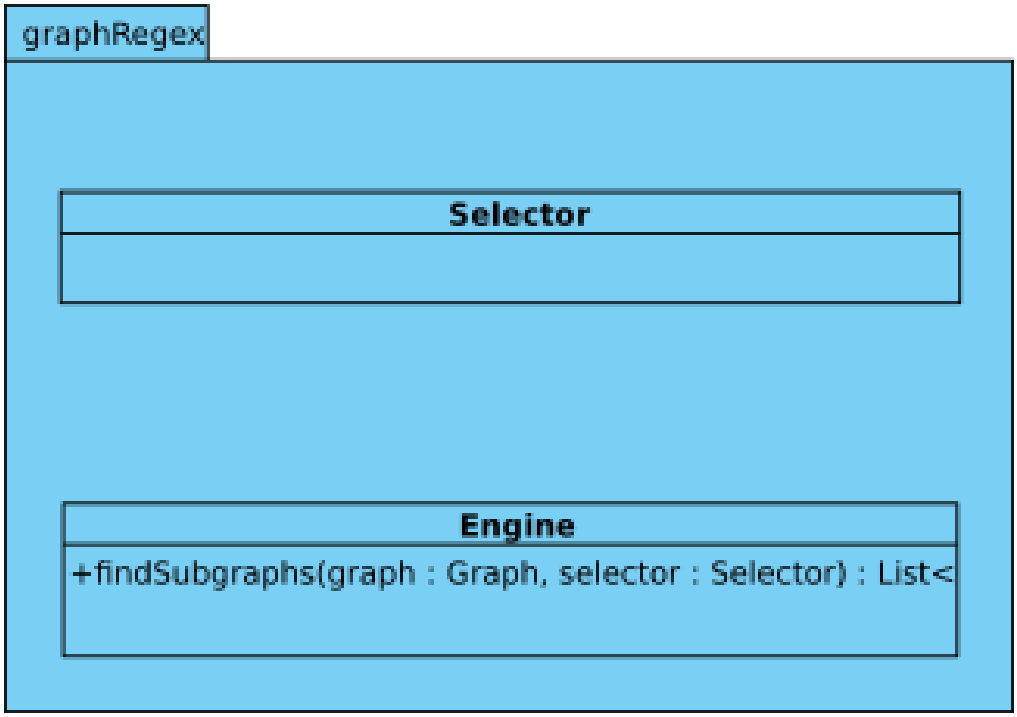

Abbildung 3.4: Paketübersicht graphRegex

In diesem Paket befinden sich die Algorithmen zum durchsuchen von Graphen nach Subgraphen mithilfe von Selektoren. Es gibt eine **Selector** Klasse, welche die Information, welche Subgraphen selektiert werden sollen, speichert und eine **Engine**, die aus einem gegebenen Graphen die Subgraphen, welche die Beschreibung erfüllen, heraussucht und zurückgibt. Dieses Paket dient hauptsächlich als Helfer für Layoutalgorithmen, die auf diese Weise für gegebene Constraints bestimmte Subgraphen selektieren und speziell verarbeiten können.

3.5 objectproperty

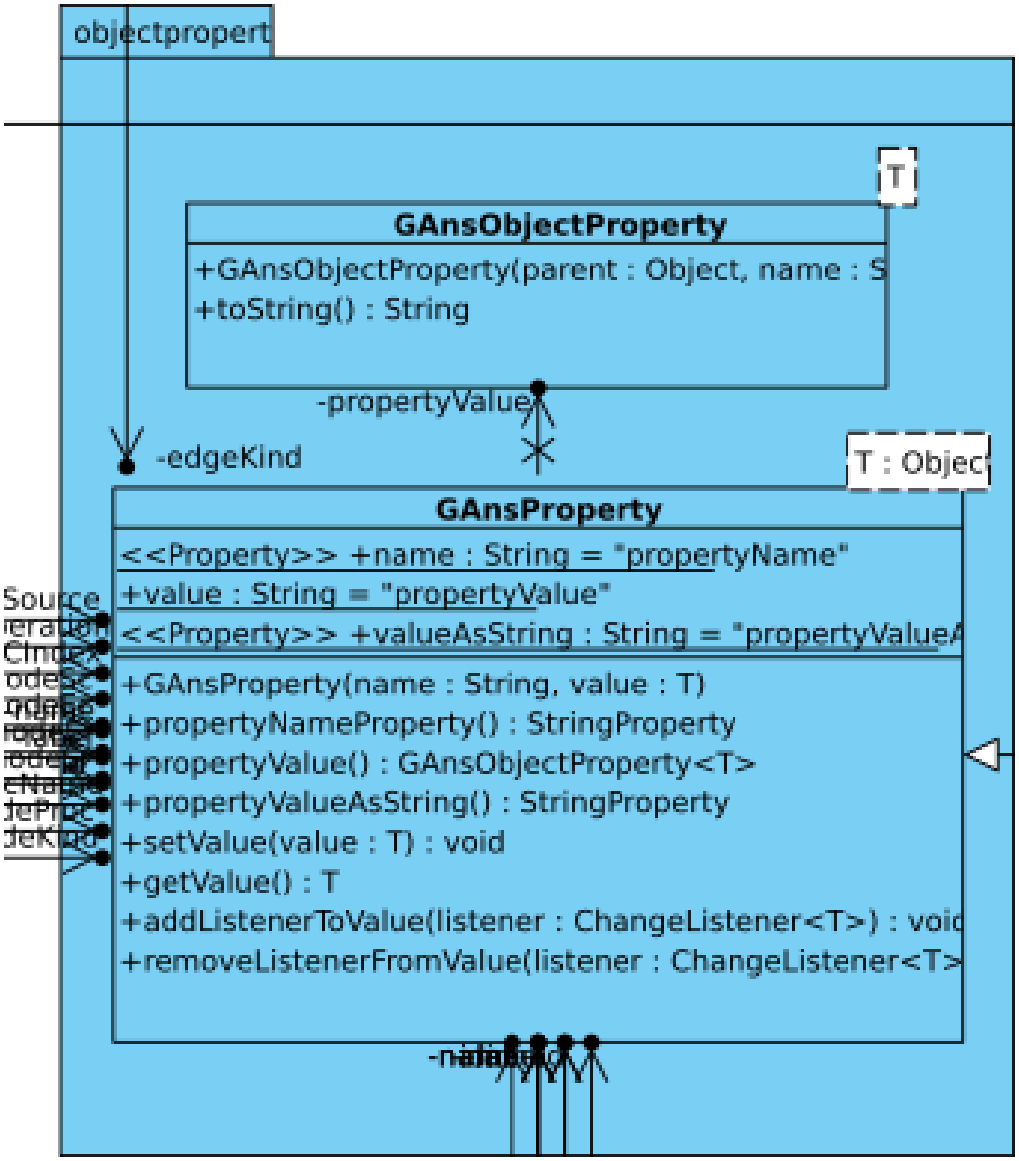

Abbildung 3.5: Paketübersicht objectproperty

Das Paket **objectproperty** bietet eine generische Eigenschaft eines Elements aus dem Paket **graphmodel**. Dabei stellt jede **GAnsProperty** eine einzelne Eigenschaft dar, welche aus einen Bezeichner/Namen, einen generischen Wert und eine Stringrepräsentation des Werts besteht. Die einzelnen Bestandteile werden in JavaFX ObjectProperties gespeichert. Darunter die **GAnsObjectProperty**, welche einen generischen Typ unterstützt. Diese Properties harmonieren gut mit den restlichen JavaFX-GUI-Elementen, somit kann auf viele Funktionen zurückgegriffen werden kann die von JavaFX direkt angeboten werden ohne große Eigenentwicklungen anfertigen zu müssen. Das beste Beispiel dafür ist die InformationView der GUI, deren Inhalt

durch **PropertyValueFactories** von JavaFX automatisch erstellt werden, sobald eine Liste mit GAnsProperties gesetzt wird. Dabei ist der eigentliche Typ des Werts irrelevant, da in der GUI auf die Stringrepräsentation des Werts zugegriffen werden kann, wobei z.B. im Model oder Layoutalgorithmus mit dem eigentlichen Wert gearbeitet werden kann. Die PropertyValueFactories arbeiten mit dem Bestandteil der GAnsProperty, welcher mit einem bestimmten Referenz-String erstellt wurde. Diese Strings sind in GAnsProperty statisch gegeben, sodass die Aufrufe selbst bei Änderungen in der GAnsProperty im restlichen Programm konstant bleiben.

3.6 plugin

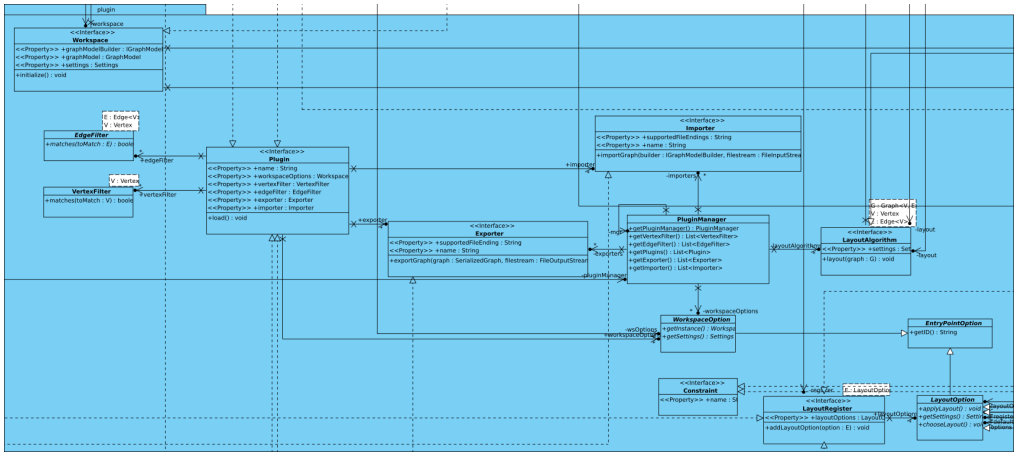

Abbildung 3.6: Paketübersicht plugin

Das Paket **plugin** bietet Interfaces für Plugins sowie einen **PluginManager** welcher die Plugins lädt, verwaltet und zur Verfügung stellt. Klassen die Interfaces dieses Plugins implementieren müssen sich beim laden, der Klasse die **Plugin** erweitert beim **PluginManager** registriert werden, um dann vom Programm verwendet werden zu können. Es bietet somit die Möglichkeit Funktionalität von außen zu laden und somit das Programm zu erweitern.
Ein **Importer** und ein **Exporter** Interface können implementiert werden um weitere File Importer oder Exporter hinzuzufügen.
Ein **LayoutAlgorithm** ist in der Lage einen Graphen zu layouten. Er hat **LayoutOption**s und **Constraint**s die eingestellt werden können, um die funktionsweise des Algorithmus zu beeinflussen. Die **LayoutOption**s können dann in einem **LayoutRegister** gespeichert werden.
EdgeFilter, **VertexFilter** und **FilterSet** erlauben es Plugins Filter für Kanten und Knoten zu definieren die bei Layouten und Darstellung unbeachtet bleiben.
Workspaces beschreiben eine Arbeitssituation in der spezifische Graphtypen verarbeitet werden können. Ein Beispiel hierfür ist der **JoanaWorkspace** im **joana** Paket. Ein Workspace definiert, welche Dateien mit welchem Importer importiert werden und ermöglicht so das öffnen spezifischer Graphtypen welche in generischen Formaten wie graphml gespeichert werden können. Dieses kann wiederum über **WorkspaceOption**s eingestellt werden.

3.7 sugiyama

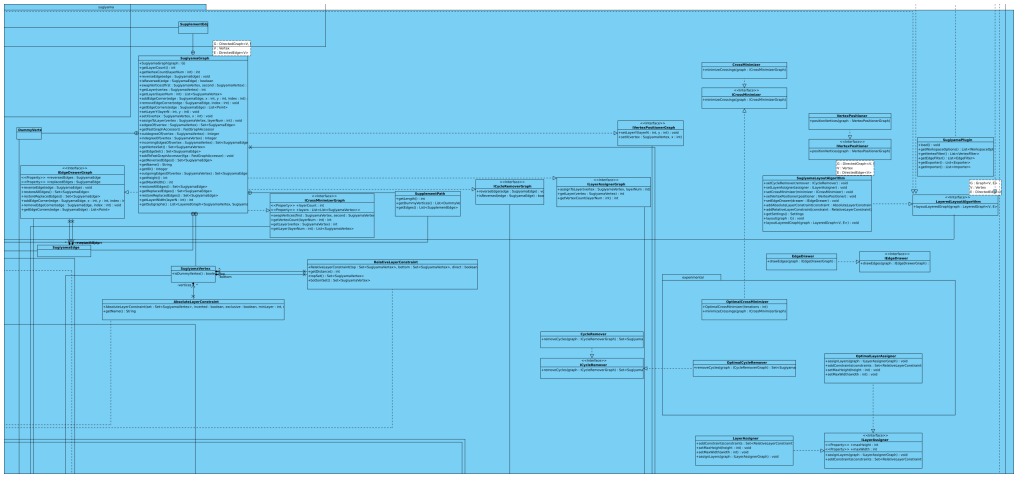

Abbildung 3.7: Paketübersicht sugiyama

Das Paket **sugiyama** bietet neben einer Implementierung des Sugiyama-Frameworks zum hierarchischen layouten eines gerichteten Graphen Schnittstellen zum ändern und austauschen einzelner Phasen dieses Frameworks.
Die fünf Phasen des Sugiyama-Frameworks: Kreise entfernen, Lagenzuordnung, Kreuzungsminimierung, Knotenpositionierung und Kantenzeichnen werden in dieser Reihenfolge durchlaufen und durch einen **SugiyamaLayoutAlgorithm** verwaltet und gesteuert. Die Klassen, die diese Schritte darstellen **CycleRemover**, **LayerAssigner**, **CrossMinimizer**, **VertexPositioner** und **EdgeDrawer** arbeiten auf einem **SugiyamaGraph**, welcher den zu layoutenden Graphen repräsentiert. Hierzu implementiert er die jeweiligen Interfaces, die diese Schritte fordern.
In der ersten Phase werden die Zykel des Graphen entfernt, indem die Richtungen einer minimalen Kantenmenge umgekehrt wird.
Anschließend wird jedem Knoten eine Lage zugewiesen, welche abhängig von der Anzahl der ein- und ausgehenden Kanten des Knotens, sowie der Länge des kürzesten Pfades des Knotens zu einem Knoten aus der ersten Lage ist. Zudem werden, falls eine Kante zwischen zwei Knoten mindestens eine Lage überspringt, in jede dieser Lagen ein **DummyVertex** ein.
In der dritten Phase wird die Anzahl der Kantenkreuzungen minimiert. Dazu werden jeweils in zwei aneinanderliegenden Lagen die Knoten beider Lagen umgeordnet.
In Phase Vier bekommen die Knoten feste Positionen in den Lagen zugewiesen
In der letzten Phase wird jeder **DummyVertex** entfernt und die Kanten werden gezeichnet. Zusätzlich werden die im ersten Schritt umgedrehten Kanten wieder in ihre ursprüngliche Richtung zurückversetzt.

3.8 joana

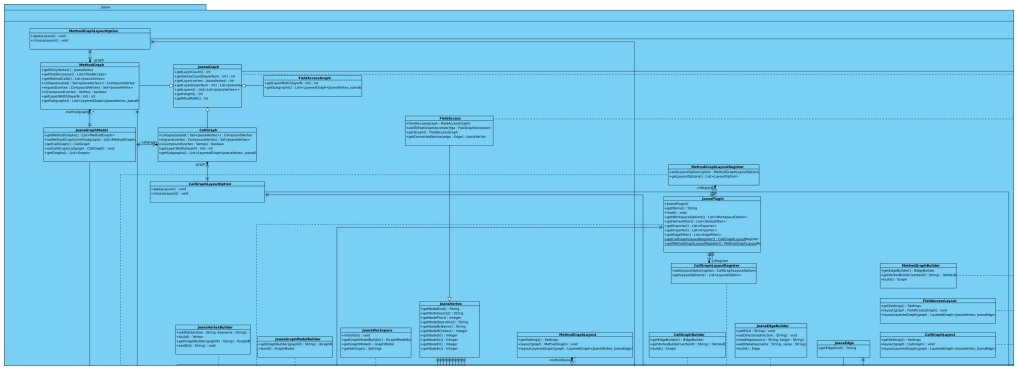

Abbildung 3.8: Paketübersicht joana

Das **joana** Paket bietet Funktionalitäten zum Erstellen von Graphen samt Knoten und Kanten, sowie das Darstellen von **Methoden**- und **Callgraphen** in ihren eigenen Layouts.
Das **Workspace** bietet die Möglichkeit zum Einstellen und Speichern von Optionen, speziell für **JOANA-Graphen**.
Zudem ermöglicht das **joana** Paket die wiedererkennbare Darstellung von Feldzugriffen über das **FieldAccessLayout**, indem die Zugriffe immer ähnlich dargestellt werden.
JoanaVertex und **JoanaEdge** beinhalten Attribute, welche speziell auf die Eigenschaften abgestimmt sind, welche durch JOANA gesetzt werden.
Callgraph und **Methodengraph**(en) werden in einem speziellen **JoanaGraphModel** gespeichert.

3.9 export

Das **export** Paket bietet eine Implementierung eines Export-Plugins von GAns. Das Plugin definiert den Export eines angezeigten Graphen als SVG-Datei. Dazu wurde ein SvgExport-Plugin erstellt, welches von GAns geladen wird und ein SvgExporter, welcher die vorgegebene Exporter-Schnittstelle implementiert.

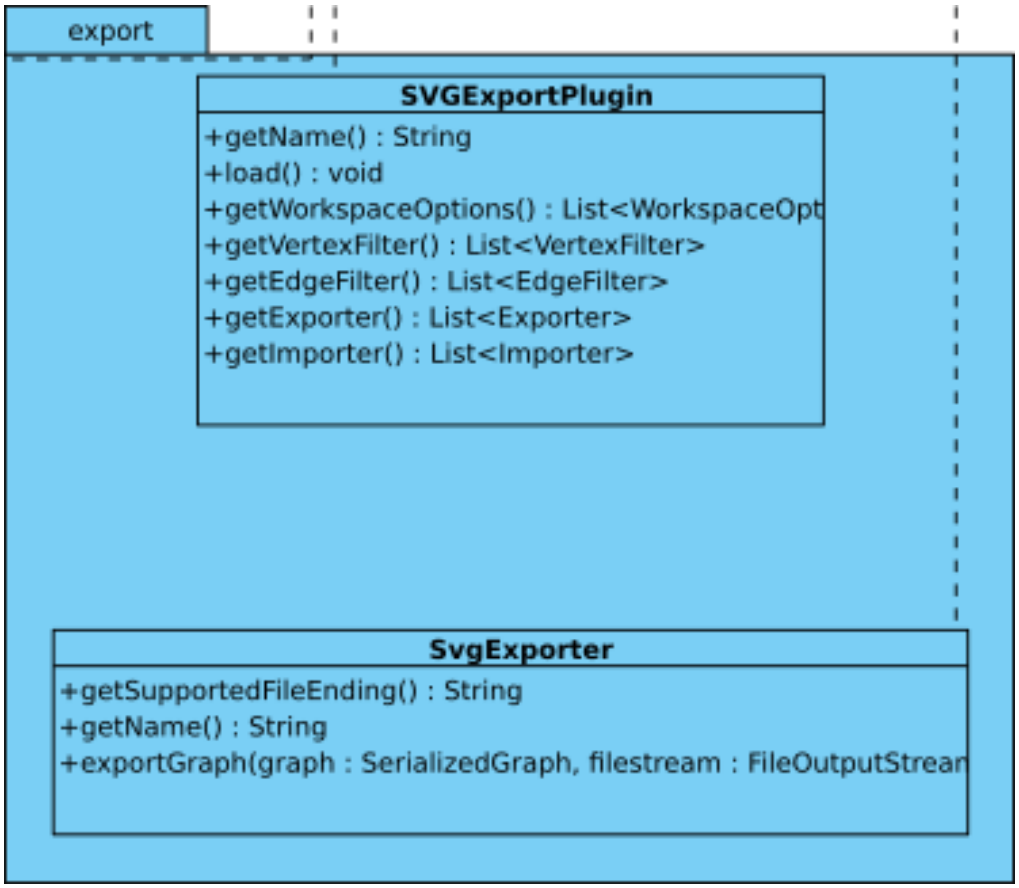

Abbildung 3.9: Paketübersicht joana

4 Verwendete Entwurfsmuster

4.1 Parameterklassen Visitor Pattern

In den Parameterklassen wird ein Visitor Pattern verwendet um bei Änderungen der Werte auch direkte Änderungen in der GUI zu erhalten und umgekehrt. Die zum textspezifischen Parameter passende Funktion des Visitors wird per Double Dispatch vom Parameter aufgerufen. In dieser Funktion kann der Visitor nun Änderungen am Parameter machen, oder ihn Auslesen. In der GUI ist ein solcher Visitor implementiert. Er liest mehrere Parameter aus und erstellt passend zu den Typen und Werten der Parameter einen Dialog, welcher genau Elemente beinhaltet, mit denen man die Werte der Parameter anpassen/editieren kann. Wurde ein Wert im Dialog geändert wird dieser im passenden Parameter gesetzt. Deshalb werden die Parameterklassen zum dynamischen Erstellen von Einstellungsdialogen für Layouts und/oder Constraints verwendet.

4.2 Plugins

Durch Plugin soll eine einfache Erweiterung des Programmes möglich gemacht werden. Dafür werden Schnittstellen zur Verfügung gestellt welche zu jeder Zeit erweitert werden können. Es existieren Schnittstellen für:

- Importer und Exporter um neue Dateiformate unterstützen zu können
- Layoutalgorithmen für verschiedene Graphrepräsentationen
- Graphrepräsentationen, welche mit einem **Workspace** implementiert werden

4.3 Sugiyama Algorithmus Strategy Pattern

Der **SugiyamaLayoutAlgorithm** basiert auf dem Strategy Pattern. Es gibt 5 Phasen/Algorithmen welche das Sugiyama Framework durchläuft. Jede Phase entspricht einer Strategie, welche durch eine Schnittstelle definiert ist (z.B **ICrossMinimizer**). Die Konkreten Strategien implementieren dann diese gegebene Schnittstelle. Dadurch können einzelne Phasen des SugiyamaLayoutAlgorithm leicht ausgetauscht werden.

4.4 Graph Builder Builder Pattern

Da viele unterschiedliche Graphtypen existieren können, wird für die Konstruktion eines speziellen Graphes ein Builder Pattern verwendet. Dies hat den großen Vorteil, dass ein Importer nur eine Schnittstelle kennen muss, aber unterschiedliche Graphtypen importieren kann. Dazu erhält der Importer einen GraphModelBuilder, welchem er die Attribute des Graphes übergibt.

Der konkrete GraphModelBuilder konstruiert nun aus diesen Attributen die spezielle Graphrepräsentation.

4.5 GUI-Klassen Model View Controller Pattern

In der GUI von GAns gibt es verschiedene **Ansichten(Views)** welche auf verschiedenen **Datensätzen(Models)** darstellen und auf Benutzerinteraktionen unterschiedlich reagieren.
Die **StrukturView** arbeitet auf einem GraphModel-Object, bzw. dessen Liste von Graphen, welche beim Import von GAnsApplication gesetzt wird. Die meisten Interaktionen werden hier von der View selber behandelt. Da aber der Doppelklick auf ein Baumelement eine anzeigeübergreifende Aktion ist, wird dieser von der GAnsApplication bearbeitet.
Eine **GraphView** arbeitet auf einem Graph-Objekt aus dem graphmodel-Paket welches durch die GAnsApplication gesetzt wird. Die GraphView behandelt alle Benutzerinteraktionen, wie Zoomen und navigieren selbst und hat ein SelectionModel(GraphViewSelectionModel), welches Änderungen in der Selektion behandelt.
Die **InformationView** arbeitet auf dem SelectionModel der aktuell angezeigten GraphView. Ändert sich die Selektion wird die InformationView informiert und bekommt neue Daten gesetzt. Einen Controller gibt es nicht, da die TableView keine besonderen Benutzerinteraktionen benötigt.

5 Sequenzdiagramme

5.1 Import von einem JOANA Graphen

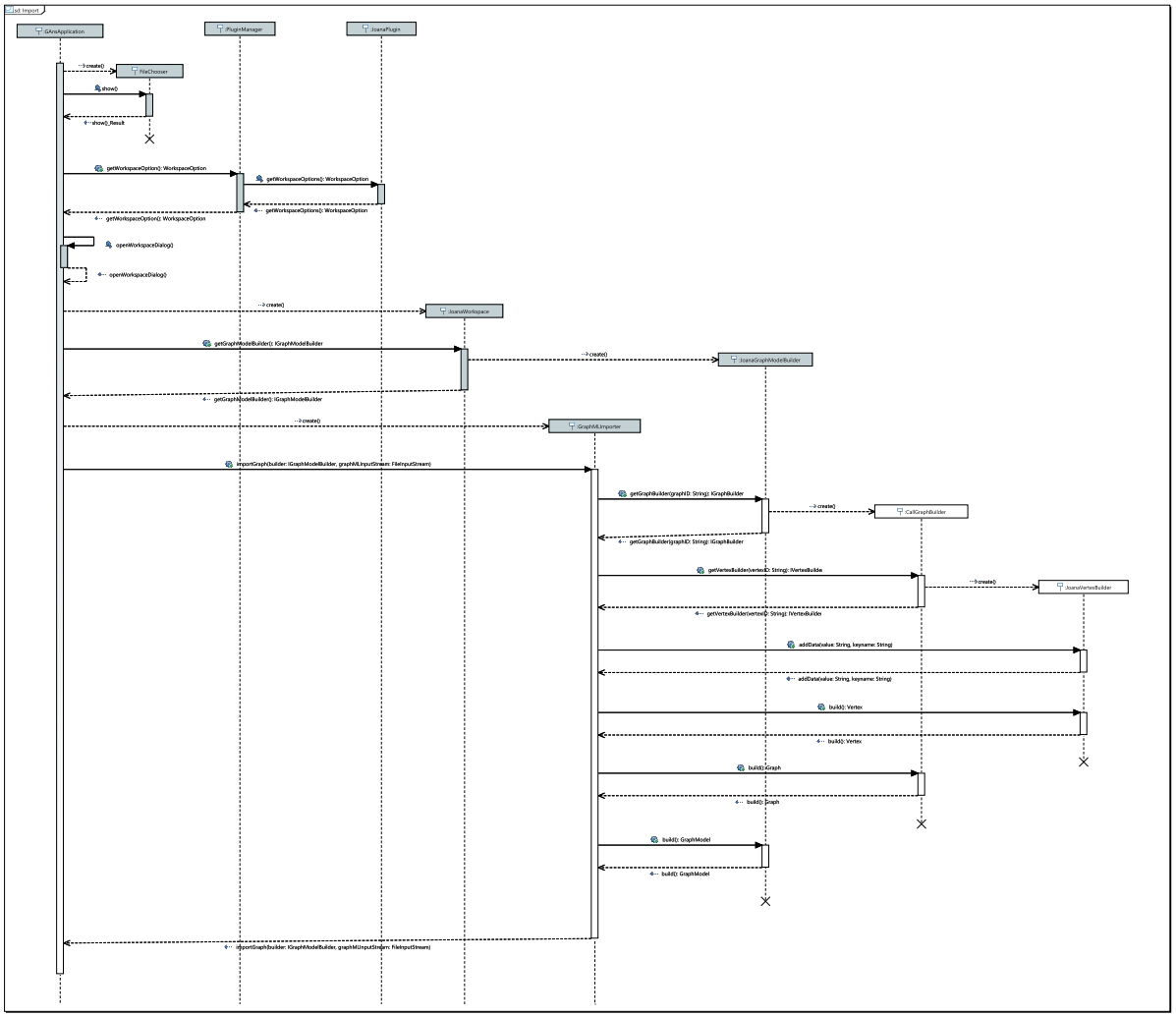

Abbildung 5.1: Sequenz Diagramm für den Import

Der Benutzer möchte einen Graphen importieren und klickt auf Import. Die GAnsApplication öffnet einen FileChooser, in welchem der Benutzer die Datei auswählen kann welche er importieren möchte. Danach öffnet sich ein Workspace Dialog in welchem der Benutzer auswählt als welcher Typ der Graph interpretiert werden soll.

Das Workspace übergibt der GAnsApplication nun den dazugehörigen IGraphModelBuilder. Dieser wird dann dem Importer übergeben, welcher mithilfe des Builders eine Repräsentation des Graphen erstellt, welche Workspace spezifisch ist.

5.2 Öffnen eines Graphen über die Strukturansicht

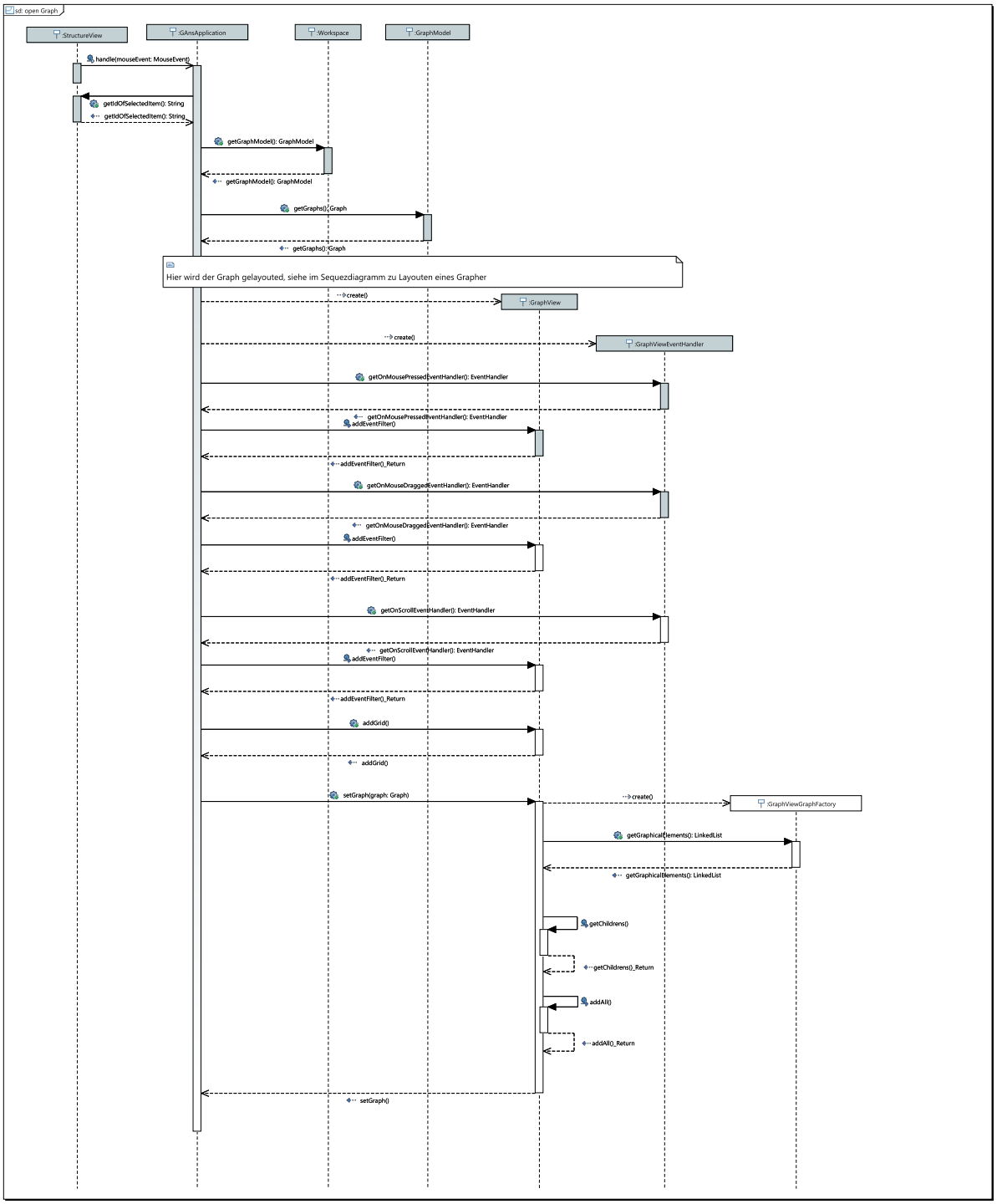

Abbildung 5.2: Sequenz Diagramm das Öffnen eines Graphen aus der Strukturansicht

Der Benutzer hat einen Doppelklick auf ein Element in der StruktureView gemacht. Aus der StruktureView wird die ID des selektierten, gerade doppelt angeklickten, Elements gespeichert. Über das Workspace wird in den Graphen des geladenen GraphModels nach dem zur ID gehörenden Graph gesucht. Der gefundene Graph wird nun gelayouted (siehe 5.4) dabei werden nicht die Dialoge geöffnet, sondern direkt die default LayoutOption des Workspace für den Graphen verwendet. Nachdem der Graph vom layouten zurückgegeben wird, wird eine neue GraphView und ein neues GraphViewEventHandler-Objekt erzeugt, die einzelnen EventFilter werden zur GraphView hinzugefügt und ein Grid wird gesetzt, welches für die Zoom-Funktion benötigt wird. Ist die GraphView aufgesetzt, wird der Graph an sie übergeben. Die GraphView erzeugt eine eigene Factory, welche passend zum Graphen, GraphischeElemente erzeugt. Die GraphView setzt diese Elemente, wodurch sie angezeigt werden.

5.3 Ändern der Selektion im Graphen

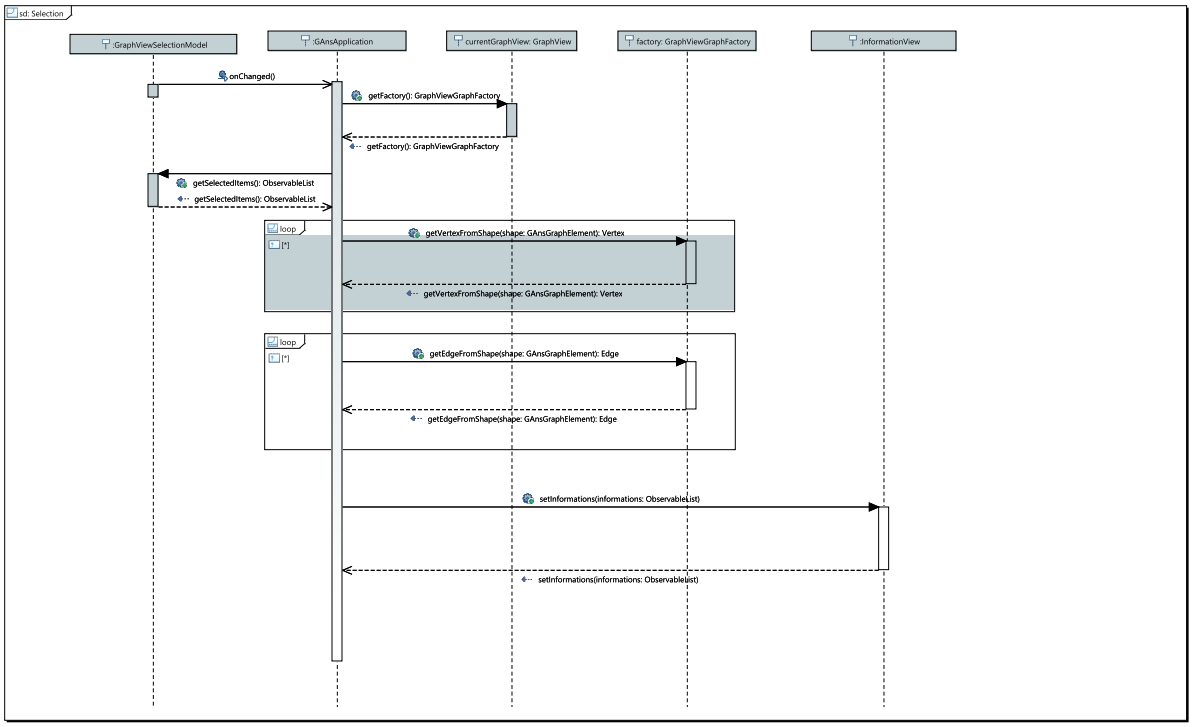

Abbildung 5.3: Sequenz Diagramm für geänderte Selektion

Der Benutzer hat in der aktuellen GraphAnsicht die Selektion geändert. Die GAnsApplication wird vom GraphViewSelectionModel benachrichtigt. Die GAnsApplication holt sich von der aktuellen GraphAnsicht die Factory, welche zugriff auf die Elemente in der GraphView bietet, und vom SelectionModel eine Liste mit allen selektierten Elementen. Über die Factory und wird nun zu jedem selektiertem Element die zugehörige Vertex oder Edge aus dem angezeigten Graphen ermittelt. Deren GAnsProperties werden in eine ObservableList zusammengeführt und an die InformationsView übergeben, welche die Informationen die über die Properties gegeben sind darstellt.

5.4 Layouten eines JOANA-Graphs

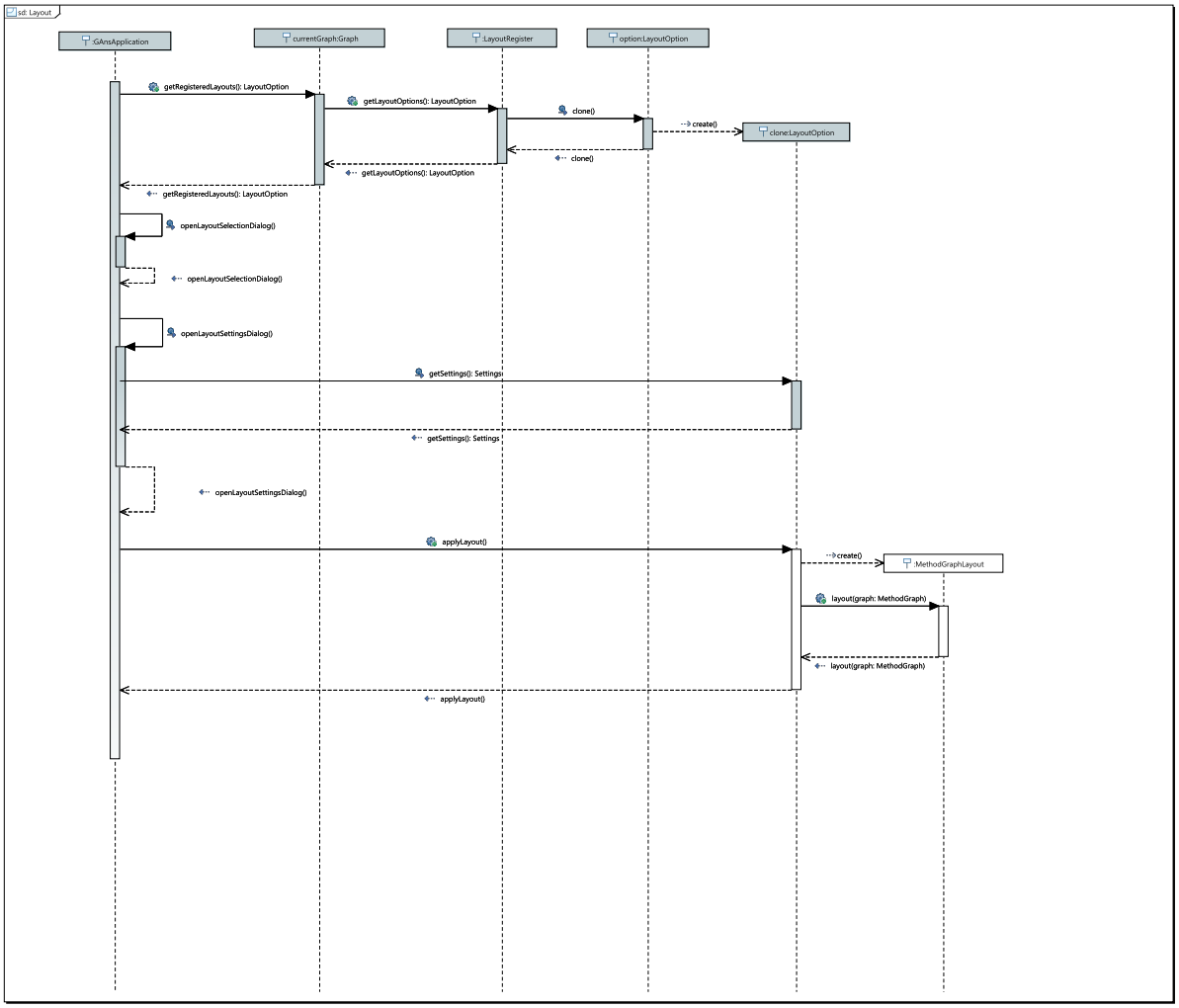

Abbildung 5.4: Sequenz Diagramm für Layouten eines JOANA-Graphs, Oberfläche

Der Benutzer hat das Layouten des angezeigten Graphen über einen Menüeintrag angestoßen. Die GAnsApplication holt sich über den Graphen alle registrierten LayoutOptions. Diese werden in einem Dialog zur Auswahl gestellt. Nachdem der Benutzer eine Wahl getroffen hat wird ein Dialog geöffnet in dem Einstellungen zur gewählten LayoutOption gemacht werden können. Nun wird auf den gewählten LayoutOption applyLayout aufgerufen, ein MethodGraphLayout erstellt und auf den Graph angewendet(siehe 5.5).

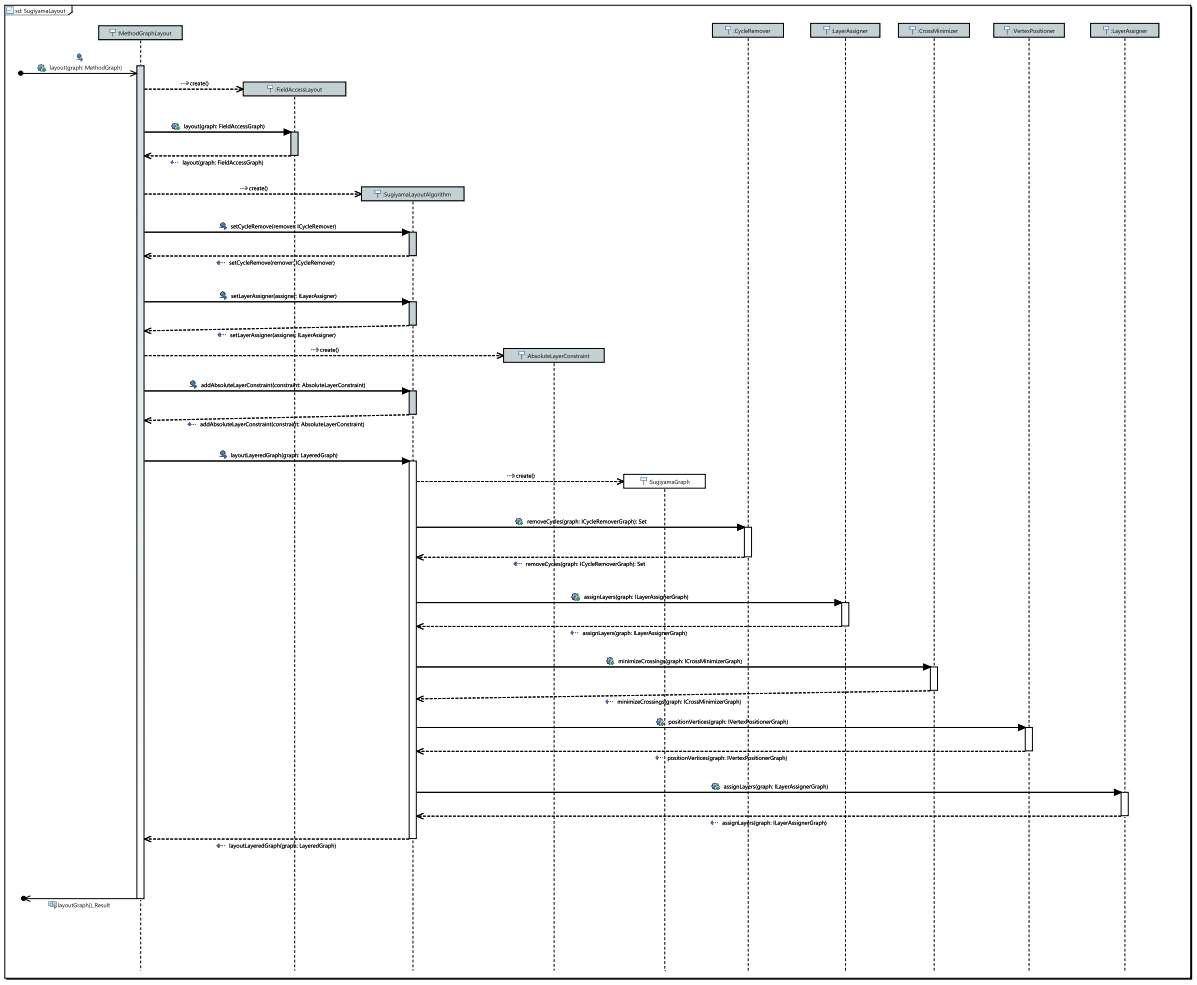

Abbildung 5.5: Sequenz Diagramm für Layouten eines JOANA-Graphs, Sugiyama

Nachdem das Layouten mit dem MethodGraphLayout angestoßen wurde, wird zuerst ein FieldAccessLayout erstellt und damit alle FieldAccessGraphen des MethodenGraphen gelayouted. Danach wird ein SugiymaLayoutAlgorithm erstellt, die jeweiligen Phasen gesetzt, ein SugiyamaGraph erstellt und die jeweiligen Phasen auf den Graphen angewandt.

5.5 Export von einem geladenen JOANA-Graphen als SVG

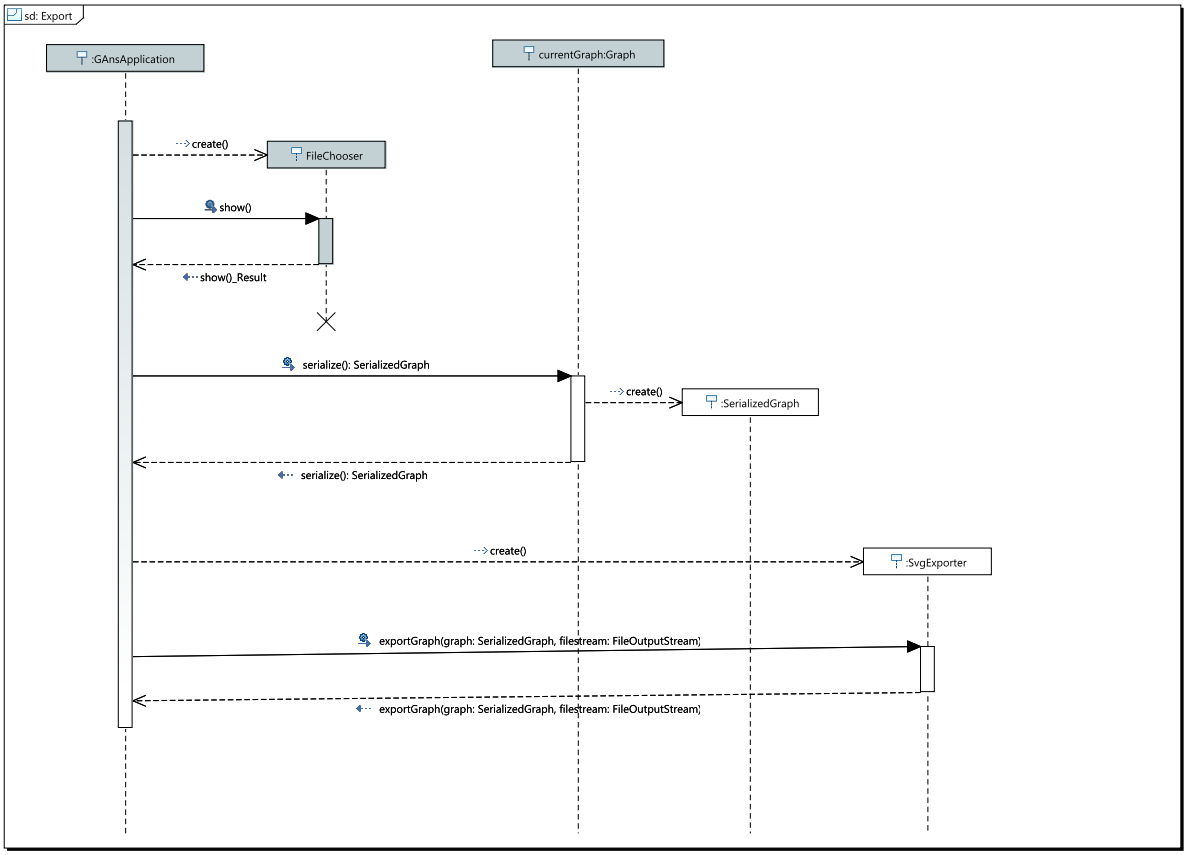

Abbildung 5.6: Sequenz Diagramm für Layouten eines JOANA-Graphs, Sugiyama

Der Benutzer möchte einen Graphen exportierten und klickt auf Export. Die GAnsApplication öffnet einen FileChooser, in welchem der Benutzer den Ort und Namen der Datei angeben kann, in die exportiert werden soll. Danach wird der Graph serialisiert und dessen serialisierte Repräsentation in den SvgExporter gegeben, welcher die Daten aus dem Graph in SVG-Form an den ausgewählten Ort schreibt.

6 Differenzen zum Pflichtenheft

Folgende Funktionale Anforderungen aus dem Abschnitt 4.2 Wunschfunktionen wurden noch nicht in den Entwurf mit aufgenommen:

- /FA300/ Constraints hinzufügen
- /FA310/ Constraint bearbeiten
- /FA320/ Constraint löschen
- /FA330/ Steuerung über Tastaturkürzel
- /FA340/ Fortschrittsbalken bei Berechnung
- /FA350/ Eine Übersicht des angezeigten Graphen
- /FA360/ Änderung der Darstellung von Kanten
- /FA370/ Reload-Funktion
- /FA380/ Automatisierte Subgraph Auswahl
- /FA390/ Testen von Erreichbarkeit von Knoten

7 Klassenbeschreibungen

Export der JavaDoc aus dem Entwurf. Das Team hat sich auf englische JavaDocs geeinigt, sodass dieses Kapitel in englisch gehalten ist.

7.1 Package plugin

Package Contents *Page*

Interfaces

Classes

7.1.1 Interface Constraint

The most abstract interface for a constraint. Can later be used to achieve compatibility between constraints of different domains and layouts.

7.1.1.1 Declaration

```
public interface Constraint
```

7.1.1.2 Methods

- **getName**

  ```
  java.lang.String getName()
  ```

  - **Description**

    The name of the constraint
  - **Returns** – the name

7.1.2 Interface Exporter

The exporter interface is implemented to export a graph from it's internal representation into a specific file. For every graph structure given as SerializedGraph/SerializedVertex/SerializedEdge interfaces the implementing class translates it into a FileOutputStream for the given file type, by `getSupportedFileEnding`.

7.1.2.1 Declaration

```
public interface Exporter
```

7.1.2.2 Methods

- **exportGraph**

  ```
  void exportGraph(graphmodel.SerializedGraph graph,java.io.
      FileOutputStream filestream)
  ```

  - **Description**

    This method writes an `SerializedGraph` (in 7.3.25, page 92) into an FileOutputStream. The `SerializedGraph` (in 7.3.25, page 92) enables us to read all attributes as Strings. To write the contained Information into the filestream is the task of this method

  - **Parameters**
    - `graph` – serializedGraph that contains the information to write to a file
    - `filestream` – to write the information into

- **getName**

```
java.lang.String getName()
```

  - **Description**

    Gets the name of this importer.
  - **Returns** – name of this exporter

- **getSupportedFileEnding**

```
java.lang.String getSupportedFileEnding()
```

  - **Description**

    Gets all filetypes which this exporter can parse.
  - **Returns** – the supported file ending.

7.1.3 Interface Importer

The importer interface is implemented when writting a class that can import files. This will enable Plugins that import specific files to be build. The main task of a class implementing this interface is to parse a FileInputStream into the Interface of an `IGraphModelBuilder` (in 7.3.7, page 66). The `IGraphModelBuilder` (in 7.3.7, page 66) will then build the representation.

7.1.3.1 Declaration

```
public interface Importer
```

7.1.3.2 Methods

- **getName**

```
java.lang.String getName()
```

  - **Description**

    Gets the name of this importer.
  - **Returns** – name of this importer

- **getSupportedFileEndings**

```
java.lang.String getSupportedFileEndings()
```

- **Description**

  Gets the filetype which this importer can parse.

- **Returns** – the supported file ending.

- **importGraph**

```
void importGraph(graphmodel.IGraphModelBuilder builder,java.io.
    FileInputStream filestream)
```

- **Description**

  This method parses an FileInputStream into an `IGraphModelBuilder` (in 7.3.7, page 66). It has to ensure that all information is transfered to a correct graphmodelbuilder.

- **Parameters**

  * `builder` – that the values are parsed into
  * `filestream` – from which the values are parsed

7.1.4 Interface LayoutAlgorithm

An implementations of LayoutAlgorithm takes a graph. It assigns all vertices absolute coordinates and assigns all edges coordinates, they have to pass through. LayoutAlgorithms can be registered with a `LayoutOption` (in 7.1.10, page 38) at a `LayoutRegister` (in 7.1.5, page 32).

7.1.4.1 Declaration

```
public interface LayoutAlgorithm
```

7.1.4.2 Methods

- **getSettings**

```
parameter.Settings getSettings()
```

- **Description**

  Get the set of parameters for this instance of the algorithm.

- **Returns** – the set of parameters

- **layout**

```
void layout(graphmodel.Graph graph)
```

- **Description**

  Layout the specified Graph.
- **Parameters**
  - `graph` – the graph to layout

7.1.5 Interface LayoutRegister

Stores a collection of layouts for a specific graph. This allows the client to select fitted layouts for every graph.

7.1.5.1 Declaration

```
public interface LayoutRegister
```

7.1.5.2 Methods

- **addLayoutOption**

  ```
  void addLayoutOption(LayoutOption option)
  ```

  - **Description**

    Adds a layout option to this register.
  - **Parameters**
    - `option` – the layout option
- **getLayoutOptions**

  ```
  java.util.List getLayoutOptions()
  ```

  - **Description**

    Returns all available layouts for the graph associated with this register.
  - **Returns** – the available layouts

7.1.6 Interface Plugin

This is the main entry point for plugins. Plugins have to register their content extensions via this interface. The plugin manager will load classes implementing this interface with a service loader, when they are correctly described in the plugins META-INF.

7.1.6.1 Declaration

```
public interface Plugin
```

7.1.6.2 Methods

- **getEdgeFilter**

  ```
  java.util.List getEdgeFilter()
  ```

  - **Description**

    Returns all by the plugin provided `EdgeFilter` (in 7.1.8, page 35). If none are provided returns `null` or an empty list.
  - **Returns** – the list of provided edge filter

- **getExporter**

  ```
  java.util.List getExporter()
  ```

  - **Description**

    Returns all by the plugin provided `Exporter` (in 7.1.2, page 29). If none are provided returns `null` or an empty list.
  - **Returns** – a list of provided exporter

- **getImporter**

  ```
  java.util.List getImporter()
  ```

  - **Description**

    Returns all by the plugin provided `Importer` (in 7.1.3, page 30). If none are provided returns `null` or an empty list.
  - **Returns** – a list of provided importer

- **getName**

  ```
  java.lang.String getName()
  ```

  - **Description**

    Returns the name of the plugin. Uniqueness can't be assumed.
  - **Returns** – the name of the plugin

- **getVertexFilter**

```
java.util.List getVertexFilter()
```

- **Description**

  Returns all by the plugin provided `VertexFilter` (in 7.1.12, page 40). If none are provided returns `null` or an empty list.

- **Returns** – the list of provided vertex filter

- **getWorkspaceOptions**

```
java.util.List getWorkspaceOptions()
```

- **Description**

  Returns all provided by the plugin `WorkspaceOption` (in 7.1.13, page 41). If none are provided returns `null` or an empty list.

- **Returns** – The list of provided workspace options

- **load**

```
void load()
```

- **Description**

  Called after all plugins have been constructed. Ïnter-Plugincommunication, like registering of layouts for graphs in other plugins should be executed in here.

7.1.7 Interface Workspace

A workspace contains a set of default actions and options for displaying a specific domain of graphs. The workspace manages the graphs instantiation through providing an `IGraphModelBuilder` (in 7.3.7, page 66). He also provides a list of layout options for graphs in his model.

7.1.7.1 Declaration

```
public interface Workspace
```

7.1.7.2 Methods

- **getGraphModel**

```
graphmodel.GraphModel getGraphModel()
```

- **Description**

  Returns the `GraphModel` (in 7.3.21, page 88) stored in the workspace.

- **Returns** – the graph model

- **getGraphModelBuilder**

  ```
  graphmodel.IGraphModelBuilder getGraphModelBuilder()
  ```

  - **Description**

    Returns a builder to build a graph model in this workspace.
  - **Returns** – the builder

- **getSettings**

  ```
  parameter.Settings getSettings()
  ```

  - **Description**

    Returns a set of parameters to initialize this workspace. When the settings have been adjusted, the client has to call `initialize()`. To initialize the workspace with the settings.
  - **Returns** – the settings

- **initialize**

  ```
  void initialize()
  ```

  - **Description**

    Initializes this workspace with the settings if they have not been adjusted. If the settings have not been adjusted, default values will be used.

7.1.8 Class EdgeFilter

This class represents a filter for edges. To check if an edge passes through this filter, the client can specify it in `matches(Edge edge)`.

7.1.8.1 Declaration

```
public abstract class EdgeFilter
 extends java.lang.Object
```

7.1.8.2 Constructors

- **EdgeFilter**

  ```
  public EdgeFilter()
  ```

7.1.8.3 Methods

- **getName**

  ```
  public java.lang.String getName()
  ```

  - **Description**

    Returns the name of the filter.
  - **Returns** – the name of the filter

- **matches**

  ```
  public abstract boolean matches(graphmodel.Edge toMatch)
  ```

  - **Description**

    This method checks if an edge matches this Filter. It will compare specified parameters of the edge with the defined parameters of this filter.
  - **Parameters**
    - `toMatch` – the edge which should be checked
  - **Returns** – true if the edge matches this Filter, otherwise false

- **setName**

  ```
  public void setName(java.lang.String name)
  ```

  - **Description**

    Sets the name of the filter.
  - **Parameters**
    - `name` – the name of the filter

7.1.9 Class EntryPointOption

An entry point option is an abstract superclass for all entry points, where the user can choose one of multiple entry points. Entry point options should allow a categorical overview for the entry point, to enable the client to differentiate the option from other options. Additionally a method for should be provided which can be executed when the client wants to select an option.

7.1.9.1 Declaration

```
public abstract class EntryPointOption
 extends java.lang.Object
```

7.1.9.2 All known subclasses

WorkspaceOption (in 7.1.13, page 41), LayoutOption (in 7.1.10, page 38)

7.1.9.3 Constructors

- **EntryPointOption**

```
public EntryPointOption()
```

7.1.9.4 Methods

- **getID**

```
public java.lang.String getID()
```

  - **Description**

    The id of an entry point should be an acronym of it's name.
  - **Returns** – the id

- **getName**

```
public java.lang.String getName()
```

  - **Description**

    Returns the name of the entry point option. This name can be displayed when displaying multiple options
  - **Returns** – the name of the entry point

- **setID**

```
protected void setID(java.lang.String id)
```

  - **Description**

    Sets the id of the entry point. The id should be an acronym of the name.
  - **Parameters**
    - `id` – the id of the entry point.

- **setName**

```
protected void setName(java.lang.String name)
```

  - **Description**

    Sets the name of the entry point
  - **Parameters**
    - `name` – the name of the entry point

7.1.10 Class LayoutOption

An option for a layout of a specific graph. Workspaces can return these for a specific graph. The client can then decide between one ore more LayoutOptions When selected the layout will be applied to the graph.

7.1.10.1 Declaration

```
public abstract class LayoutOption
 extends plugin.EntryPointOption implements java.lang.Cloneable
```

7.1.10.2 Constructors

- **LayoutOption**

  ```
  public LayoutOption()
  ```

7.1.10.3 Methods

- **applyLayout**

  ```
  public abstract void applyLayout()
  ```

  - **Description**

    This should execute the layout on the graph, which should be specified on construction, or in beforehand. The settings, which are accessible over `getSettings()` will be used to instantiate the LayoutAlgorithm.

- **chooseLayout**

  ```
  public abstract void chooseLayout()
  ```

  - **Description**

    Called when this layout option is chosen. This allows the layout option to prepare the actual LayoutAlgorithm.

- **getSettings**

  ```
  public abstract parameter.Settings getSettings()
  ```

  - **Description**

    Get the set of parameters for an algorithm of this option. `choose()` has to be called up front.

  - **Returns** – the set of parameters

7.1.11 Class PluginManager

The plugin manager manages the access of the main application to the plugins. It loads all plugins at the start of the runtime. When the client needs some service implemented by plugins, it can get a list of all available options.

7.1.11.1 Declaration

```
public class PluginManager
 extends java.lang.Object
```

7.1.11.2 Methods

- **getEdgeFilter**

  ```
  public java.util.List getEdgeFilter()
  ```

  - **Description**

    Returns a list of all edge filter provided by plugins.
  - **Returns** – a list of all edge filter

- **getExporter**

  ```
  public java.util.List getExporter()
  ```

  - **Description**

    Returns the `Exporter` (in 7.1.2, page 29) provided by all plugins.
  - **Returns** – a list of provided exporter

- **getImporter**

  ```
  public java.util.List getImporter()
  ```

  - **Description**

    Returns the `Importer` (in 7.1.3, page 30) provided by all plugins.
  - **Returns** – a list of provided importer

- **getPluginManager**

  ```
  public static PluginManager getPluginManager()
  ```

  - **Description**

    Returns the singleton instance of the plugin manager.

- **Returns** – the plugin manager

- **getPlugins**

```
public java.util.List getPlugins()
```

  - **Description**

    Returns a list of all plugins loaded by the ServiceLoader.
  - **Returns** – all loaded plugins

- **getVertexFilter**

```
public java.util.List getVertexFilter()
```

  - **Description**

    Returns all vertex filter provided by plugins.
  - **Returns** – a list of all vertex filter

- **getWorkspaceOptions**

```
public java.util.List getWorkspaceOptions()
```

  - **Description**

    Returns all `WorkspaceOption` (in 7.1.13, page 41)s provided by plugins.
  - **Returns** – a list of all workspace options

7.1.12 Class VertexFilter

This Class represents a filter for vertex types. The type of the vertex can be specified through different parameters.

7.1.12.1 Declaration

```
public class VertexFilter
 extends java.lang.Object
```

7.1.12.2 Constructors

- **VertexFilter**

```
public VertexFilter()
```

7.1.12.3 Methods

- **getName**

```
public java.lang.String getName()
```

  - **Description**

    Getter of name

- **matches**

```
public boolean matches(graphmodel.Vertex toMatch)
```

  - **Description**

    This method checks if an vertex matches this Filter. It will compare specified parameters of the vertex with the defined parameters of this filter.

  - **Parameters**

    * `toMatch` – the vertex which should be checked

  - **Returns** – true if the edge matches this Filter, otherwise false

- **setName**

```
public void setName(java.lang.String name)
```

  - **Description**

    Setter of name

7.1.13 Class WorkspaceOption

This is an option for a workspace to set and change settings on it.

7.1.13.1 Declaration

```
public abstract class WorkspaceOption
 extends plugin.EntryPointOption
```

7.1.13.2 Constructors

- **WorkspaceOption**

```
public WorkspaceOption()
```

7.1.13.3 Methods

- **getInstance**

```
public abstract Workspace getInstance()
```

  - **Description**

    Creates an instance of the workspace with the earlier adjusted Settings

  - **Returns** – a workspace

- **getSettings**

```
public abstract parameter.Settings getSettings()
```

  - **Description**

    Returns a set of parameters to initialize a workspace for this. When the settings have been adjusted, the client has to call `getInstance()`, to get an instance of the workspace with the settings.

  - **Returns** – the settings

7.2 Package gui

Package Contents *Page*

Classes

7.2.1 Class EdgeShape

A visual representation of an edge with a text.

7.2.1.1 Declaration

```
public class EdgeShape
 extends gui.GAnsGraphElement
```

7.2.1.2 Constructors

- **EdgeShape**

  ```
  public EdgeShape(graphmodel.Edge edge)
  ```

  - **Description**

    Constructs a EdgeShape with the information supplied by edge. The path of the EdgeShape is set by the EdgePath of edge.

  - **Parameters**
    - `edge` – The Edge that supplies the information for building an EdgeShape.

7.2.1.3 Methods

- **getText**

  ```
  public abstract java.lang.String getText()
  ```

  - **Description copied from GAnsGraphElement** (**in 7.2.3, page 45**)

    Returns the text shown on the element.

  - **Returns** – The text that is being displayed on the element.

- **setColor**

  ```
  public abstract void setColor(javafx.scene.paint.Color color)
  ```

- **Description copied from GAnsGraphElement** (**in 7.2.3, page 45**)
  Sets the color of the element.
- **Parameters**
  * `color` – The color the element will be displayed in.

- **setText**

```
public abstract void setText(java.lang.String text)
```

- **Description copied from GAnsGraphElement** (**in 7.2.3, page 45**)
  Sets the text shown on the element.
- **Parameters**
  * `text` – The text that will be displayed on the element.

7.2.2 Class GAnsApplication

Main application of GAns.

7.2.2.1 Declaration

```
public class GAnsApplication
 extends javafx.application.Application
```

7.2.2.2 Constructors

- **GAnsApplication**

```
public GAnsApplication()
```

7.2.2.3 Methods

- **main**

```
public static void main(java.lang.String[] args)
```

- **Description**
  Main method.
- **Parameters**
  * `args` – Arguments.

- **start**

```
public abstract void start(javafx.stage.Stage arg0) throws java.
    lang.Exception
```

7.2.3 Class GAnsGraphElement

A abstract class to generalize the visual elements that can be displayed in the `GraphView` (in 7.2.4, page 46).

7.2.3.1 Declaration

```
public abstract class GAnsGraphElement
 extends javafx.scene.layout.StackPane
```

7.2.3.2 All known subclasses

VertexShape (in 7.2.11, page 56), EdgeShape (in 7.2.1, page 43)

7.2.3.3 Constructors

- **GAnsGraphElement**

  ```
  public GAnsGraphElement()
  ```

7.2.3.4 Methods

- **getText**

  ```
  public abstract java.lang.String getText()
  ```

  - **Description**

    Returns the text shown on the element.
  - **Returns** – The text that is being displayed on the element.

- **setColor**

  ```
  public abstract void setColor(javafx.scene.paint.Color color)
  ```

  - **Description**

    Sets the color of the element.
  - **Parameters**
    - `color` – The color the element will be displayed in.

- **setText**

  ```
  public abstract void setText(java.lang.String text)
  ```

- **Description**

  Sets the text shown on the element.

- **Parameters**

  * `text` – The text that will be displayed on the element.

7.2.4 Class GraphView

A view used for showing and creating a graph in GAns. It supports zooming and other general navigation features.

7.2.4.1 Declaration

```
public class GraphView
 extends javafx.scene.layout.Pane
```

7.2.4.2 Constructors

- **GraphView**

  ```
  public GraphView()
  ```

  - **Description**

    Constructor.

7.2.4.3 Methods

- **addGrid**

  ```
  public void addGrid()
  ```

  - **Description**

    Adds a grid to the GraphView, on which the dragging can be mapped.

- **getFactory**

  ```
  public GraphViewGraphFactory getFactory()
  ```

  - **Description**

    Returns the current `GraphViewGraphFactory` (in 7.2.6, page 49) from the view.

  - **Returns** – The current `GraphViewGraphFactory` (in 7.2.6, page 49).

- **getScale**

`public double getScale()`

- **Description**
  Returns the scale on which the GraphView currently is.
- **Returns** – The scale of the GraphView.

- **getSelectionModel**

`public GraphViewSelectionModel getSelectionModel()`

- **Description**
  Returns the selection model of the GraphView.
- **Returns** – The selection model of the GraphView.

- **setGraph**

`public void setGraph(graphmodel.Graph graph)`

- **Description**
  Sets a graph. Every element in the graph will be generated and then shown.
- **Parameters**
  - `graph` – The graph to be visualized in the view.

- **setPivot**

`public void setPivot(double x,double y)`

- **Description**
  Sets the pivot so the scrolling follows the mouse position on the GraphView.
- **Parameters**
  - `x` – The x coordinate of the pivot.
  - `y` – The y coordinate of the pivot.

- **setScale**

`public void setScale(double scale)`

- **Description**
  Sets the scale of the GraphView.
- **Parameters**
  - `scale` – The scale of the GraphView.

- **setSelectionModel**

```
public void setSelectionModel(GraphViewSelectionModel
    selectionModel)
```

  - **Description**

    Sets the selection model for the GraphView.

  - **Parameters**

    * `selectionModel` – The selection model for the GraphView.

- **updateGraph**

```
public void updateGraph()
```

  - **Description**

    Updates the shown graph.

7.2.5 Class GraphViewEventHandler

GraphViewEventHandler provides listeners for making the `GraphView` (in 7.2.4, page 46) draggable and zoomable.

7.2.5.1 Declaration

```
public class GraphViewEventHandler
 extends java.lang.Object
```

7.2.5.2 Constructors

- **GraphViewEventHandler**

```
public GraphViewEventHandler(GraphView canvas)
```

  - **Description**

    Constructor.

  - **Parameters**

    * `canvas` – The `GraphView` (in 7.2.4, page 46) which later on will get the EventHandler set.

7.2.5.3 Methods

- **getOnMouseDraggedEventHandler**

  ```
  public javafx.event.EventHandler getOnMouseDraggedEventHandler()
  ```

  - **Description**

    Returns an EventHandler which handles dragging inside the `GraphView` (in 7.2.4, page 46).
  - **Returns** – An EventHandler which handles dragging.

- **getOnMousePressedEventHandler**

  ```
  public javafx.event.EventHandler getOnMousePressedEventHandler()
  ```

  - **Description**

    Returns an EventHandler which handles pressing the mouse inside the `GraphView` (in 7.2.4, page 46).
  - **Returns** – An EventHandler which handles pressing the mouse.

- **getOnScrollEventHandler**

  ```
  public javafx.event.EventHandler getOnScrollEventHandler()
  ```

  - **Description**

    Returns an EventHandler which maps scrolling the mousewheel to zooming on the `GraphView` (in 7.2.4, page 46).
  - **Returns** – An EventHandler which maps scrolling the mousewheel to zooming.

7.2.6 Class GraphViewGraphFactory

The GraphViewGraphFactory generates the visual representation of a given `Graph` (in 7.3.4, page 62) and gives access to the set `Graph` (in 7.3.4, page 62).

7.2.6.1 Declaration

```
public class GraphViewGraphFactory
 extends java.lang.Object
```

7.2.6.2 Constructors

- **GraphViewGraphFactory**

  ```
  public GraphViewGraphFactory(graphmodel.Graph graph)
  ```

- **Description**

  Constructor. Sets the graph and generates the vertices and edges for visualization.

- **Parameters**

  * `graph` – The graph data that will be shown.

7.2.6.3 Methods

- **getEdgeFromShape**

```
public graphmodel.Edge getEdgeFromShape(GAnsGraphElement shape)
```

  - **Description**

    Returns the edge element from the graph model that is being represented by the shape. Can be null if an `VertexShape` (in 7.2.11, page 56) is passed.

  - **Parameters**

    * `shape` – The shape that represents the edge.

  - **Returns** – The Edge being represented by the passed shape.

- **getGraphicalElements**

```
public java.util.LinkedList getGraphicalElements()
```

  - **Description**

    Returns all graphical elements that have been generated by the factory.

  - **Returns** – All graphical elements generated by the factory.

- **getSizeOfVertex**

```
public static javafx.util.Pair getSizeOfVertex(java.lang.String
    text)
```

  - **Description**

    Calculates and returns the size of a vertex with the given text.

  - **Parameters**

    * `text` – The text which size the vertex depends on.

  - **Returns** – A Pair of width and height of the vertex.

- **getVertexFromShape**

```
public graphmodel.Vertex getVertexFromShape(GAnsGraphElement
    shape)
```

– **Description**

Returns the vertex element from the graph model that is being represented by the shape. Can be null if an `EdgeShape` (in 7.2.1, page 43) is passed.

– **Parameters**

∗ `shape` – The shape that represents the vertex.

– **Returns** – The Vertex being represented by the passed shape.

7.2.7 Class GraphViewSelectionModel

The selection model for the `GraphView` (in 7.2.4, page 46), that supports multiple selection of vertices and edges.

7.2.7.1 Declaration

```
public class GraphViewSelectionModel
 extends javafx.scene.control.MultipleSelectionModel
```

7.2.7.2 Constructors

- **GraphViewSelectionModel**

```
public GraphViewSelectionModel()
```

7.2.7.3 Methods

- **clearAndSelect**

```
public abstract void clearAndSelect(int arg0)
```

- **clearSelection**

```
public abstract void clearSelection()
```

- **clearSelection**

```
public abstract void clearSelection(int arg0)
```

- **getSelectedIndices**

```
public abstract javafx.collections.ObservableList
    getSelectedIndices()
```

- **getSelectedItems**

```
public abstract javafx.collections.ObservableList
    getSelectedItems()
```

- **getSelectedItemsProperties**

```
public javafx.collections.ObservableList
    getSelectedItemsProperties()
```

  - **Description**

    Returns the `GAnsProperty` (in 7.7.2, page 172) of all selected items.

  - **Returns** – A list with all the `GAnsProperty` (in 7.7.2, page 172) of all selected items.

- **isEmpty**

```
public abstract boolean isEmpty()
```

- **isSelected**

```
public abstract boolean isSelected(int arg0)
```

- **select**

```
public void select(GAnsGraphElement obj)
```

- **select**

```
public abstract void select(int arg0)
```

- **selectAll**

```
public abstract void selectAll()
```

- **selectFirst**

```
public abstract void selectFirst()
```

- **selectIndices**

```
public abstract void selectIndices(int arg0,int[] arg1)
```

- **selectLast**

```
public abstract void selectLast()
```

- **selectNext**

```
public abstract void selectNext()
```

- **selectPrevious**

```
public abstract void selectPrevious()
```

7.2.8 Class InformationView

The InformationView shows a given set of properties from the selected vertices in the `GraphView` (in 7.2.4, page 46).

7.2.8.1 Declaration

```
public class InformationView
 extends javafx.scene.control.TableView
```

7.2.8.2 Constructors

- **InformationView**

```
public InformationView()
```

7.2.8.3 Methods

- **setInformations**

```
public void setInformations(javafx.collections.ObservableList
    informations)
```

  - **Description**

    Sets the properties which should be shown in the InformationView. The `GAnsProperty` (in 7.7.2, page 172) are being processed in an internal factory, which automatically creates the tablecells. The function will be called whenever the selection of the `GraphView` (in 7.2.4, page 46) changes.

  - **Parameters**

    * `informations` – List with `GAnsProperty` (in 7.7.2, page 172) elements which define the content of the InformationView

7.2.9 Class ParameterDialogGenerator

Generates a parameter dialog given a parent node and a set of parameters.

7.2.9.1 Declaration

```
public class ParameterDialogGenerator
 extends parameter.ParameterVisitor
```

7.2.9.2 Constructors

- **ParameterDialogGenerator**

  ```
  public ParameterDialogGenerator(javafx.scene.layout.GridPane
      parent,parameter.Settings settings)
  ```

  - **Description**

    Constructs a new ParameterDialogGenerator and sets the parent, where all parameter GUI-Elements are placed in afterwards.

7.2.9.3 Methods

- **visit**

  ```
  public abstract void visit(parameter.IntegerParameter parameter)
  ```

  - **Description copied from parameter.ParameterVisitor** (**in 7.5.6, page 133**)

    Visits the specified parameter and performs some by the subclass chosen actions on it.

  - **Parameters**

    * `parameter` – The parameter to visit

- **visit**

  ```
  public abstract void visit(parameter.MultipleChoiceParameter
      parameter)
  ```

  - **Description copied from parameter.ParameterVisitor** (**in 7.5.6, page 133**)

    Visits the specified parameter and performs some by the subclass chosen actions on it.

  - **Parameters**

    * `parameter` – The parameter to visit

- **visit**

```
public abstract void visit(parameter.StringParameter parameter)
```

  - **Description copied from parameter.ParameterVisitor** (**in 7.5.6, page 133**)

    Visits the specified parameter and performs some by the subclass chosen actions on it.
  - **Parameters**
    - `parameter` – The parameter to visit

7.2.10 Class StructureView

The StructureView regulates the access and representation of the elements in the StructureView of GAns.

7.2.10.1 Declaration

```
public class StructureView
 extends javafx.scene.control.TreeView
```

7.2.10.2 Constructors

- **StructureView**

```
public StructureView()
```

  - **Description**

    Constructor.

7.2.10.3 Methods

- **getIdOfSelectedItem**

```
public java.lang.String getIdOfSelectedItem()
```

  - **Description**

    Returns the id of the selected graph.
  - **Returns** – The id of the selected graph.

- **showTree**

```
public void showTree(graphmodel.Graph graph)
```

- **Description**

  Creates a tree like representation from a given graph and its subgraphs. Should be called, before calling other methods, because there could be a dummy root-node in the View.

- **Parameters**

  * `graph` – The graph which should be represented.

7.2.11 Class VertexShape

A visual representation of a vertex with a text inside of it.

7.2.11.1 Declaration

```
public class VertexShape
 extends gui.GAnsGraphElement
```

7.2.11.2 Constructors

- **VertexShape**

  ```
  public VertexShape()
  ```

  - **Description**

    Constructor

- **VertexShape**

  ```
  public VertexShape(java.lang.String text)
  ```

  - **Description**

    Constructor which directly sets the text.

  - **Parameters**

    * `text` – The text that will be displayed in the vertex.

7.2.11.3 Methods

- **getText**

  ```
  public abstract java.lang.String getText()
  ```

  - **Description copied from GAnsGraphElement** (**in 7.2.3, page 45**)

    Returns the text shown on the element.

  - **Returns** – The text that is being displayed on the element.

- **setColor**

  ```
  public abstract void setColor(javafx.scene.paint.Color color)
  ```

  - **Description copied from GAnsGraphElement** (**in 7.2.3, page 45**)
    Sets the color of the element.
  - **Parameters**
    * `color` – The color the element will be displayed in.

- **setText**

  ```
  public abstract void setText(java.lang.String text)
  ```

  - **Description copied from GAnsGraphElement** (**in 7.2.3, page 45**)
    Sets the text shown on the element.
  - **Parameters**
    * `text` – The text that will be displayed on the element.

7.3 Package graphmodel

Package Contents *Page*

Interfaces

Classes

7.3.1 Interface CompoundVertex

A interface of a vertex that contains an entire subgraph.

7.3.1.1 Declaration

```
public interface CompoundVertex
 extends Vertex
```

7.3.1.2 Methods

- **getConnectedVertex**

  ```
  Vertex getConnectedVertex(Edge edge)
  ```

  - **Description**

    Returns the connected vertex contained in the compound vertex for a given edge, where one end point of the edge has to be this vertex.

  - **Parameters**

    * `edge` – the edge to get the vertex to

  - **Returns** – the connected vertex if the edge is valid

- **getGraph**

  ```
  Graph getGraph()
  ```

  - **Description**

    Returns the graph contained in the vertex.

  - **Returns** – the graph contained in the vertex.

7.3.2 Interface DirectedGraph

A `DirectedGraph` (in 7.3.2, page 59) is a specific Graph which contains just `DirectedEdge` (in 7.3.15, page 80) as edges

7.3.2.1 Declaration

```
public interface DirectedGraph
 extends Graph
```

7.3.2.2 All known subinterfaces

LayeredGraph (in 7.3.9, page 68), DefaultDirectedGraph (in 7.3.13, page 75)

7.3.2.3 All classes known to implement interface

DefaultDirectedGraph (in 7.3.13, page 75)

7.3.2.4 Methods

- **incomingEdgesOf**

```
java.util.Set incomingEdgesOf(Vertex vertex)
```

  - **Description**

    Returns a set of all incoming edges of a vertex.
  - **Parameters**
    - `vertex` – Vertex whose incoming edges will be returned.
  - **Returns** – The edges coming in the supplied vertex.

- **indegreeOf**

```
java.lang.Integer indegreeOf(Vertex vertex)
```

  - **Description**

    Returns the indegree of a vertex of the graph.
  - **Parameters**
    - `vertex` – Vertex whose indegree will be returned.
  - **Returns** – The number of edges going into the supplied vertex.

- **outdegreeOf**

```
java.lang.Integer outdegreeOf(Vertex vertex)
```

  - **Description**

    Returns the outdegree of a vertex of the graph.
  - **Parameters**
    - `vertex` – Vertex whose outdegree will be returned.
  - **Returns** – The number of edges going out of the supplied vertex.

- **outgoingEdgesOf**

```
java.util.Set outgoingEdgesOf(Vertex vertex)
```

  - **Description**

    Returns a set of all outgoing edges of a vertex.

- **Parameters**
  - `vertex` – Vertex whose outgoing edges will be returned.
- **Returns** – The edges going out of the supplied vertex.

7.3.3 Interface Edge

This edge interface specifies an edge. An edge contains two vertices, an ID, a name and a label.

7.3.3.1 Declaration

```
public interface Edge
```

7.3.3.2 All known subinterfaces

SerializedEdge (in 7.3.24, page 90), DirectedEdge (in 7.3.15, page 80)

7.3.3.3 All classes known to implement interface

SerializedEdge (in 7.3.24, page 90), DirectedEdge (in 7.3.15, page 80)

7.3.3.4 Methods

- **addToFastGraphAccessor**

  ```
  void addToFastGraphAccessor(FastGraphAccessor fga)
  ```

  - **Description**

    Adds the edge to a `FastGraphAccessor` (in 7.3.20, page 85).
  - **Parameters**
    - `fga` – The `FastGraphAccessor` (in 7.3.20, page 85) to whom this edge will be added.

- **getID**

  ```
  java.lang.Integer getID()
  ```

  - **Description**

    Returns the ID of the edge.
  - **Returns** – The id of the edge.

- **getLabel**

  ```
  java.lang.String getLabel()
  ```

- **Description**

  Returns the label of the edge.

- **Returns** – The label of the edge.

- **getName**

```
java.lang.String getName()
```

  - **Description**

    Returns the name of the edge.

  - **Returns** – The name of the edge.

- **getPath**

```
EdgePath getPath()
```

  - **Description**

    Returns the `EdgePath` (in 7.3.18, page 83) of the edge. The edge path is attached to the edge and cannot be replaced.

  - **Returns** – the edge path

- **getVertices**

```
java.util.List getVertices()
```

  - **Description**

    Returns the vertices connected with this edge.

  - **Returns** – The vertices connected with the edge.

- **serialize**

```
SerializedEdge serialize(java.util.List attributes)
```

  - **Description**

    Returns a `SerializedEdge` (in 7.3.24, page 90) representation of the edge.

  - **Parameters**

    * `attributes` – The attributes that have to be serialized.

  - **Returns** – The `SerializedEdge` (in 7.3.24, page 90) representation of the edge.

7.3.4 Interface Graph

This graph interface specifies a graph. A graph contains edges and vertices.

7.3.4.1 Declaration

```
public interface Graph
```

7.3.4.2 All known subinterfaces

ViewableGraph (in 7.3.12, page 74), SerializedGraph (in 7.3.25, page 92), LayeredGraph (in 7.3.9, page 68), DirectedGraph (in 7.3.2, page 59), DefaultDirectedGraph (in 7.3.13, page 75)

7.3.4.3 All classes known to implement interface

SerializedGraph (in 7.3.25, page 92)

7.3.4.4 Methods

- **addToFastGraphAccessor**

  ```
  void addToFastGraphAccessor(FastGraphAccessor fga)
  ```

  - **Description**

    Adds the graph to a `FastGraphAccessor` (in 7.3.20, page 85).
  - **Parameters**
    - `fga` – the `FastGraphAccessor` (in 7.3.20, page 85) to whom this graph will be added.

- **edgesOf**

  ```
  java.util.Set edgesOf(Vertex vertex)
  ```

  - **Description**

    Returns a list of all edges of a vertex.
  - **Parameters**
    - `vertex` – the vertex which edges will be returned.
  - **Returns** – All edges which are connected with the supplied vertex.

- **getEdgeSet**

  ```
  java.util.Set getEdgeSet()
  ```

  - **Description**

    Returns all edges of the graph.
  - **Returns** – A set of all edges of the graph.

- **getFastGraphAccessor**

```
FastGraphAccessor getFastGraphAccessor()
```

  - **Description**
    Returns the FastGraphAccessor of this Graph.
  - **Returns** – the FastGraphAccessor of this Graph

- **getID**

```
java.lang.Integer getID()
```

  - **Description**
    Returns the ID of the graph.
  - **Returns** – The id of the graph.

- **getName**

```
java.lang.String getName()
```

  - **Description**
    Returns the name of the Graph.
  - **Returns** – The name of the graph.

- **getVertexSet**

```
java.util.Set getVertexSet()
```

  - **Description**
    Returns all vertices of the graph.
  - **Returns** – A set of all vertices of the graph.

7.3.5 Interface IEdgeBuilder

An abstract interface, which is used to build one edge.

7.3.5.1 Declaration

```
public interface IEdgeBuilder
```

7.3.5.2 Methods

- **addData**

```
void addData(java.lang.String keyname,java.lang.String value)
```

- **Description**

  Adds additional data to this edge. The specific EdgeBuilder implementation needs to decide how to save the value for given edge type.

- **Parameters**
  - `keyname` – Name of the attribute
  - `value` – Value of the attribute

- **build**

```
Edge build()
```

  - **Description**

    . Builds an Edge with the given Data and returns it.

  - **Returns** – The Edge that is being build by the IEdgeBuilder

- **newEdge**

```
void newEdge(java.lang.String source,java.lang.String target)
```

  - **Description**

    Sets source and target vertices of the edge build by this.

  - **Parameters**
    - `source` – String representation of the source vertex as ID
    - `target` – String representation of the target vertex as ID

- **setDirection**

```
void setDirection(java.lang.String direction)
```

  - **Description**

    Sets the direction of the edge build by this.

  - **Parameters**
    - `direction` – String representation of the direction. Can be one of

- **setID**

```
void setID(java.lang.String id)
```

  - **Description**

    Sets the ID of the edge build by this.

  - **Parameters**
    - `id` – value to which the id is set

7.3.6 Interface IGraphBuilder

An abstract interface, which is used to build a graph.

7.3.6.1 Declaration

```
public interface IGraphBuilder
```

7.3.6.2 Methods

- **build**

  ```
  Graph build()
  ```

  - **Description**
    Builds a graph from the given settings and returns it.
  - **Returns** – The graph that is being build by the IGraphBuilder.

- **getEdgeBuilder**

  ```
  IEdgeBuilder getEdgeBuilder()
  ```

  - **Description**
    Returns the EdgeBuilder which is specified for this graph.
  - **Returns** – The `IEdgeBuilder` (in 7.3.5, page 64) which is specified for this graph.

- **getVertexBuilder**

  ```
  IVertexBuilder getVertexBuilder(java.lang.String vertexID)
  ```

  - **Description**
    Returns the VertexBuilder which is specified for this graph.
  - **Parameters**
    * `vertexID` – The id of the vertex which associated IVertexBuilder will be returned.
  - **Returns** – The `IVertexBuilder` (in 7.3.8, page 67) which is specified for this graph.

7.3.7 Interface IGraphModelBuilder

An abstract interface, which is used to build a graphmodel. This class is based on the Builder Pattern.

7.3.7.1 Declaration

```
public interface IGraphModelBuilder
```

7.3.7.2 Methods

- **build**

  ```
  GraphModel build()
  ```

  - **Description**

    Builds a graphmodel from the given settings and returns it.

  - **Returns** – The `GraphModel` (in 7.3.21, page 88) that is being build by the IGraphModelBuilder.

- **getGraphBuilder**

  ```
  IGraphBuilder getGraphBuilder(java.lang.String graphID)
  ```

  - **Description**

    Returns a specific `IGraphBuilder` (in 7.3.6, page 66) for a graph, which belongs to the `GraphModel` (in 7.3.21, page 88).

  - **Parameters**

    * `graphID` – The id of the graph which associated `IGraphBuilder` (in 7.3.6, page 66) will be returned.

  - **Returns** – The IGraphBuilder of the graph which is referenced over the graphID.

7.3.8 Interface IVertexBuilder

An abstract interface, which is used to build one vertex.

7.3.8.1 Declaration

```
public interface IVertexBuilder
```

7.3.8.2 Methods

- **addData**

  ```
  void addData(java.lang.String keyname, java.lang.String value)
  ```

  - **Description**

    Add Data to this Vertex. The IVertexBuilder needs to parse the data and add it to the edge

  - **Parameters**
    - `keyname` – Name of the attribute which is added
    - `value` – Value of the attribute

- **build**

```
Vertex build()
```

  - **Description**

    . Builds a Vertex with the given Data and returns it.
  - **Returns** – The Vertex that is being build by the IVertexBuilder

- **getGraphBuilder**

```
IGraphBuilder getGraphBuilder(java.lang.String graphID)
```

  - **Description**

    This method returns an specific GraphBuilder. This method is used to implement nested Graphs.
  - **Parameters**
    - `graphID` – The id of the graph which associated `IGraphBuilder` (in 7.3.6, page 66) will be returned.
  - **Returns** – The IGraphBuilder of the graph which is referenced over the graphID.

- **setID**

```
void setID(java.lang.String id)
```

  - **Description**

    Sets the ID of the vertex build by this.
  - **Parameters**
    - `id` – value to which the id is set

7.3.9 Interface LayeredGraph

A DirectedGraph which in addition to coordinates saves the relative position of all vertices in a layered structure. Every vertex is in a layer. Every layer is sorted so that every node has zero to two horizontal neighbors.

7.3.9.1 Declaration

```
public interface LayeredGraph
 extends DirectedGraph
```

7.3.9.2 Methods

- **getHeight**

  ```
  int getHeight()
  ```

  - **Description**

    Returns the height, i.e. the number of layers.
  - **Returns** – the height

- **getLayer**

  ```
  java.util.List getLayer(int layerNum)
  ```

  - **Description**

    Get all vertices from a certain layer.
  - **Parameters**
    - `layerNum` – the index of the layer
  - **Returns** – a list of all vertices which are on this layer

- **getLayer**

  ```
  int getLayer(Vertex vertex)
  ```

  - **Description**

    Get the layer from the vertex
  - **Parameters**
    - `vertex` – the vertex to get its layer from
  - **Returns** – the layer number from this vertex

- **getLayerCount**

  ```
  int getLayerCount()
  ```

  - **Description**

    Get the amount of layers.
  - **Returns** – the amount of layers that contain at least one vertex

- **getLayers**

  ```
  java.util.List getLayers()
  ```

  - **Description**

    Get all layers that contain vertices.

  - **Returns** – a list of lists of vertices which are on this layer

- **getLayerWidth**

  ```
  int getLayerWidth(int layerN)
  ```

  - **Description**

    Returns the width of the layer specified by its index, i.e. the number of vertices in the layer.

  - **Parameters**

    * `layerN` – the index of the layer

  - **Returns** – the width of the layer

- **getMaxWidth**

  ```
  int getMaxWidth()
  ```

  - **Description**

    Returns the width of the widest layer, i.e. the number of vertices the layer with the most vertices contains.

  - **Returns** – the maximum width

- **getSubgraphs**

  ```
  java.util.List getSubgraphs()
  ```

  - **Description**

    Returns all subgraphs contained in this graph. All subgraphs of layered graphs have to be layered graphs with equal parameters themselves.

  - **Returns** – subgraphs in this layered graph

- **getVertexCount**

  ```
  int getVertexCount(int layerNum)
  ```

- **Description**

  Get the number of vertices which are on a certain layer

- **Parameters**
  - `layerNum` – the layer number to get the vertex count from
- **Returns** – the number of vertices which are on this layer

7.3.10 Interface Vertex

This vertex interface specifies a vertex. Every vertex contains an ID, a name and a label. The ID of a vertex is unique.

7.3.10.1 Declaration

```
public interface Vertex
```

7.3.10.2 All known subinterfaces

SerializedVertex (in 7.3.26, page 94), DefaultVertex (in 7.3.14, page 78), CompoundVertex (in 7.3.1, page 59)

7.3.10.3 All classes known to implement interface

SerializedVertex (in 7.3.26, page 94), DefaultVertex (in 7.3.14, page 78)

7.3.10.4 Methods

- **addToFastGraphAccessor**

  ```
  void addToFastGraphAccessor(FastGraphAccessor fga)
  ```

  - **Description**

    Adds the vertex to a `FastGraphAccessor` (in 7.3.20, page 85).

  - **Parameters**
    - `fga` – The `FastGraphAccessor` (in 7.3.20, page 85) to whom this vertex will be added.

- **getID**

  ```
  java.lang.Integer getID()
  ```

  - **Description**

    Returns the ID of the vertex. Every vertex in one graph has a unique ID.

  - **Returns** – The ID of the vertex.

- **getLabel**

  ```
  java.lang.String getLabel()
  ```

  - **Description**

    Returns the label of the vertex, that will be shown in the GUI. The label can be an empty string.
  - **Returns** – The label of the vertex

- **getName**

  ```
  java.lang.String getName()
  ```

  - **Description**

    Returns the name of the vertex. A descriptive name of the vertex. Multiple vertices with equal name in one graph are allowed. Therefore don't use this as identifier, instead use `getID()`.
  - **Returns** – The name of the vertex.

- **getX**

  ```
  int getX()
  ```

  - **Description**

    Returns the X-coordinate of the vertex.
  - **Returns** – The X-coordinate of this vertex.

- **getY**

  ```
  int getY()
  ```

  - **Description**

    Returns the Y-coordinate of the vertex.
  - **Returns** – The Y-coordinate of the vertex.

- **serialize**

  ```
  SerializedVertex serialize(java.util.List attributes)
  ```

  - **Description**

    Returns a `SerializedVertex` (in 7.3.26, page 94) representation of the graph.
  - **Parameters**
    - `attributes` – The attributes that have to be serialized.
  - **Returns** – The `SerializedVertex` (in 7.3.26, page 94) representation of the graph.

7.3.11 Interface Viewable

Adds methods for manipulation and observation of graphs. This interface differentiates between domain specific graphs, which can be viewed (children of ViewableGraph), and utility graphs like `SerializedGraph` (in 7.3.25, page 92) and SugiyamaGraph

7.3.11.1 Declaration

```
public interface Viewable
```

7.3.11.2 All known subinterfaces

ViewableGraph (in 7.3.12, page 74), DefaultDirectedGraph (in 7.3.13, page 75)

7.3.11.3 Methods

- **collapse**

  ```
  CompoundVertex collapse(java.util.Set subset)
  ```

  - **Description**

    Collapses a set of vertices in one compound vertex. The collapsed vertices can be expanded back into their previous state with `expand(CompoundVertex)`.

  - **Parameters**
    - `subset` – the subset to collapse
  - **Returns** – the resulting collapsed vertex

- **expand**

  ```
  java.util.Set expand(CompoundVertex vertex)
  ```

  - **Description**

    Expands a collapsed vertex into its substituted set of vertices The vertices will be added back to the set of vertices of this graph. The compound vertex will be removed from the set of vertices. All to the compound vertex incident edges, will be resolved back into an edge between the vertices it connected before the collapse.

  - **Parameters**
    - `vertex` – the collapsed vertex to expand
  - **Returns** – the set of vertices which was substituted by the collapsed vertex

- **getDefaultLayout**

  ```
  plugin.LayoutOption getDefaultLayout()
  ```

- **Description**

  Returns the default layout for this graph. This can be called when to quickly get a suiting layout without having to decide between multiple options.

- **Returns** – the default layout for this graph

- **getRegisteredLayouts**

```
java.util.List getRegisteredLayouts()
```

  - **Description**

    Returns a list of layouts which have been registered at the corresponding LayoutRegister for this graph type. The graph implementing this interface will be set as target of the LayoutOption.

  - **Returns** – A list of layouts which have been registered at the corresponding LayoutRegister for this graph type.

- **isCompound**

```
boolean isCompound(Vertex vertex)
```

  - **Description**

    Returns true if the specified vertex is a compound vertex

  - **Parameters**

    * `vertex` – the vertex to check

  - **Returns** – true if the vertex is a compound, false otherwise

- **serialize**

```
SerializedGraph serialize(java.util.List attributes)
```

  - **Description**

    Returns a `SerializedGraph` (in 7.3.25, page 92) representation of the graph.

  - **Parameters**

    * `attributes` – The attributes that have to be serialized.

  - **Returns** – The `SerializedGraph` (in 7.3.25, page 92) representation of the graph.

7.3.12 Interface ViewableGraph

The base graph accessed by the UI.

7.3.12.1 Declaration

```
public interface ViewableGraph
 extends Viewable, Graph
```

7.3.12.2 All known subinterfaces

DefaultDirectedGraph (in 7.3.13, page 75)

7.3.12.3 All classes known to implement interface

DefaultDirectedGraph (in 7.3.13, page 75)

7.3.13 Class DefaultDirectedGraph

A `DefaultDirectedGraph` (in 7.3.13, page 75) is a specific Graph which only contains `DirectedEdge` (in 7.3.15, page 80) as edges.

7.3.13.1 Declaration

```
public class DefaultDirectedGraph
 extends java.lang.Object implements DirectedGraph, ViewableGraph
```

7.3.13.2 Constructors

- **DefaultDirectedGraph**

  ```
  public DefaultDirectedGraph()
  ```

7.3.13.3 Methods

- **addToFastGraphAccessor**

  ```
  public void addToFastGraphAccessor(FastGraphAccessor fga)
  ```

- **collapse**

  ```
  public CompoundVertex collapse(java.util.Set subset)
  ```

- **edgesOf**

  ```
  public java.util.Set edgesOf(Vertex vertex)
  ```

- **expand**

```
public java.util.Set expand(CompoundVertex vertex)
```

- **getDefaultLayout**

```
public plugin.LayoutOption getDefaultLayout()
```

- **getEdgeSet**

```
public java.util.Set getEdgeSet()
```

- **getFastGraphAccessor**

```
public FastGraphAccessor getFastGraphAccessor()
```

- **getID**

```
public java.lang.Integer getID()
```

- **getName**

```
public java.lang.String getName()
```

- **getRegisteredLayouts**

```
public java.util.List getRegisteredLayouts()
```

- **getSource**

```
public Vertex getSource(DirectedEdge edge)
```

  - **Description**
    Returns the source of a edge of the graph.
  - **Parameters**
    - `edge` – A edge which is contained in the graph.
  - **Returns** – The vertex which the edge is coming from.

- **getVertexSet**

```
public java.util.Set getVertexSet()
```

- **incomingEdgesOf**

```
java.util.Set incomingEdgesOf(Vertex vertex)
```

  - **Description copied from DirectedGraph** (**in 7.3.2, page 59**)
    Returns a set of all incoming edges of a vertex.
  - **Parameters**
    - `vertex` – Vertex whose incoming edges will be returned.
  - **Returns** – The edges coming in the supplied vertex.

- **indegreeOf**

```
java.lang.Integer indegreeOf(Vertex vertex)
```

  - **Description copied from DirectedGraph** (**in 7.3.2, page 59**)
    Returns the indegree of a vertex of the graph.
  - **Parameters**
    - `vertex` – Vertex whose indegree will be returned.
  - **Returns** – The number of edges going into the supplied vertex.

- **isCompound**

```
public boolean isCompound(Vertex vertex)
```

- **outdegreeOf**

```
java.lang.Integer outdegreeOf(Vertex vertex)
```

  - **Description copied from DirectedGraph** (**in 7.3.2, page 59**)
    Returns the outdegree of a vertex of the graph.
  - **Parameters**
    - `vertex` – Vertex whose outdegree will be returned.
  - **Returns** – The number of edges going out of the supplied vertex.

- **outgoingEdgesOf**

```
java.util.Set outgoingEdgesOf(Vertex vertex)
```

  - **Description copied from DirectedGraph** (**in 7.3.2, page 59**)
    Returns a set of all outgoing edges of a vertex.

- **Parameters**
  - `vertex` – Vertex whose outgoing edges will be returned.
- **Returns** – The edges going out of the supplied vertex.

- **serialize**

```
public SerializedGraph serialize(java.util.List attributes)
```

7.3.14 Class DefaultVertex

This is an DefaultVertex, which has basic functions and is provided by the main application. This vertex can be derived by plugins which offer more functionality than the basic vertex.

7.3.14.1 Declaration

```
public class DefaultVertex
 extends java.lang.Object implements Vertex
```

7.3.14.2 Constructors

- **DefaultVertex**

```
public DefaultVertex()
```

7.3.14.3 Methods

- **addToFastGraphAccessor**

```
void addToFastGraphAccessor(FastGraphAccessor fga)
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Adds the vertex to a `FastGraphAccessor` (in 7.3.20, page 85).
  - **Parameters**
    - `fga` – The `FastGraphAccessor` (in 7.3.20, page 85) to whom this vertex will be added.

- **getID**

```
java.lang.Integer getID()
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the ID of the vertex. Every vertex in one graph has a unique ID.

- **Returns** – The ID of the vertex.

- **getLabel**

```
java.lang.String getLabel()
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the label of the vertex, that will be shown in the GUI. The label can be an empty string.
  - **Returns** – The label of the vertex

- **getName**

```
java.lang.String getName()
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the name of the vertex. A descriptive name of the vertex. Multiple vertices with equal name in one graph are allowed. Therefore don't use this as identifier, instead use `getID()`.
  - **Returns** – The name of the vertex.

- **getX**

```
int getX()
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the X-coordinate of the vertex.
  - **Returns** – The X-coordinate of this vertex.

- **getY**

```
int getY()
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the Y-coordinate of the vertex.
  - **Returns** – The Y-coordinate of the vertex.

- **serialize**

```
SerializedVertex serialize(java.util.List attributes)
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns a `SerializedVertex` (in 7.3.26, page 94) representation of the graph.

- **Parameters**
  - `attributes` – The attributes that have to be serialized.
- **Returns** – The `SerializedVertex` (in 7.3.26, page 94) representation of the graph.

7.3.15 Class DirectedEdge

A `DirectedEdge` (in 7.3.15, page 80) is an edge that has one source and one target vertex. The direction of the edge is specified.

7.3.15.1 Declaration

```
public class DirectedEdge
 extends java.lang.Object implements Edge
```

7.3.15.2 Constructors

- **DirectedEdge**

  ```
  public DirectedEdge()
  ```

7.3.15.3 Methods

- **addToFastGraphAccessor**

  ```
  void addToFastGraphAccessor(FastGraphAccessor fga)
  ```

  - **Description copied from Edge** (**in 7.3.3, page 61**)
    Adds the edge to a `FastGraphAccessor` (in 7.3.20, page 85).
  - **Parameters**
    - `fga` – The `FastGraphAccessor` (in 7.3.20, page 85) to whom this edge will be added.

- **getID**

  ```
  java.lang.Integer getID()
  ```

  - **Description copied from Edge** (**in 7.3.3, page 61**)
    Returns the ID of the edge.
  - **Returns** – The id of the edge.

- **getLabel**

  ```
  java.lang.String getLabel()
  ```

- **Description copied from Edge** (**in 7.3.3, page 61**)

  Returns the label of the edge.
- **Returns** – The label of the edge.

- **getName**

```
java.lang.String getName()
```

- **Description copied from Edge** (**in 7.3.3, page 61**)

  Returns the name of the edge.
- **Returns** – The name of the edge.

- **getPath**

```
EdgePath getPath()
```

- **Description copied from Edge** (**in 7.3.3, page 61**)

  Returns the `EdgePath` (in 7.3.18, page 83) of the edge. The edge path is attached to the edge and cannot be replaced.
- **Returns** – the edge path

- **getSource**

```
public Vertex getSource()
```

- **Description**

  Returns the source vertex of this directed edge.
- **Returns** – The vertex the edge is coming from.

- **getTarget**

```
public Vertex getTarget()
```

- **Description**

  Returns the target vertex of this edge.
- **Returns** – The vertex the edge is pointing at/going to.

- **getVertices**

```
java.util.List getVertices()
```

- **Description copied from Edge** (**in 7.3.3, page 61**)
  Returns the vertices connected with this edge.
- **Returns** – The vertices connected with the edge.

- **serialize**

```
SerializedEdge serialize(java.util.List attributes)
```

  - **Description copied from Edge** (**in 7.3.3, page 61**)
    Returns a `SerializedEdge` (in 7.3.24, page 90) representation of the edge.
  - **Parameters**
    * `attributes` – The attributes that have to be serialized.
  - **Returns** – The `SerializedEdge` (in 7.3.24, page 90) representation of the edge.

7.3.16 Class DirectedGraphLayoutOption

A `LayoutOption` (in 7.1.10, page 38) which is specific for `DirectedGraph` (in 7.3.2, page 59).

7.3.16.1 Declaration

```
public abstract class DirectedGraphLayoutOption
 extends plugin.LayoutOption
```

7.3.16.2 Constructors

- **DirectedGraphLayoutOption**

```
public DirectedGraphLayoutOption()
```

7.3.16.3 Methods

- **setGraph**

```
public void setGraph(DirectedGraph graph)
```

  - **Description**
    Sets the graph that will be the target of the DirectedGraphLayoutOption.
  - **Parameters**
    * `graph` – The graph that will be the target of this DirectedGraphLayoutOption.

7.3.17 Class DirectedGraphLayoutRegister

A `LayoutRegister` (in 7.1.5, page 32) which is specific for `DirectedGraphLayoutOption` (in 7.3.16, page 82).

7.3.17.1 Declaration

```
public class DirectedGraphLayoutRegister
 extends java.lang.Object implements plugin.LayoutRegister
```

7.3.17.2 Constructors

- **DirectedGraphLayoutRegister**

  ```
  public DirectedGraphLayoutRegister()
  ```

7.3.17.3 Methods

- **addLayoutOption**

  ```
  public void addLayoutOption(DirectedGraphLayoutOption option)
  ```

- **getLayoutOptions**

  ```
  java.util.List getLayoutOptions()
  ```

  - **Description copied from plugin.LayoutRegister** (**in 7.1.5, page 32**)
    Returns all available layouts for the graph associated with this register.
  - **Returns** – the available layouts

7.3.18 Class EdgePath

An abstract super class for edge paths. Contains basic information every edge path must provide.

7.3.18.1 Declaration

```
public abstract class EdgePath
 extends java.lang.Object
```

7.3.18.2 All known subclasses

OrthogonalEdgePath (in 7.3.23, page 89)

7.3.18.3 Constructors

- **EdgePath**

```
public EdgePath()
```

7.3.18.4 Methods

- **getNodes**

```
public abstract java.util.List getNodes()
```

  - **Description**

    Returns all nodes the edge has to pass through. In the order it has to pass through them.

  - **Returns** – the list of nodes

- **getSegmentsCount**

```
public abstract int getSegmentsCount()
```

  - **Description**

    Returns out of how many segments the path consists.

  - **Returns** – the number of segments

7.3.19 Class EdgePath.Point

This class is a standard immutable 2D Vector with integer values as it's components.

7.3.19.1 Declaration

```
public class EdgePath.Point
 extends java.lang.Object
```

7.3.19.2 Fields

- `public final int x`
- `public final int y`

7.3.19.3 Constructors

- **Point**

```
public Point(int x,int y)
```

7.3.20 Class FastGraphAccessor

This class provides a fast lookup of `Vertex` (in 7.3.10, page 71) and `Edge` (in 7.3.3, page 61) for a given Attribute value pair without traversing a `Graph` (in 7.3.4, page 62). FastGraphAccessor is a helper class for looking up all `Vertex` (in 7.3.10, page 71) and `Edge` (in 7.3.3, page 61) that have a specific value for a specific attribute. To achieve this all elements of a Graph need to add their attributes and values to a FastGraphAccessor. These values are not linked to the origin values so the fastGraphAccessor needs to be updated after changes when needed for following steps. This should be done by reverting and adding the values again.

7.3.20.1 Declaration

```
public class FastGraphAccessor
 extends java.lang.Object
```

7.3.20.2 Constructors

- **FastGraphAccessor**

  ```
  public FastGraphAccessor()
  ```

7.3.20.3 Methods

- **addEdgeForAttribute**

  ```
  public void addEdgeForAttribute(java.lang.String name,Edge edge,
    int value)
  ```

  - **Description**

    Adds an `Edge` (in 7.3.3, page 61) for a given attribute with a given value.

  - **Parameters**

    * `name` – name of the attribute
    * `value` – value of the attribute
    * `edge` – edge that has this value for the given attribute

- **addEdgeForAttribute**

  ```
  public void addEdgeForAttribute(java.lang.String name,java.lang.
    String value,Edge edge)
  ```

  - **Description**

    Adds an `Edge` (in 7.3.3, page 61) for a given attribute with a given value.

  - **Parameters**

  - `name` – name of the attribute
  - `value` – value of the attribute
  - `edge` – edge that has this value for the given attribute

- **addVertexForAttribute**

```
public void addVertexForAttribute(Vertex vertex,java.lang.String
    value,int name)
```

  - **Description**

    adds an `Vertex` (in 7.3.10, page 71) for a given attribute with a given value

  - **Parameters**
    - `name` – name of the attribute
    - `value` – value of the attribute
    - `vertex` – vertex that has this value for the given attribute

- **addVertexForAttribute**

```
public void addVertexForAttribute(Vertex vertex,java.lang.String
    value,java.lang.String name)
```

  - **Description**

    adds an `Vertex` (in 7.3.10, page 71) for a given attribute with a given value

  - **Parameters**
    - `name` – name of the attribute
    - `value` – value of the attribute
    - `vertex` – vertex that has this value for the given attribute

- **getEdgesByAttribute**

```
public java.util.List getEdgesByAttribute(java.lang.String name,
   int value)
```

  - **Description**

    gets a `List` of `Edge` (in 7.3.3, page 61) that contains all `Edge` (in 7.3.3, page 61) that have the value for given attribute

  - **Parameters**
    - `name` – name of the attribute
    - `value` – value of the attribute
  - **Returns** – a `List` of `Edge` (in 7.3.3, page 61) that has the value for given attribute

- **getEdgesByAttribute**

  ```
  public java.util.List getEdgesByAttribute(java.lang.String name,
      java.lang.String value)
  ```

  - **Description**

    gets a List of `Edge` (in 7.3.3, page 61) that contains all `Edge` (in 7.3.3, page 61) that have the value for given attribute

  - **Parameters**
    - `name` – name of the attribute
    - `value` – value of the attribute
  - **Returns** – a `List` of `Edge` (in 7.3.3, page 61) that has tthe value for given attribute

- **getVerticesByAttribute**

  ```
  public java.util.List getVerticesByAttribute(java.lang.String
      name,int value)
  ```

  - **Description**

    gets a `List` of `Vertex` (in 7.3.10, page 71) that contains all `Vertex` (in 7.3.10, page 71) that have the value for given attribut

  - **Parameters**
    - `name` – name of the attribute
    - `value` – value of the attribute
  - **Returns** – a `List` of `Vertex` (in 7.3.10, page 71) that has the value for given attribute

- **getVerticesByAttribute**

  ```
  public java.util.List getVerticesByAttribute(java.lang.String
      name,java.lang.String value)
  ```

  - **Description**

    gets a `List` of `Vertex` (in 7.3.10, page 71) that contains all `Vertex` (in 7.3.10, page 71) that have the value for given attribut

  - **Parameters**
    - `name` – name of the attribute
    - `value` – value of the attribute
  - **Returns** – a `List` of `Vertex` (in 7.3.10, page 71) that has the value for given attribute

- **reset**

```
public void reset()
```

- **Description**

  Deletes all data in this FastGraphAccessor. After this step all the information needs to be readded to this. This is necessary when updating the FastGraphAccessor

7.3.21 Class GraphModel

A GraphModel contains one or more graphs. It is used to save nested or hierarchical graphs in one class.

7.3.21.1 Declaration

```
public abstract class GraphModel
 extends java.lang.Object
```

7.3.21.2 Constructors

- **GraphModel**

  ```
  public GraphModel()
  ```

7.3.21.3 Methods

- **getGraphs**

  ```
  public abstract java.util.List getGraphs()
  ```

  - **Description**

    Returns all `Graph` (in 7.3.4, page 62) contained in the GraphModel.
  - **Returns** – A list of all the `Graph` (in 7.3.4, page 62) contained in the GraphModel.

7.3.22 Class Group

This class allows to collect an amount of vertices.

7.3.22.1 Declaration

```
public class Group
 extends java.lang.Object
```

7.3.22.2 Constructors

- **Group**

  ```
  public Group()
  ```

7.3.22.3 Methods

- **addVertex**

```
public void addVertex(Vertex vertex)
```

  - **Description**

    Adds the vertex to this Group.

  - **Parameters**

    * `vertex` – vertex to add to this group

- **removeVertex**

```
public void removeVertex(Vertex vertex)
```

  - **Description**

    Removes the vertex argument from this Group.

  - **Parameters**

    * `vertex` – vertex to remove from this Group

7.3.23 Class OrthogonalEdgePath

An orthogonal edge path used as standard graphical edge representation.

7.3.23.1 Declaration

```
public class OrthogonalEdgePath
 extends graphmodel.EdgePath
```

7.3.23.2 Constructors

- **OrthogonalEdgePath**

```
public OrthogonalEdgePath()
```

7.3.23.3 Methods

- **getNodes**

```
public abstract java.util.List getNodes()
```

  - **Description copied from EdgePath** (**in 7.3.18, page 83**)

    Returns all nodes the edge has to pass through. In the order it has to pass through them.

- **Returns** – the list of nodes

- **getSegmentsCount**

```
public abstract int getSegmentsCount()
```

  - **Description copied from EdgePath** (**in 7.3.18, page 83**)
    Returns out of how many segments the path consists.
  - **Returns** – the number of segments

7.3.24 Class SerializedEdge

A serialized version of a `Edge` (in 7.3.3, page 61). It contains all attributes as a `List` of String to String entries which can be used by an Exporter to export a `Edge` (in 7.3.3, page 61). It is designed as an intermediate Step in the export workflow and should not be used for other purposes. Attributes in the `List` are not synchronized with attributes outside the `List` , and Attributes of SerializedEdge are not synchronized with the origin `Edge` (in 7.3.3, page 61) attributes.

7.3.24.1 Declaration

```
public class SerializedEdge
 extends java.lang.Object implements Edge
```

7.3.24.2 Constructors

- **SerializedEdge**

```
public SerializedEdge()
```

7.3.24.3 Methods

- **addToFastGraphAccessor**

```
void addToFastGraphAccessor(FastGraphAccessor fga)
```

  - **Description copied from Edge** (**in 7.3.3, page 61**)
    Adds the edge to a `FastGraphAccessor` (in 7.3.20, page 85).
  - **Parameters**
    - `fga` – The `FastGraphAccessor` (in 7.3.20, page 85) to whom this edge will be added.

- **getAttributes**

```
public java.util.Map getAttributes()
```

- **Description**

  Gets all serialized Attributes as a Map from String to String. This Map gets created when serializing a `Edge` (in 7.3.3, page 61) and is returned on demand. This should only be used for exporting Edges since the attributes are not synchronized with the attributes of the unserialized `Edge` (in 7.3.3, page 61)

- **Returns** – The Map of serialized Attributes

- **See also**

  * `java.util.Map`

- **getID**

```
java.lang.Integer getID()
```

  - **Description copied from Edge** (**in 7.3.3, page 61**)

    Returns the ID of the edge.

  - **Returns** – The id of the edge.

- **getLabel**

```
java.lang.String getLabel()
```

  - **Description copied from Edge** (**in 7.3.3, page 61**)

    Returns the label of the edge.

  - **Returns** – The label of the edge.

- **getName**

```
java.lang.String getName()
```

  - **Description copied from Edge** (**in 7.3.3, page 61**)

    Returns the name of the edge.

  - **Returns** – The name of the edge.

- **getPath**

```
EdgePath getPath()
```

  - **Description copied from Edge** (**in 7.3.3, page 61**)

    Returns the `EdgePath` (in 7.3.18, page 83) of the edge. The edge path is attached to the edge and cannot be replaced.

  - **Returns** – the edge path

- **getVertices**

```
java.util.List getVertices()
```

  - **Description copied from Edge** (**in 7.3.3, page 61**)
    Returns the vertices connected with this edge.
  - **Returns** – The vertices connected with the edge.

- **serialize**

```
SerializedEdge serialize(java.util.List attributes)
```

  - **Description copied from Edge** (**in 7.3.3, page 61**)
    Returns a `SerializedEdge` (in 7.3.24, page 90) representation of the edge.
  - **Parameters**
    - `attributes` – The attributes that have to be serialized.
  - **Returns** – The `SerializedEdge` (in 7.3.24, page 90) representation of the edge.

7.3.25 Class SerializedGraph

A serialized version of a `Graph` (in 7.3.4, page 62). It contains all attributes as a `List` of String to String entries which can be used by an `Exporter` (in 7.1.2, page 29) to export a `Graph` (in 7.3.4, page 62). It is designed as an intermediate Step in the export workflow and should not be used for other purposes. Attributes in the `List` are not synchronized with attributes outside the `List`, and Attributes of SerializedGraph are not synchronized with the origin `Graph` (in 7.3.4, page 62) attributes.

7.3.25.1 Declaration

```
public class SerializedGraph
 extends java.lang.Object implements Graph
```

7.3.25.2 Constructors

- **SerializedGraph**

```
public SerializedGraph()
```

7.3.25.3 Methods

- **addToFastGraphAccessor**

```
void addToFastGraphAccessor(FastGraphAccessor fga)
```

- **Description copied from Graph** (**in 7.3.4, page 62**)

  Adds the graph to a `FastGraphAccessor` (in 7.3.20, page 85).

- **Parameters**
  - `fga` – the `FastGraphAccessor` (in 7.3.20, page 85) to whom this graph will be added.

- **edgesOf**

```
java.util.Set edgesOf(Vertex vertex)
```

  - **Description copied from Graph** (**in 7.3.4, page 62**)

    Returns a list of all edges of a vertex.

  - **Parameters**
    - `vertex` – the vertex which edges will be returned.
  - **Returns** – All edges which are connected with the supplied vertex.

- **getAttributes**

```
public java.util.Map getAttributes()
```

  - **Description**

    Gets all serialized Attributes as a Map from String to String. This Map gets created when serializing a `Graph` (in 7.3.4, page 62) and is returned on demand. This should only be used for exporting Graphs since the attributes are not synchronized with the attributes of the unserialized `Graph` (in 7.3.4, page 62)

  - **Returns** – The Map of serialized Attributes
  - **See also**
    - `java.util.Map`

- **getEdgeSet**

```
java.util.Set getEdgeSet()
```

  - **Description copied from Graph** (**in 7.3.4, page 62**)

    Returns all edges of the graph.

  - **Returns** – A set of all edges of the graph.

- **getFastGraphAccessor**

```
FastGraphAccessor getFastGraphAccessor()
```

- **Description copied from Graph** (**in 7.3.4, page 62**)
  Returns the FastGraphAccessor of this Graph.
- **Returns** – the FastGraphAccessor of this Graph

- **getID**

```
java.lang.Integer getID()
```

  - **Description copied from Graph** (**in 7.3.4, page 62**)
    Returns the ID of the graph.
  - **Returns** – The id of the graph.

- **getName**

```
java.lang.String getName()
```

  - **Description copied from Graph** (**in 7.3.4, page 62**)
    Returns the name of the Graph.
  - **Returns** – The name of the graph.

- **getVertexSet**

```
java.util.Set getVertexSet()
```

  - **Description copied from Graph** (**in 7.3.4, page 62**)
    Returns all vertices of the graph.
  - **Returns** – A set of all vertices of the graph.

7.3.26 Class SerializedVertex

A serialized version of a `Vertex` (in 7.3.10, page 71). It contains all attributes as a `List` of String to String entries which can be used by an `Exporter` (in 7.1.2, page 29) to export a `Vertex` (in 7.3.10, page 71). It is designed as an intermediate Step in the export workflow and should not be used for other purposes. Attributes in the `List` are not synchronized with attributes outside the `List` , and Attributes of SerializedVertex are not synchronized with the origin `Vertex` (in 7.3.10, page 71) attributes.

7.3.26.1 Declaration

```
public class SerializedVertex
 extends java.lang.Object implements Vertex
```

7.3.26.2 Constructors

- **SerializedVertex**

  ```
  public SerializedVertex(java.util.List attributes)
  ```

  - **Description**

    creates

  - **Parameters**

    * `attributes` –

7.3.26.3 Methods

- **addToFastGraphAccessor**

  ```
  void addToFastGraphAccessor(FastGraphAccessor fga)
  ```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Adds the vertex to a `FastGraphAccessor` (in 7.3.20, page 85).

  - **Parameters**

    * `fga` – The `FastGraphAccessor` (in 7.3.20, page 85) to whom this vertex will be added.

- **getAttributes**

  ```
  public java.util.Map getAttributes()
  ```

  - **Description**

    Gets all serialized Attributes as a Map from String to String. This Map gets created when serializing a `Vertex` (in 7.3.10, page 71) and is returned on demand. This should only be used for exporting Vertices since the attributes are not synchronized with the attributes of the unserialized `Vertex` (in 7.3.10, page 71)

  - **Returns** – The Map of serialized Attributes

  - **See also**

    * `java.util.Map`

- **getID**

  ```
  java.lang.Integer getID()
  ```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the ID of the vertex. Every vertex in one graph has a unique ID.

- **Returns** – The ID of the vertex.

- **getLabel**

```
java.lang.String getLabel()
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the label of the vertex, that will be shown in the GUI. The label can be an empty string.
  - **Returns** – The label of the vertex

- **getName**

```
java.lang.String getName()
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the name of the vertex. A descriptive name of the vertex. Multiple vertices with equal name in one graph are allowed. Therefore don't use this as identifier, instead use `getID()`.
  - **Returns** – The name of the vertex.

- **getX**

```
int getX()
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the X-coordinate of the vertex.
  - **Returns** – The X-coordinate of this vertex.

- **getY**

```
int getY()
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns the Y-coordinate of the vertex.
  - **Returns** – The Y-coordinate of the vertex.

- **serialize**

```
SerializedVertex serialize(java.util.List attributes)
```

  - **Description copied from Vertex** (**in 7.3.10, page 71**)

    Returns a `SerializedVertex` (in 7.3.26, page 94) representation of the graph.

- **Parameters**
  - `attributes` – The attributes that have to be serialized.
- **Returns** – The `SerializedVertex` (in 7.3.26, page 94) representation of the graph.

7.4 Package joana

Package Contents *Page*

Classes

7.4.1 Class CallGraph

This is a specified graph representation for the Callgraph in Joana.

7.4.1.1 Declaration

```
public class CallGraph
 extends joana.JoanaGraph
```

7.4.1.2 Constructors

- **CallGraph**

```
public CallGraph()
```

7.4.1.3 Methods

- **collapse**

```
public graphmodel.CompoundVertex collapse(java.util.Set subset)
```

- **expand**

```
public java.util.Set expand(graphmodel.CompoundVertex vertex)
```

- **getLayerWidth**

```
public int getLayerWidth(int layerN)
```

- **getSubgraphs**

```
public java.util.List getSubgraphs()
```

- **isCompound**

```
public boolean isCompound(graphmodel.Vertex vertex)
```

7.4.2 Class CallGraphBuilder

The CallGraphBuilder implements an `IGraphBuilder` (in 7.3.6, page 66) and builds one `CallGraph` (in 7.4.1, page 98).

7.4.2.1 Declaration

```
public class CallGraphBuilder
 extends java.lang.Object implements graphmodel.IGraphBuilder
```

7.4.2.2 Constructors

- **CallGraphBuilder**

```
public CallGraphBuilder()
```

7.4.2.3 Methods

- **build**

```
graphmodel.Graph build()
```

  - **Description copied from graphmodel.IGraphBuilder** (**in 7.3.6, page 66**)
    Builds a graph from the given settings and returns it.
  - **Returns** – The graph that is being build by the IGraphBuilder.

- **getEdgeBuilder**

```
graphmodel.IEdgeBuilder getEdgeBuilder()
```

- **Description copied from graphmodel.IGraphBuilder** (**in 7.3.6, page 66**)
  Returns the EdgeBuilder which is specified for this graph.
- **Returns** – The `IEdgeBuilder` (in 7.3.5, page 64) which is specified for this graph.

- **getVertexBuilder**

```
graphmodel.IVertexBuilder getVertexBuilder(java.lang.String
    vertexID)
```

- **Description copied from graphmodel.IGraphBuilder** (**in 7.3.6, page 66**)
  Returns the VertexBuilder which is specified for this graph.
- **Parameters**
  - `vertexID` – The id of the vertex which associated IVertexBuilder will be returned.
- **Returns** – The `IVertexBuilder` (in 7.3.8, page 67) which is specified for this graph.

7.4.3 Class CallGraphLayout

Offers a layout for `CallGraph` (in 7.4.1, page 98). Groups vertices representing the same JavaMethod together.

7.4.3.1 Declaration

```
public class CallGraphLayout
 extends java.lang.Object implements sugiyama.LayeredLayoutAlgorithm
```

7.4.3.2 Constructors

- **CallGraphLayout**

```
public CallGraphLayout()
```

7.4.3.3 Methods

- **getSettings**

```
public parameter.Settings getSettings()
```

- **layout**

```
public void layout(CallGraph graph)
```

- **layoutLayeredGraph**

```
void layoutLayeredGraph(graphmodel.LayeredGraph graph)
```

- **Description copied from sugiyama.LayeredLayoutAlgorithm** (**in 7.6.11, page 149**)

  Applies its layout to a graph as in `layout(G graph)` but keeps the notion of layers. The algorithm will assign every vertex a coordinate and every edge a path. Additionally every vertex will be assigned a position in a layer in the LayeredGraph. A possible application is drawing of recursive graphs.

- **Parameters**
  - `graph` – the graph to apply the layout to

7.4.4 Class CallGraphLayoutOption

A `LayoutOption` (in 7.1.10, page 38) which is specific for `CallGraph` (in 7.4.1, page 98).

7.4.4.1 Declaration

```
public abstract class CallGraphLayoutOption
 extends plugin.LayoutOption
```

7.4.4.2 Constructors

- **CallGraphLayoutOption**

```
public CallGraphLayoutOption()
```

7.4.4.3 Methods

- **applyLayout**

```
public abstract void applyLayout()
```

  - **Description copied from plugin.LayoutOption** (**in 7.1.10, page 38**)

    This should execute the layout on the graph, which should be specified on construction, or in beforehand. The settings, which are accessible over `getSettings()` will be used to instantiate the LayoutAlgorithm.

- **chooseLayout**

```
public abstract void chooseLayout()
```

  - **Description copied from plugin.LayoutOption** (**in 7.1.10, page 38**)

    Called when this layout option is chosen. This allows the layout option to prepare the actual LayoutAlgorithm.

- **setGraph**

```
public void setGraph(CallGraph graph)
```

  - **Description**

    Sets the `CallGraph` (in 7.4.1, page 98) that will be the target of the CallGraphLayoutOption.

  - **Parameters**

    * `graph` – The `CallGraph` (in 7.4.1, page 98) that will be the target of this CallGraphLayoutOption.

- **setLayout**

```
public void setLayout(plugin.LayoutAlgorithm layout)
```

  - **Description**

    Sets the LayoutAlgorithm that will be used to layout the set graph.

  - **Parameters**

    * `layout` – The LayoutAlgorithm that will be used to layout the set graph.

7.4.5 Class FieldAccess

This specifies the vertex representation of FieldAccesses in a MethodGraph It contains a `FieldAccessGraph`.

7.4.5.1 Declaration

```
public class FieldAccess
 extends joana.JoanaVertex implements graphmodel.CompoundVertex
```

7.4.5.2 Constructors

- **FieldAccess**

```
public FieldAccess(FieldAccessGraph graph)
```

  - **Description**

    Constructor.

  - **Parameters**

    * `graph` – The FieldAccessGraph that will be set in the FieldAccess.

7.4.5.3 Methods

- **addToFastGraphAccessor**

  ```
  void addToFastGraphAccessor(graphmodel.FastGraphAccessor fga)
  ```

  - **Description copied from graphmodel.Vertex** (**in 7.3.10, page 71**)

    Adds the vertex to a `FastGraphAccessor` (in 7.3.20, page 85).
  - **Parameters**
    - `fga` – The `FastGraphAccessor` (in 7.3.20, page 85) to whom this vertex will be added.

- **getConnectedVertex**

  ```
  graphmodel.Vertex getConnectedVertex(graphmodel.Edge edge)
  ```

  - **Description copied from graphmodel.CompoundVertex** (**in 7.3.1, page 59**)

    Returns the connected vertex contained in the compound vertex for a given edge, where one end point of the edge has to be this vertex.
  - **Parameters**
    - `edge` – the edge to get the vertex to
  - **Returns** – the connected vertex if the edge is valid

- **getGraph**

  ```
  graphmodel.Graph getGraph()
  ```

  - **Description copied from graphmodel.CompoundVertex** (**in 7.3.1, page 59**)

    Returns the graph contained in the vertex.
  - **Returns** – the graph contained in the vertex.

7.4.6 Class FieldAccessGraph

A `JoanaGraph` (in 7.4.10, page 107) which specifies a `FieldAccess` (in 7.4.5, page 102) in a `JoanaGraph` (in 7.4.10, page 107)

7.4.6.1 Declaration

```
public class FieldAccessGraph
 extends joana.JoanaGraph
```

7.4.6.2 Constructors

- **FieldAccessGraph**

```
public FieldAccessGraph()
```

7.4.6.3 Methods

- **getLayerWidth**

```
public int getLayerWidth(int layerN)
```

- **getSubgraphs**

```
public java.util.List getSubgraphs()
```

7.4.7 Class FieldAccessLayout

The FieldAccessLayout applies its layout to a `FieldAccess` (in 7.4.5, page 102).

7.4.7.1 Declaration

```
public class FieldAccessLayout
 extends java.lang.Object implements sugiyama.LayeredLayoutAlgorithm
```

7.4.7.2 Constructors

- **FieldAccessLayout**

```
public FieldAccessLayout()
```

7.4.7.3 Methods

- **getSettings**

```
public parameter.Settings getSettings()
```

- **layout**

```
public void layout(FieldAccessGraph graph)
```

- **layoutLayeredGraph**

```
void layoutLayeredGraph(graphmodel.LayeredGraph graph)
```

- **Description copied from sugiyama.LayeredLayoutAlgorithm** (**in 7.6.11, page 149**)

  Applies its layout to a graph as in `layout(G graph)` but keeps the notion of layers. The algorithm will assign every vertex a coordinate and every edge a path. Additionally every vertex will be assigned a position in a layer in the LayeredGraph. A possible application is drawing of recursive graphs.

- **Parameters**
  - `graph` – the graph to apply the layout to

7.4.8 Class JoanaEdge

A Joana specific Edge. It contains parameters which are only used/usefull in `JoanaGraph` (in 7.4.10, page 107).

7.4.8.1 Declaration

```
public class JoanaEdge
 extends graphmodel.DirectedEdge
```

7.4.8.2 Constructors

- **JoanaEdge**

```
public JoanaEdge()
```

7.4.8.3 Methods

- **getEdgeKind**

```
public java.lang.String getEdgeKind()
```

  - **Description**

    Returns the edgeKind of the JoanaEdge.

  - **Returns** – The edgeKind of the JoanaEdge.

7.4.9 Class JoanaEdgeBuilder

The JoanaEdgeBuilder is a `IEdgeBuilder` (in 7.3.5, page 64), specifically for building `JoanaEdge` (in 7.4.8, page 105).

7.4.9.1 Declaration

```
public class JoanaEdgeBuilder
 extends java.lang.Object implements graphmodel.IEdgeBuilder
```

7.4.9.2 Constructors

- **JoanaEdgeBuilder**

  ```
  public JoanaEdgeBuilder()
  ```

7.4.9.3 Methods

- **addData**

  ```
  void addData(java.lang.String keyname,java.lang.String value)
  ```

  - **Description copied from graphmodel.IEdgeBuilder** (**in 7.3.5, page 64**)
    Adds additional data to this edge. The specific EdgeBuilder implementation needs to decide how to save the value for given edge type.
  - **Parameters**
    - `keyname` – Name of the attribute
    - `value` – Value of the attribute

- **build**

  ```
  graphmodel.Edge build()
  ```

  - **Description copied from graphmodel.IEdgeBuilder** (**in 7.3.5, page 64**)
    . Builds an Edge with the given Data and returns it.
  - **Returns** – The Edge that is being build by the IEdgeBuilder

- **newEdge**

  ```
  void newEdge(java.lang.String source,java.lang.String target)
  ```

  - **Description copied from graphmodel.IEdgeBuilder** (**in 7.3.5, page 64**)
    Sets source and target vertices of the edge build by this.
  - **Parameters**
    - `source` – String representation of the source vertex as ID
    - `target` – String representation of the target vertex as ID

- **setDirection**

```
void setDirection(java.lang.String direction)
```

  - **Description copied from graphmodel.IEdgeBuilder** (**in 7.3.5, page 64**)
    Sets the direction of the edge build by this.
  - **Parameters**
    * `direction` – String representation of the direction. Can be one of

- **setID**

```
void setID(java.lang.String id)
```

  - **Description copied from graphmodel.IEdgeBuilder** (**in 7.3.5, page 64**)
    Sets the ID of the edge build by this.
  - **Parameters**
    * `id` – value to which the id is set

7.4.10 Class JoanaGraph

An abstract superclass for all JOANA specific graphs.

7.4.10.1 Declaration

```
public abstract class JoanaGraph
 extends graphmodel.DefaultDirectedGraph implements graphmodel.
    LayeredGraph
```

7.4.10.2 All known subclasses

MethodGraph (in 7.4.19, page 120), FieldAccessGraph (in 7.4.6, page 103), CallGraph (in 7.4.1, page 98)

7.4.10.3 Constructors

- **JoanaGraph**

```
public JoanaGraph()
```

7.4.10.4 Methods

- **getHeight**

  ```
  int getHeight()
  ```

  - **Description copied from graphmodel.LayeredGraph** (**in 7.3.9, page 68**) Returns the height, i.e. the number of layers.
  - **Returns** – the height

- **getLayer**

  ```
  java.util.List getLayer(int layerNum)
  ```

  - **Description copied from graphmodel.LayeredGraph** (**in 7.3.9, page 68**) Get all vertices from a certain layer.
  - **Parameters**
    - `layerNum` – the index of the layer
  - **Returns** – a list of all vertices which are on this layer

- **getLayer**

  ```
  public int getLayer(JoanaVertex vertex)
  ```

- **getLayerCount**

  ```
  int getLayerCount()
  ```

  - **Description copied from graphmodel.LayeredGraph** (**in 7.3.9, page 68**) Get the amount of layers.
  - **Returns** – the amount of layers that contain at least one vertex

- **getLayers**

  ```
  java.util.List getLayers()
  ```

  - **Description copied from graphmodel.LayeredGraph** (**in 7.3.9, page 68**) Get all layers that contain vertices.
  - **Returns** – a list of lists of vertices which are on this layer

- **getMaxWidth**

```
int getMaxWidth()
```

- **Description copied from graphmodel.LayeredGraph** (**in 7.3.9, page 68**)
  Returns the width of the widest layer, i.e. the number of vertices the layer with the most vertices contains.
- **Returns** – the maximum width

- **getVertexCount**

```
int getVertexCount(int layerNum)
```

- **Description copied from graphmodel.LayeredGraph** (**in 7.3.9, page 68**)
  Get the number of vertices which are on a certain layer
- **Parameters**
  - `layerNum` – the layer number to get the vertex count from
- **Returns** – the number of vertices which are on this layer

7.4.11 Class JoanaGraphModel

A Joana specific `GraphModel` (in 7.3.21, page 88). It can only contain `MethodGraph` (in 7.4.19, page 120) and `CallGraph` (in 7.4.1, page 98).

7.4.11.1 Declaration

```
public class JoanaGraphModel
 extends graphmodel.GraphModel
```

7.4.11.2 Constructors

- **JoanaGraphModel**

```
public JoanaGraphModel()
```

7.4.11.3 Methods

- **getCallGraph**

```
public CallGraph getCallGraph()
```

- **Description**
  Returns all `CallGraph` (in 7.4.1, page 98) contained in the JoanaGraphModel.
- **Returns** – A list of all the `CallGraph` (in 7.4.1, page 98) contained in the JoanaGraphModel.

- **getGraphs**

  ```
  public abstract java.util.List getGraphs()
  ```

  - **Description copied from graphmodel.GraphModel** (**in 7.3.21, page 88**)

    Returns all `Graph` (in 7.3.4, page 62) contained in the GraphModel.
  - **Returns** – A list of all the `Graph` (in 7.3.4, page 62) contained in the GraphModel.

- **getMethodGraphs**

  ```
  public java.util.List getMethodGraphs()
  ```

  - **Description**

    Returns all `MethodGraph` (in 7.4.19, page 120) contained in the JoanaGraphModel.
  - **Returns** – A list of all the `MethodGraph` (in 7.4.19, page 120) contained in the JoanaGraphModel.

- **setCallGraph**

  ```
  public void setCallGraph(CallGraph callgraph)
  ```

  - **Description**

    Sets the `CallGraph` (in 7.4.1, page 98) in the JoanaGraphModel.
  - **Parameters**
    - `callgraph` – The `CallGraph` (in 7.4.1, page 98) that will be set in the JoanaGraphModel.

- **setMethodGraphs**

  ```
  public void setMethodGraphs(java.util.List methodgraphs)
  ```

  - **Description**

    Sets the `MethodGraph` (in 7.4.19, page 120) objects in the JoanaGraphModel.
  - **Parameters**
    - `methodgraphs` – The `MethodGraph` (in 7.4.19, page 120) objects that will be set in the JoanaGraphModel.

7.4.12 Class JoanaGraphModelBuilder

The JoanaGraphModelBuilder implements the `IGraphModelBuilder` (in 7.3.7, page 66) and creates a `JoanaGraphModel` (in 7.4.11, page 109).

7.4.12.1 Declaration

```
public class JoanaGraphModelBuilder
 extends java.lang.Object implements graphmodel.IGraphModelBuilder
```

7.4.12.2 Constructors

- **JoanaGraphModelBuilder**

  ```
  public JoanaGraphModelBuilder()
  ```

7.4.12.3 Methods

- **build**

  ```
  graphmodel.GraphModel build()
  ```

  - **Description copied from graphmodel.IGraphModelBuilder** (**in 7.3.7, page 66**)

    Builds a graphmodel from the given settings and returns it.
  - **Returns** – The `GraphModel` (in 7.3.21, page 88) that is being build by the IGraphModelBuilder.

- **getGraphBuilder**

  ```
  graphmodel.IGraphBuilder getGraphBuilder(java.lang.String
      graphID)
  ```

  - **Description copied from graphmodel.IGraphModelBuilder** (**in 7.3.7, page 66**)

    Returns a specific `IGraphBuilder` (in 7.3.6, page 66) for a graph, which belongs to the `GraphModel` (in 7.3.21, page 88).
  - **Parameters**
    - `graphID` – The id of the graph which associated `IGraphBuilder` (in 7.3.6, page 66) will be returned.
  - **Returns** – The IGraphBuilder of the graph which is referenced over the graphID.

7.4.13 Class JoanaPlugin

A plugin for GAns that supports the creation and visualization of Joana system dependence graphs.

7.4.13.1 Declaration

```
public class JoanaPlugin
 extends java.lang.Object implements plugin.Plugin
```

7.4.13.2 Constructors

- **JoanaPlugin**

  ```
  public JoanaPlugin()
  ```

  - **Description**

    Constructor. The constructor is called by the ServiceLoader.

7.4.13.3 Methods

- **getCallGraphLayoutRegister**

  ```
  public static JoanaPlugin.CallGraphLayoutRegister
    getCallGraphLayoutRegister()
  ```

- **getEdgeFilter**

  ```
  java.util.List getEdgeFilter()
  ```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

    Returns all by the plugin provided `EdgeFilter` (in 7.1.8, page 35). If none are provided returns `null` or an empty list.
  - **Returns** – the list of provided edge filter

- **getExporter**

  ```
  java.util.List getExporter()
  ```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

    Returns all by the plugin provided `Exporter` (in 7.1.2, page 29). If none are provided returns `null` or an empty list.
  - **Returns** – a list of provided exporter

- **getImporter**

  ```
  java.util.List getImporter()
  ```

- **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

  Returns all by the plugin provided `Importer` (in 7.1.3, page 30). If none are provided returns `null` or an empty list.

- **Returns** – a list of provided importer

- **getMethodGraphLayoutRegister**

```
public static JoanaPlugin.MethodGraphLayoutRegister
    getMethodGraphLayoutRegister()
```

- **getName**

```
java.lang.String getName()
```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

    Returns the name of the plugin. Uniqueness can't be assumed.

  - **Returns** – the name of the plugin

- **getVertexFilter**

```
java.util.List getVertexFilter()
```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

    Returns all by the plugin provided `VertexFilter` (in 7.1.12, page 40). If none are provided returns `null` or an empty list.

  - **Returns** – the list of provided vertex filter

- **getWorkspaceOptions**

```
java.util.List getWorkspaceOptions()
```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

    Returns all provided by the plugin `WorkspaceOption` (in 7.1.13, page 41). If none are provided returns `null` or an empty list.

  - **Returns** – The list of provided workspace options

- **load**

```
void load()
```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

    Called after all plugins have been constructed. Ïnter-Plugincommunication, like registering of layouts for graphs in other plugins should be executed in here.

7.4.14 Class JoanaPlugin.CallGraphLayoutRegister

7.4.14.1 Declaration

```
public static class JoanaPlugin.CallGraphLayoutRegister
 extends java.lang.Object implements plugin.LayoutRegister
```

7.4.14.2 Constructors

- **CallGraphLayoutRegister**

  ```
  public CallGraphLayoutRegister()
  ```

7.4.14.3 Methods

- **addLayoutOption**

  ```
  public void addLayoutOption(CallGraphLayoutOption option)
  ```

- **getLayoutOptions**

  ```
  java.util.List getLayoutOptions()
  ```

  - **Description copied from plugin.LayoutRegister** (**in 7.1.5, page 32**)
    Returns all available layouts for the graph associated with this register.
  - **Returns** – the available layouts

7.4.15 Class JoanaPlugin.MethodGraphLayoutRegister

7.4.15.1 Declaration

```
public static class JoanaPlugin.MethodGraphLayoutRegister
 extends java.lang.Object implements plugin.LayoutRegister
```

7.4.15.2 Constructors

- **MethodGraphLayoutRegister**

  ```
  public MethodGraphLayoutRegister()
  ```

7.4.15.3 Methods

- **addLayoutOption**

```
public void addLayoutOption(MethodGraphLayoutOption option)
```

- **getLayoutOptions**

```
java.util.List getLayoutOptions()
```

  - **Description copied from plugin.LayoutRegister** (**in 7.1.5, page 32**)
    Returns all available layouts for the graph associated with this register.
  - **Returns** – the available layouts

7.4.16 Class JoanaVertex

A Joana specific Vertex. It contains parameters which are only used/useful for Joana.

7.4.16.1 Declaration

```
public class JoanaVertex
 extends graphmodel.DefaultVertex
```

7.4.16.2 All known subclasses

FieldAccess (in 7.4.5, page 102)

7.4.16.3 Constructors

- **JoanaVertex**

```
public JoanaVertex()
```

7.4.16.4 Methods

- **getNodeBCIndex**

```
public java.lang.Integer getNodeBCIndex()
```

  - **Description**
    Returns the nodeBCIndex of the JoanaVertex.
  - **Returns** – The nodeBCIndex of the JoanaVertex.

- **getNodeBcName**

  ```
  public java.lang.String getNodeBcName()
  ```

  - **Description**
    Returns the nodeBcName of the JoanaVertex.
  - **Returns** – The nodeBcName of the JoanaVertex.

- **getNodeEc**

  ```
  public java.lang.Integer getNodeEc()
  ```

  - **Description**
    Returns the nodeEc of the JoanaVertex.
  - **Returns** – The nodeEc of the JoanaVertex.

- **getNodeEr**

  ```
  public java.lang.Integer getNodeEr()
  ```

  - **Description**
    Returns the nodeEr of the JoanaVertex.
  - **Returns** – The nodeEr of the JoanaVertex.

- **getNodeKind**

  ```
  public java.lang.String getNodeKind()
  ```

  - **Description**
    Returns the nodeKind of the JoanaVertex.
  - **Returns** – The nodeKind of the JoanaVertex.

- **getNodeOperation**

  ```
  public java.lang.String getNodeOperation()
  ```

  - **Description**
    Returns the nodeOperation of the JoanaVertex.
  - **Returns** – The nodeOperation of the JoanaVertex.

- **getNodeProc**

```
public java.lang.Integer getNodeProc()
```

- **Description**
  Returns the nodeProc of the JoanaVertex.
- **Returns** – The nodeProc of the JoanaVertex.

- **getNodeSc**

```
public java.lang.Integer getNodeSc()
```

- **Description**
  Returns the nodeSc of the JoanaVertex.
- **Returns** – The nodeSc of the JoanaVertex.

- **getNodeSource**

```
public java.lang.String getNodeSource()
```

- **Description**
  Returns the nodeSource of the JoanaVertex.
- **Returns** – The nodeSource of the JoanaVertex.

- **getNodeSr**

```
public java.lang.Integer getNodeSr()
```

- **Description**
  Returns the nodeSr of the JoanaVertex.
- **Returns** – The nodeSr of the JoanaVertex.

7.4.17 Class JoanaVertexBuilder

The JoanaVertexBuilder implements an `IVertexBuilder` (in 7.3.8, page 67) and creates a `JoanaVertex` (in 7.4.16, page 115).

7.4.17.1 Declaration

```
public class JoanaVertexBuilder
 extends java.lang.Object implements graphmodel.IVertexBuilder
```

7.4.17.2 Constructors

- **JoanaVertexBuilder**

```
public JoanaVertexBuilder()
```

7.4.17.3 Methods

- **addData**

  ```
  void addData(java.lang.String keyname,java.lang.String value)
  ```

  - **Description copied from graphmodel.IVertexBuilder** (**in 7.3.8, page 67**)

    Add Data to this Vertex. The IVertexBuilder needs to parse the data and add it to the edge

  - **Parameters**
    - `keyname` – Name of the attribute which is added
    - `value` – Value of the attribute

- **build**

  ```
  graphmodel.Vertex build()
  ```

  - **Description copied from graphmodel.IVertexBuilder** (**in 7.3.8, page 67**)

    . Builds a Vertex with the given Data and returns it.

  - **Returns** – The Vertex that is being build by the IVertexBuilder

- **getGraphBuilder**

  ```
  graphmodel.IGraphBuilder getGraphBuilder(java.lang.String
      graphID)
  ```

  - **Description copied from graphmodel.IVertexBuilder** (**in 7.3.8, page 67**)

    This method returns an specific GraphBuilder. This method is used to implement nested Graphs.

  - **Parameters**
    - `graphID` – The id of the graph which associated `IGraphBuilder` (in 7.3.6, page 66) will be returned.
  - **Returns** – The IGraphBuilder of the graph which is referenced over the graphID.

- **setID**

  ```
  void setID(java.lang.String id)
  ```

  - **Description copied from graphmodel.IVertexBuilder** (**in 7.3.8, page 67**)

    Sets the ID of the vertex build by this.

  - **Parameters**
    - `id` – value to which the id is set

7.4.18 Class JoanaWorkspace

The `JoanaWorkspace` (in 7.4.18, page 119) is the workspace for Joana graphs. It is used to define parameters, provides an `IGraphModelBuilder` (in 7.3.7, page 66) and contains a `JoanaGraphModel` (in 7.4.11, page 109).

7.4.18.1 Declaration

```
public class JoanaWorkspace
 extends java.lang.Object implements plugin.Workspace
```

7.4.18.2 Constructors

- **JoanaWorkspace**

  ```
  public JoanaWorkspace()
  ```

7.4.18.3 Methods

- **getGraphModel**

  ```
  graphmodel.GraphModel getGraphModel()
  ```

  - **Description copied from plugin.Workspace** (**in 7.1.7, page 34**)
    Returns the `GraphModel` (in 7.3.21, page 88) stored in the workspace.
  - **Returns** – the graph model

- **getGraphModelBuilder**

  ```
  graphmodel.IGraphModelBuilder getGraphModelBuilder()
  ```

  - **Description copied from plugin.Workspace** (**in 7.1.7, page 34**)
    Returns a builder to build a graph model in this workspace.
  - **Returns** – the builder

- **getSettings**

  ```
  parameter.Settings getSettings()
  ```

  - **Description copied from plugin.Workspace** (**in 7.1.7, page 34**)
    Returns a set of parameters to initialize this workspace. When the settings have been adjusted, the client has to call `initialize()`. To initialize the workspace with the settings.

- **Returns** – the settings

- **initialize**

```
void initialize()
```

  - **Description copied from plugin.Workspace** (**in 7.1.7, page 34**)

    Initializes this workspace with the settings if they have not been adjusted. If the settings have not been adjusted, default values will be used.

7.4.19 Class MethodGraph

This is a specific graph representation for a MethodGraph in JOANA

7.4.19.1 Declaration

```
public class MethodGraph
 extends joana.JoanaGraph
```

7.4.19.2 Constructors

- **MethodGraph**

```
public MethodGraph()
```

7.4.19.3 Methods

- **collapse**

```
public graphmodel.CompoundVertex collapse(java.util.Set subset)
```

- **expand**

```
public java.util.Set expand(graphmodel.CompoundVertex vertex)
```

- **getEntryVertex**

```
public JoanaVertex getEntryVertex()
```

  - **Description**

    Returns the entry vertex of a method.

  - **Returns** – The entry vertex of a method.

- **getFieldAccesses**

  ```
  public java.util.List getFieldAccesses()
  ```

  - **Description**

    Returns a list of all `FieldAccess` (in 7.4.5, page 102) in the MethodGraph.

  - **Returns** – A list of all `FieldAccess` (in 7.4.5, page 102) in the MethodGraph.

- **getLayerWidth**

  ```
  public int getLayerWidth(int layerN)
  ```

- **getMethodCalls**

  ```
  public java.util.List getMethodCalls()
  ```

  - **Description**

    Returns a list of all `JoanaVertex` (in 7.4.16, page 115) which are method calls in the MethodGraph.

  - **Returns** – A list of all method calls.

- **getSubgraphs**

  ```
  public java.util.List getSubgraphs()
  ```

- **isCompound**

  ```
  public boolean isCompound(graphmodel.Vertex vertex)
  ```

- **setRegister**

  ```
  protected static void setRegister(plugin.LayoutRegister register
      )
  ```

  - **Description**

    Sets the `LayoutRegister` (in 7.1.5, page 32), which stores the available `LayoutOption` (in 7.1.10, page 38) for all method graphs statically.

  - **Parameters**

    * `register` – The `LayoutRegister` (in 7.1.5, page 32) that will be set.

7.4.20 Class MethodGraphBuilder

The MethodGraphBuilder is a `IGraphBuilder` (in 7.3.6, page 66), specifically for building `MethodGraph` (in 7.4.19, page 120).

7.4.20.1 Declaration

```
public class MethodGraphBuilder
 extends java.lang.Object implements graphmodel.IGraphBuilder
```

7.4.20.2 Constructors

- **MethodGraphBuilder**

  ```
  public MethodGraphBuilder()
  ```

7.4.20.3 Methods

- **build**

  ```
  graphmodel.Graph build()
  ```

  - **Description copied from graphmodel.IGraphBuilder** (**in 7.3.6, page 66**) Builds a graph from the given settings and returns it.
  - **Returns** – The graph that is being build by the IGraphBuilder.

- **getEdgeBuilder**

  ```
  graphmodel.IEdgeBuilder getEdgeBuilder()
  ```

  - **Description copied from graphmodel.IGraphBuilder** (**in 7.3.6, page 66**) Returns the EdgeBuilder which is specified for this graph.
  - **Returns** – The `IEdgeBuilder` (in 7.3.5, page 64) which is specified for this graph.

- **getVertexBuilder**

  ```
  graphmodel.IVertexBuilder getVertexBuilder(java.lang.String
      vertexID)
  ```

  - **Description copied from graphmodel.IGraphBuilder** (**in 7.3.6, page 66**) Returns the VertexBuilder which is specified for this graph.
  - **Parameters**
    - `vertexID` – The id of the vertex which associated IVertexBuilder will be returned.
  - **Returns** – The `IVertexBuilder` (in 7.3.8, page 67) which is specified for this graph.

7.4.21 Class MethodGraphLayout

Implements hierarchical layout with layers for `MethodGraph` (in 7.4.19, page 120). This graph contains field access subgraphs.

7.4.21.1 Declaration

```
public class MethodGraphLayout
 extends java.lang.Object implements sugiyama.LayeredLayoutAlgorithm
```

7.4.21.2 Constructors

- **MethodGraphLayout**

  ```
  public MethodGraphLayout()
  ```

7.4.21.3 Methods

- **getSettings**

  ```
  public parameter.Settings getSettings()
  ```

- **layout**

  ```
  public void layout(MethodGraph graph)
  ```

  - **Description**

    Layouts a single `MethodGraph` (in 7.4.19, page 120) with the configured settings.

  - **Parameters**

    * `graph` – The `MethodGraph` (in 7.4.19, page 120) to layout.

- **layoutLayeredGraph**

  ```
  void layoutLayeredGraph(graphmodel.LayeredGraph graph)
  ```

  - **Description copied from sugiyama.LayeredLayoutAlgorithm** (**in 7.6.11, page 149**)

    Applies its layout to a graph as in `layout(G graph)` but keeps the notion of layers. The algorithm will assign every vertex a coordinate and every edge a path. Additionally every vertex will be assigned a position in a layer in the LayeredGraph. A possible application is drawing of recursive graphs.

  - **Parameters**

    * `graph` – the graph to apply the layout to

7.4.22 Class MethodGraphLayoutOption

A `LayoutOption` (in 7.1.10, page 38) which is specific for `MethodGraph` (in 7.4.19, page 120).

7.4.22.1 Declaration

```
public abstract class MethodGraphLayoutOption
 extends plugin.LayoutOption
```

7.4.22.2 Constructors

- **MethodGraphLayoutOption**

  ```
  public MethodGraphLayoutOption()
  ```

7.4.22.3 Methods

- **applyLayout**

  ```
  public abstract void applyLayout()
  ```

  - **Description copied from plugin.LayoutOption** (**in 7.1.10, page 38**)

    This should execute the layout on the graph, which should be specified on construction, or in beforehand. The settings, which are accessible over `getSettings()` will be used to instantiate the LayoutAlgorithm.

- **chooseLayout**

  ```
  public abstract void chooseLayout()
  ```

  - **Description copied from plugin.LayoutOption** (**in 7.1.10, page 38**)

    Called when this layout option is chosen. This allows the layout option to prepare the actual LayoutAlgorithm.

- **setGraph**

  ```
  public void setGraph(MethodGraph graph)
  ```

  - **Description**

    Sets the `MethodGraph` (in 7.4.19, page 120) that will be the target of the CallGraphLayoutOption.

  - **Parameters**

    * `graph` – The `MethodGraph` (in 7.4.19, page 120) that will be the target of the MethodGraphLayoutOption.

- **setLayout**

```
public void setLayout(plugin.LayoutAlgorithm layout)
```

  - **Description**

    Sets the LayoutAlgorithm that will be used to layout the set graph.

  - **Parameters**

    * `layout` – The LayoutAlgorithm that will be used to layout the set graph.

7.5 Package parameter

Package Contents *Page*

Classes

7.5.1 Class BooleanParameter

BooleanParameter are parameters with an boolean value space.

7.5.1.1 Declaration

```
public class BooleanParameter
 extends parameter.Parameter
```

7.5.1.2 Constructors

- **BooleanParameter**

```
public BooleanParameter(java.lang.String name,boolean value)
```

- **Description**

  Constructs a new BooleanParameter, sets its name and its default value.

- **Parameters**
  - `name` – The name of the parameter.
  - `value` – The value of the parameter.

7.5.1.3 Methods

- **accept**

```
public abstract void accept(ParameterVisitor visitor)
```

- **Description copied from Parameter** (**in 7.5.5, page 132**)

  Let the visitor visit this parameter.

- **Parameters**
  - `visitor` – The visitor to visit

- **compareTo**

```
public int compareTo(BooleanParameter o)
```

7.5.2 Class DoubleParameter

DoubleParameters are parameters with an double value space.

7.5.2.1 Declaration

```
public class DoubleParameter
 extends parameter.Parameter
```

7.5.2.2 Constructors

- **DoubleParameter**

```
public DoubleParameter(java.lang.String name,java.lang.Double
    value,java.lang.Double min,java.lang.Double max)
```

- **Description**

  Constructs a new DoubleParameter, sets its name, its default value and boundaries.

- **Parameters**

  * `name` – The name of the parameter.
  * `value` – The value of the parameter.
  * `min` – The minimum boundary of the parameter.
  * `max` – The maximum boundary of the parameter.

7.5.2.3 Methods

- **accept**

```
public abstract void accept(ParameterVisitor visitor)
```

  - **Description copied from Parameter** (**in 7.5.5, page 132**)
    Let the visitor visit this parameter.
  - **Parameters**
    * `visitor` – The visitor to visit

- **compareTo**

```
public int compareTo(DoubleParameter o)
```

- **getMax**

```
public double getMax()
```

  - **Description**
    Returns the maximum boundary.
  - **Returns** – The maximum boundary.

- **getMin**

```
public double getMin()
```

  - **Description**
    Returns the minimum boundary.
  - **Returns** – The minimum boundary.

- **setMax**

```
public void setMax(double max)
```

- **Description**

  Sets the maximum boundary.

- **Parameters**

  * `min` – The maximum boundary.

- **setMin**

```
public void setMin(double min)
```

- **Description**

  Sets the minimum boundary.

- **Parameters**

  * `min` – The minimum boundary.

7.5.3 Class IntegerParameter

IntegerParameters are parameters with an integer value space.

7.5.3.1 Declaration

```
public class IntegerParameter
 extends parameter.Parameter
```

7.5.3.2 Constructors

- **IntegerParameter**

```
public IntegerParameter(java.lang.String name,int value,int min,
    int max)
```

- **Description**

  Constructs a new IntegerParameter, sets its name, its default value and boundaries.

- **Parameters**

  * `name` – The name of the parameter.
  * `value` – The value of the parameter.
  * `min` – The minimum boundary of the parameter.
  * `max` – The maximum boundary of the parameter.

7.5.3.3 Methods

- **accept**

```
public abstract void accept(ParameterVisitor visitor)
```

  - **Description copied from Parameter** (**in 7.5.5, page 132**)
    Let the visitor visit this parameter.
  - **Parameters**
    - `visitor` – The visitor to visit

- **compareTo**

```
public int compareTo(IntegerParameter iw)
```

- **getMax**

```
public int getMax()
```

  - **Description**
    Returns the maximum boundary.
  - **Returns** – The maximum boundary.

- **getMin**

```
public int getMin()
```

  - **Description**
    Returns the minimum boundary.
  - **Returns** – The minimum boundary.

- **setMax**

```
public void setMax(int max)
```

  - **Description**
    Sets the maximum boundary.
  - **Parameters**
    - `min` – The maximum boundary.

- **setMin**

```
public void setMin(int min)
```

- **Description**

  Sets the minimum boundary.

- **Parameters**

  * `min` – The minimum boundary.

7.5.4 Class MultipleChoiceParameter

MultipleChoiceParameter are parameters with an predefined String value space.

7.5.4.1 Declaration

```
public class MultipleChoiceParameter
 extends parameter.Parameter
```

7.5.4.2 Constructors

- **MultipleChoiceParameter**

  ```
  public MultipleChoiceParameter(java.lang.String name)
  ```

  - **Description**

    Constructs a new MultipleChoiceParameter and sets its name. The possible choices and index are set as null.

  - **Parameters**

    * `name` – The name of the parameter.

- **MultipleChoiceParameter**

  ```
  public MultipleChoiceParameter(java.lang.String name,java.util.
      List choices,int init)
  ```

  - **Description**

    Constructs a new MultipleChoiceParameter, sets its name, its possible choices and initialized index.

  - **Parameters**

    * `name` – The name of the parameter.
    * `choices` – The choices of the parameter.
    * `init` – The initialized index of the parameter.

7.5.4.3 Methods

- **accept**

```
public abstract void accept(ParameterVisitor visitor)
```

  - **Description copied from Parameter** (**in 7.5.5, page 132**)
    Let the visitor visit this parameter.
  - **Parameters**
    * `visitor` – The visitor to visit

- **addChoice**

```
public void addChoice(java.lang.String choice,int index)
```

  - **Description**
    Adds a choice to the MultipleChoiceParameter.
  - **Parameters**
    * `choice` – The choice that will be added.
    * `index` – The index in the list the new choice will be added.

- **compareTo**

```
public int compareTo(MultipleChoiceParameter o)
```

- **getChoices**

```
public java.util.List getChoices()
```

  - **Description**
    Returns a list of all set possible choices in the MultipleChoiceParameter.
  - **Returns** – A list of all set possible choices in the MultipleChoiceParameter.

- **getSelectionIndex**

```
public int getSelectionIndex()
```

  - **Description**
    Returns the index of the currently selected choice.
  - **Returns** – The index of the currently selected choice.

- **remove**

```
public void remove(int index)
```

  - **Description**

    Removes a choice from the MultipleChoiceParameter.

  - **Parameters**

    * `index` – The index in the list of the choice to be removed.

- **setValue**

```
public void setValue(int selected)
```

  - **Description**

    Overloads the setValue of GAnsProperty. Sets the String at position selected in values as the value of the Parameter.

  - **Parameters**

    * `selected` – The position in values that has been selected and will be set as value.

7.5.5 Class Parameter

An abstract parameter class. A Parameter contains a value and a name. The value can be transformed into a string. Clients can set Listeners to track changes of the value. Classes inheriting from this class can visited by a ParameterVisitor.

7.5.5.1 Declaration

```
public abstract class Parameter
 extends objectproperty.GAnsProperty implements java.lang.Comparable
```

7.5.5.2 All known subclasses

StringParameter (in 7.5.8, page 135), MultipleChoiceParameter (in 7.5.4, page 130), IntegerParameter (in 7.5.3, page 128), DoubleParameter (in 7.5.2, page 126), BooleanParameter (in 7.5.1, page 125)

7.5.5.3 Constructors

- **Parameter**

```
public Parameter(java.lang.String name,java.lang.Object value)
```

  - **Description**

    Constructor, setting the name and value of the property.

- **Parameters**
  - `name` – The string will be set as the name of the GAnsProperty.
  - `value` – The value that will be set in the GAnsProperty.

7.5.5.4 Methods

- **accept**

```
public abstract void accept(ParameterVisitor visitor)
```

  - **Description**

    Let the visitor visit this parameter.
  - **Parameters**
    - `visitor` – The visitor to visit

- **compareTo**

```
int compareTo(java.lang.Object arg0)
```

7.5.6 Class ParameterVisitor

An abstract visitor class, as described in the Visitor-Pattern. Thought to give other elements access to a custom `Settings` (in 7.5.7, page 134) and retrieve all informations of the specialized `Parameter` (in 7.5.5, page 132) interfaces.

7.5.6.1 Declaration

```
public abstract class ParameterVisitor
 extends java.lang.Object
```

7.5.6.2 Constructors

- **ParameterVisitor**

```
public ParameterVisitor()
```

7.5.6.3 Methods

- **visit**

```
public abstract void visit(IntegerParameter parameter)
```

- **Description**

  Visits the specified parameter and performs some by the subclass chosen actions on it.

- **Parameters**

  * `parameter` – The parameter to visit

- **visit**

```
public abstract void visit(MultipleChoiceParameter parameter)
```

  - **Description**

    Visits the specified parameter and performs some by the subclass chosen actions on it.

  - **Parameters**

    * `parameter` – The parameter to visit

- **visit**

```
public abstract void visit(StringParameter parameter)
```

  - **Description**

    Visits the specified parameter and performs some by the subclass chosen actions on it.

  - **Parameters**

    * `parameter` – The parameter to visit

7.5.7 Class Settings

A compound object to store parameters.

7.5.7.1 Declaration

```
public class Settings
 extends java.lang.Object
```

7.5.7.2 Constructors

- **Settings**

```
public Settings(java.util.Map parameters)
```

  - **Description**

    Constructs a new Settings-Object and sets its parameters.

- **Parameters**
  - `parameters` – The parameters the Settings-Object will have.

7.5.7.3 Methods

- **entrySet**

```
public java.util.Set entrySet()
```

  - **Description**
    Returns a Set of all the parameters.
  - **Returns** – A Set of all the parameters.

- **get**

```
public Parameter get(java.lang.String key)
```

  - **Description**
    Returns the Parameter associated with the given key.
  - **Parameters**
    - `key` – The key which is associated with the Parameter.
  - **Returns** – The Parameter associated with the given key.

- **size**

```
public int size()
```

  - **Description**
    Returns the amount of parameters in the Settings.
  - **Returns** – The amount of parameters in the Settings.

- **values**

```
public java.util.Collection values()
```

  - **Description**
    Returns all the Parameters in the Settings-Object.
  - **Returns** – All the Parameters in the Settings-Object.

7.5.8 Class StringParameter

StringParameter are parameters with an freely set String value space.

7.5.8.1 Declaration

```
public class StringParameter
 extends parameter.Parameter
```

7.5.8.2 Constructors

- **StringParameter**

  ```
  public StringParameter(java.lang.String name,java.lang.String
      value)
  ```

  - **Description**

    Constructs a new StringParameter, sets its name and its default value.
  - **Parameters**
    - `name` – The name of the parameter.
    - `value` – The value of the parameter.

7.5.8.3 Methods

- **accept**

  ```
  public abstract void accept(ParameterVisitor visitor)
  ```

  - **Description copied from Parameter** (**in 7.5.5, page 132**)

    Let the visitor visit this parameter.
  - **Parameters**
    - `visitor` – The visitor to visit

- **compareTo**

  ```
  public int compareTo(StringParameter o)
  ```

7.6 Package sugiyama

Package Contents *Page*

Interfaces

The SugiyamaGraph is a wrapper for a directed graph to enable easy and fast accessibility of attributes and constructs needed during the computation of the hierarchical layout of a directed graph.

7.6.1 Interface ICrossMinimizer

This interface represents a class that takes a Sugiyama Graph and rearranges its vertices on each layer to minimize the amount of edge crossings.

7.6.1.1 Declaration

```
public interface ICrossMinimizer
```

7.6.1.2 All known subinterfaces

CrossMinimizer (in 7.6.13, page 152)

7.6.1.3 All classes known to implement interface

CrossMinimizer (in 7.6.13, page 152)

7.6.1.4 Methods

- **minimizeCrossings**

```
void minimizeCrossings(ICrossMinimizerGraph graph)
```

  - **Description**

    Rearranges vertices in the graph argument in order to remove the amount of crosses of their edges.

  - **Parameters**

    * `graph` – input graph

7.6.2 Interface ICrossMinimizerGraph

A LayeredGraph which additionally defines functions that can be used to minimize the crossings in the sugiyama-layout.

7.6.2.1 Declaration

```
public interface ICrossMinimizerGraph
 extends graphmodel.LayeredGraph
```

7.6.2.2 All known subinterfaces

SugiyamaGraph (in 7.6.18, page 157)

7.6.2.3 All classes known to implement interface

SugiyamaGraph (in 7.6.18, page 157)

7.6.2.4 Methods

- **getLayer**

```
java.util.List getLayer(int layerNum)
```

  - **Description**

    Get all vertices from a certain layer.

  - **Parameters**

    * `layerNum` – the layer number to get all vertices from

  - **Returns** – a list of all vertices which are on this layer

- **getLayer**

```
int getLayer(SugiyamaGraph.SugiyamaVertex vertex)
```

- **Description**

  Get the layer from the vertex

- **Parameters**

  * `vertex` – the vertex to get its layer from

- **Returns** – the layer number from this vertex

- **getLayerCount**

```
int getLayerCount()
```

- **Description**

  Get the amount of layers.

- **Returns** – the amount of layers that contain at least one vertex

- **getLayers**

```
java.util.List getLayers()
```

- **Description**

  Get all layers that contain vertices.

- **Returns** – a list of lists of vertices which are on this layer

- **getVertexCount**

```
int getVertexCount(int layerNum)
```

- **Description**

  Get the number of vertices which are on a certain layer

- **Parameters**

  * `layerNum` – the layer number to get the vertex count from

- **Returns** – the number of vertices which are on this layer

- **swapVertices**

```
void swapVertices(SugiyamaGraph.SugiyamaVertex first,
    SugiyamaGraph.SugiyamaVertex second)
```

- **Description**

  Swaps the position of two vertices that are on the same layer

- **Parameters**
  - `first` – first vertex to change position with
  - `second` – second vertex to change position with

7.6.3 Interface ICycleRemover

This interfaces represents a class that generates a DAG(Directed Acyclic Graph) from a `DirectedGraph` (in 7.3.2, page 59).

7.6.3.1 Declaration

```
public interface ICycleRemover
```

7.6.3.2 All known subinterfaces

CycleRemover (in 7.6.14, page 152)

7.6.3.3 All classes known to implement interface

CycleRemover (in 7.6.14, page 152)

7.6.3.4 Methods

- **removeCycles**

  ```
  java.util.Set removeCycles(ICycleRemoverGraph graph)
  ```

  - **Description**

    Searches for a acyclic subgraph in the graph argument and reversed the direction of the edges that are not part of this subgraph.
  - **Parameters**
    - `graph` – the input graph to remove cycles from
  - **Returns** – a set of edges whose direction has been reversed in order to remove cycles from the graph

7.6.4 Interface ICycleRemoverGraph

A DirectedGraph which additionally defines functions to remove cycles in the graph.

7.6.4.1 Declaration

```
public interface ICycleRemoverGraph
 extends graphmodel.DirectedGraph
```

7.6.4.2 All known subinterfaces

SugiyamaGraph (in 7.6.18, page 157)

7.6.4.3 All classes known to implement interface

SugiyamaGraph (in 7.6.18, page 157)

7.6.4.4 Methods

- **isReversed**

  ```
  boolean isReversed(SugiyamaGraph.SugiyamaEdge edge)
  ```

  - **Description**

    Returns true if the specified edge is reversed, false otherwise
  - **Parameters**
    - `edge` – the edge
  - **Returns** – true if the edge is reversed, false otherwise

- **reverseEdge**

  ```
  void reverseEdge(SugiyamaGraph.SugiyamaEdge edge)
  ```

  - **Description**

    Reverses the direction of a sugiyama edge. The underlying edge won't be reversed to avoid inconsistencies in the underlying graph Instead the reversing will be saved in the SugiyamaEdge. If the edge is already reversed it will be reversed again.
  - **Parameters**
    - `edge` – the edge to reverse its direction

7.6.5 Interface IEdgeDrawer

This interface represents a class that takes a directed graph, as a SugiyamaClass. It removes dummy vertices and reverses previously reversed edges. Afterwards it assigns every edge points it must run through.

7.6.5.1 Declaration

```
public interface IEdgeDrawer
```

7.6.5.2 All known subinterfaces

EdgeDrawer (in 7.6.15, page 153)

7.6.5.3 All classes known to implement interface

EdgeDrawer (in 7.6.15, page 153)

7.6.5.4 Methods

- **drawEdges**

```
void drawEdges(IEdgeDrawerGraph graph)
```

  - **Description**

    Draws the edges from the graph argument and reverses the edges, which have been reversed earlier, so they have now the correct direction.

  - **Parameters**

    * `graph` – the input graph

7.6.6 Interface IEdgeDrawerGraph

A LayeredGraph which additionally defines functions draw the edges in the sugiyama-layout.

7.6.6.1 Declaration

```
public interface IEdgeDrawerGraph
 extends graphmodel.LayeredGraph
```

7.6.6.2 All known subinterfaces

SugiyamaGraph (in 7.6.18, page 157)

7.6.6.3 All classes known to implement interface

SugiyamaGraph (in 7.6.18, page 157)

7.6.6.4 Methods

- **addEdgeCorner**

```
void addEdgeCorner(SugiyamaGraph.SugiyamaEdge edge, int x, int y,
    int index)
```

  - **Description**

    Adds a new edge corner to the specified edge. The index specifies the position between other edge corners. Every edge corner is connected with the corners with index +/- 1 of it's index. Counting starts at 0 at the endpoint at the source vertex of the edge. End- and startpoint are also counted as corners

  - **Parameters**

    * `edge` – the edge to add a new corner

    * `x` – the x coordinate of the corner

    * `y` – the y coordinate of the corner
    * `index` – the index on the edge of the corner

- **getEdgeCorners**

  ```
  java.util.List getEdgeCorners(SugiyamaGraph.SugiyamaEdge edge)
  ```

  - **Description**

    Returns a list of points, which describe the coordinates of the edges
  - **Parameters**
    * `edge` – the edge
  - **Returns** – the list of points of the corners on the edge

- **getReplacedEdges**

  ```
  java.util.Set getReplacedEdges()
  ```

  - **Description**

    Returns the set of replaced edges.
  - **Returns** – the set of replaced edges

- **getReversedEdges**

  ```
  java.util.Set getReversedEdges()
  ```

  - **Description**

    Returns the set of all with `reverseEdge(E edge)` reversed edges.
  - **Returns** – the set of all reversed edges.

- **removeEdgeCorner**

  ```
  void removeEdgeCorner(SugiyamaGraph.SugiyamaEdge edge,int index)
  ```

  - **Description**

    Removes the corner on the specified edge at the index
  - **Parameters**
    * `edge` – the edge to remove the corner
    * `index` – the index of the corner to remove

- **restoreAllEdges**

```
java.util.Set restoreAllEdges()
```

- **Description**

  Deletes the supplement edges, which have been created when an edge was reversed. Adds all reversed edges back to the set of edges and returns them.
- **Returns** – the set of edges, which have been restored.

- **restoreReplacedEdges**

```
java.util.Set restoreReplacedEdges()
```

- **Description**

  Deletes all dummy vertices and edges connecting dummy vertices. Adds the replaced edges back to set of edges.
- **Returns** – the set of edges, which has been restored

- **reverseEdge**

```
void reverseEdge(SugiyamaGraph.SugiyamaEdge edge)
```

- **Description**

  Reverses the direction of an directed edge and returns it.
- **Parameters**
  - `edge` – the edge to return its direction

7.6.7 Interface ILayerAssigner

This interface represents a class that takes a directed graph and assigns every vertex in it a layer.

7.6.7.1 Declaration

```
public interface ILayerAssigner
```

7.6.7.2 All known subinterfaces

LayerAssigner (in 7.6.16, page 154)

7.6.7.3 All classes known to implement interface

LayerAssigner (in 7.6.16, page 154)

7.6.7.4 Methods

- **addConstraints**

  ```
  void addConstraints(java.util.Set constraints)
  ```

  - **Description**

    Defines a set of constraints which should be considered by the algorithm.

  - **Parameters**

    * `constraints` – relative layer constraints the algorithm should consider

- **assignLayers**

  ```
  void assignLayers(ILayerAssignerGraph graph)
  ```

  - **Description**

    Assigns every vertex in the graph parameter e relative height.

  - **Parameters**

    * `graph` – input graph

- **setMaxHeight**

  ```
  void setMaxHeight(int height)
  ```

  - **Description**

    Reassigns the layer of vertices whose layer is greater than the height parameter.

  - **Parameters**

    * `height` – maximum height for vertices

- **setMaxWidth**

  ```
  void setMaxWidth(int width)
  ```

  - **Description**

    Reassigns the layer of vertices in case there are more than the width parameter in one layer.

  - **Parameters**

    * `width` – maximum amount of vertices in one layer

7.6.8 Interface ILayerAssignerGraph

A LayeredGraph which additionally defines functions to assign layers in the sugiyama-layout.

7.6.8.1 Declaration

```
public interface ILayerAssignerGraph
 extends graphmodel.LayeredGraph
```

7.6.8.2 All known subinterfaces

SugiyamaGraph (in 7.6.18, page 157)

7.6.8.3 All classes known to implement interface

SugiyamaGraph (in 7.6.18, page 157)

7.6.8.4 Methods

- **assignToLayer**

  ```
  void assignToLayer(SugiyamaGraph.SugiyamaVertex vertex, int
      layerNum)
  ```

  - **Description**

    Assigns a vertex to a certain layer represented by a number.
  - **Parameters**
    - `vertex` – the vertex to assign to a layer
    - `layerNum` – the layer number to assign a vertex to

- **getLayer**

  ```
  int getLayer(SugiyamaGraph.SugiyamaVertex vertex)
  ```

  - **Description**

    Get the layer from the vertex
  - **Parameters**
    - `vertex` – the vertex to get its layer from
  - **Returns** – the layer number from this vertex

- **getVertexCount**

  ```
  int getVertexCount(int layerNum)
  ```

  - **Description**

    Get the number of vertices which are on a certain layer
  - **Parameters**
    - `layerNum` – the layer number to get the vertex count from
  - **Returns** – the number of vertices which are on this layer

7.6.9 Interface IVertexPositioner

This interface represents a class that takes a directed graph and position its vertices in order to look more clearly. (e.g. position vertices in a row or column) This step has to access the GraphicEngine to get the size of each vertex.

7.6.9.1 Declaration

```
public interface IVertexPositioner
```

7.6.9.2 All known subinterfaces

VertexPositioner (in 7.6.26, page 170)

7.6.9.3 All classes known to implement interface

VertexPositioner (in 7.6.26, page 170)

7.6.9.4 Methods

- **positionVertices**

  ```
  void positionVertices(IVertexPositionerGraph graph)
  ```

  - **Description**

    Sets absolute coordinates for every vertex in the graph. Before this step the vertices are only positioned relatively to each other (layer and order in one layer)

  - **Parameters**

    * `graph` – input graph

7.6.10 Interface IVertexPositionerGraph

A LayeredGraph which additionally defines functions to position vertices in the sugiyama-layout.

7.6.10.1 Declaration

```
public interface IVertexPositionerGraph
 extends graphmodel.LayeredGraph
```

7.6.10.2 All known subinterfaces

SugiyamaGraph (in 7.6.18, page 157)

7.6.10.3 All classes known to implement interface

SugiyamaGraph (in 7.6.18, page 157)

7.6.10.4 Methods

- **setLayerY**

```
void setLayerY(int layerN,int y)
```

  - **Description**

    Sets the y-coordinate of all vertices on layer Y.

  - **Parameters**

    * `layerN` – the index of the layer
    * `y` – the y-coordinate

- **setX**

```
void setX(SugiyamaGraph.SugiyamaVertex vertex,int x)
```

  - **Description**

    Sets the x-coordinate of the specified vertex

  - **Parameters**

    * `vertex` – the vertex to position
    * `x` – the x-coordinate

7.6.11 Interface LayeredLayoutAlgorithm

A layout algorithm which is able to apply Layouts while keeping the notion of layers in addition to layout regular graphs.

7.6.11.1 Declaration

```
public interface LayeredLayoutAlgorithm
 extends plugin.LayoutAlgorithm
```

7.6.11.2 All known subinterfaces

SugiyamaLayoutAlgorithm (in 7.6.24, page 166)

7.6.11.3 All classes known to implement interface

SugiyamaLayoutAlgorithm (in 7.6.24, page 166)

7.6.11.4 Methods

- **layoutLayeredGraph**

```
void layoutLayeredGraph(graphmodel.LayeredGraph graph)
```

  - **Description**

    Applies its layout to a graph as in `layout(G graph)` but keeps the notion of layers. The algorithm will assign every vertex a coordinate and every edge a path. Additionally every vertex will be assigned a position in a layer in the LayeredGraph. A possible application is drawing of recursive graphs.

  - **Parameters**

    * `graph` – the graph to apply the layout to

7.6.12 Class AbsoluteLayerConstraint

A absolute constraint, regarding layer assignment, for one set of vertices. Can describe if one set of vertices should be placed in one layer, in a range of layers. The constraint can be inverted, meaning it should not be placed on this layer. Additionally an exclusive flag can be set to mark that only this set of vertices should be placed on the selected range of layer. Consequently inverted and exclusive means all other vertices have to be placed in this range of layers.

7.6.12.1 Declaration

```
public class AbsoluteLayerConstraint
 extends java.lang.Object implements plugin.Constraint
```

7.6.12.2 Constructors

- **AbsoluteLayerConstraint**

```
public AbsoluteLayerConstraint(java.util.Set set,boolean
    inverted,boolean exclusive,int minLayer,int maxLayer)
```

  - **Description**

    Constructs an AbsoluteLayerConstraint.

  - **Parameters**

    * `set` – set of sugiyama vertices to apply this constraint on
    * `inverted` – invertes this constraint, if true
    * `exclusive` – places only this set of vertices in this range of layers
    * `minLayer` – lower bound of layer number
    * `maxLayer` – upper bound of layer number

7.6.12.3 Methods

- **getMaxLayer**

  ```
  public int getMaxLayer()
  ```

  - **Description**

    Returns the maximum layer the vertices should be on.
  - **Returns** – the maximum layer

- **getMinLayer**

  ```
  public int getMinLayer()
  ```

  - **Description**

    Returns the minimum layer the vertices should be on.
  - **Returns** – the minimum layer

- **getName**

  ```
  java.lang.String getName()
  ```

  - **Description copied from plugin.Constraint** (**in 7.1.1, page 29**)

    The name of the constraint
  - **Returns** – the name

- **getVertices**

  ```
  public java.util.Set getVertices()
  ```

  - **Description**

    Returns the set of vertices which should be affected by the constraint.
  - **Returns** – the set of vertices

- **isExclusive**

  ```
  public boolean isExclusive()
  ```

  - **Description**

    Returns true if the set of vertices should be affected by the constraint exclusively.
  - **Returns** – true if exclusive

- **isInverted**

```
public boolean isInverted()
```

  - **Description**

    Returns true if the constraint should be inverted.
  - **Returns** – true if inverted

7.6.13 Class CrossMinimizer

This class takes a Sugiyama Graph and rearranges its vertices on each layer to minimize the amount of edge crossings.

7.6.13.1 Declaration

```
public class CrossMinimizer
 extends java.lang.Object implements ICrossMinimizer
```

7.6.13.2 Constructors

- **CrossMinimizer**

```
public CrossMinimizer()
```

7.6.13.3 Methods

- **minimizeCrossings**

```
void minimizeCrossings(ICrossMinimizerGraph graph)
```

  - **Description copied from ICrossMinimizer** (**in 7.6.1, page 138**)

    Rearranges vertices in the graph argument in order to remove the amount of crosses of their edges.
  - **Parameters**
    - `graph` – input graph

7.6.14 Class CycleRemover

This class takes a directed Graph G = (V, E) and removes a set of edges E_ so that the resulting Graph G' = (V, E\E_) is a DAG(Directed Acyclic Graph).

7.6.14.1 Declaration

```
public class CycleRemover
 extends java.lang.Object implements ICycleRemover
```

7.6.14.2 Constructors

- **CycleRemover**

  ```
  public CycleRemover()
  ```

7.6.14.3 Methods

- **removeCycles**

  ```
  java.util.Set removeCycles(ICycleRemoverGraph graph)
  ```

  - **Description copied from ICycleRemover** (**in 7.6.3, page 141**)

    Searches for a acyclic subgraph in the graph argument and reversed the direction of the edges that are not part of this subgraph.
  - **Parameters**
    - `graph` – the input graph to remove cycles from
  - **Returns** – a set of edges whose direction has been reversed in order to remove cycles from the graph

7.6.15 Class EdgeDrawer

This class takes a directed graph, as a SugiyamaClass. It removes dummy vertices and reverses previously reversed edges. Afterwards it assigns every edge points it must run through.

7.6.15.1 Declaration

```
public class EdgeDrawer
 extends java.lang.Object implements IEdgeDrawer
```

7.6.15.2 Constructors

- **EdgeDrawer**

  ```
  public EdgeDrawer()
  ```

7.6.15.3 Methods

- **drawEdges**

  ```
  void drawEdges(IEdgeDrawerGraph graph)
  ```

  - **Description copied from IEdgeDrawer** (**in 7.6.5, page 142**)

    Draws the edges from the graph argument and reverses the edges, which have been reversed earlier, so they have now the correct direction.

  - **Parameters**

    * `graph` – the input graph

7.6.16 Class LayerAssigner

This class takes a directed graph and assigns every vertex in it a layer.

7.6.16.1 Declaration

```
public class LayerAssigner
 extends java.lang.Object implements ILayerAssigner
```

7.6.16.2 Constructors

- **LayerAssigner**

  ```
  public LayerAssigner()
  ```

7.6.16.3 Methods

- **addConstraints**

  ```
  void addConstraints(java.util.Set constraints)
  ```

  - **Description copied from ILayerAssigner** (**in 7.6.7, page 145**)

    Defines a set of constraints which should be considered by the algorithm.

  - **Parameters**

    * `constraints` – relative layer constraints the algorithm should consider

- **assignLayers**

  ```
  void assignLayers(ILayerAssignerGraph graph)
  ```

- **Description copied from ILayerAssigner** (**in 7.6.7, page 145**)

  Assigns every vertex in the graph parameter e relative height.

- **Parameters**

  * `graph` – input graph

- **setMaxHeight**

```
void setMaxHeight(int height)
```

  - **Description copied from ILayerAssigner** (**in 7.6.7, page 145**)

    Reassigns the layer of vertices whose layer is greater than the height parameter.

  - **Parameters**

    * `height` – maximum height for vertices

- **setMaxWidth**

```
void setMaxWidth(int width)
```

  - **Description copied from ILayerAssigner** (**in 7.6.7, page 145**)

    Reassigns the layer of vertices in case there are more than the width parameter in one layer.

  - **Parameters**

    * `width` – maximum amount of vertices in one layer

7.6.17 Class RelativeLayerConstraint

A relative constraint, regarding layer assignment, between to sets of vertices. Can describe if one set of vertices should be on top of the other. When the exact is set a layer distance can be set.

7.6.17.1 Declaration

```
public class RelativeLayerConstraint
 extends java.lang.Object implements plugin.Constraint
```

7.6.17.2 Constructors

- **RelativeLayerConstraint**

```
public RelativeLayerConstraint(java.util.Set top,java.util.Set
    bottom,boolean direct,int distance)
```

- **Description**

  Constructs a new RelativeLayerConstraint, sets the top and bottom vertices, whether its direct and the distance.

- **Parameters**
  - `top` – The top vertices.
  - `bottom` – The bottom vertices.
  - `direct` – True is direct, false is not direct.
  - `distance` – The distance between top and bottom layer.

7.6.17.3 Methods

- **bottomSet**

```
public java.util.Set bottomSet()
```

  - **Description**

    Returns the set which should be below.

  - **Returns** – the bottom layer

- **getDistance**

```
public int getDistance() throws java.lang.IllegalStateException
```

  - **Description**

    Returns the distance the two sets should be apart, if this constraint is exact.

  - **Returns** – the number of layers between the sets
  - **Throws**
    - `java.lang.IllegalStateException` – if the set is not exact

- **getName**

```
public java.lang.String getName()
```

  - **Description**

    Returns the name of the layout constraint

- **isExact**

```
public boolean isExact()
```

- **Description**

  Returns true if the constraints describes an exact distance between the two sets, false otherwise.

- **Returns** – true if exact

- **topSet**

```
public java.util.Set topSet()
```

  - **Description**

    Returns the set which should be on top.

  - **Returns** – the top layer

7.6.18 Class SugiyamaGraph

The SugiyamaGraph is a wrapper for a directed graph to enable easy and fast accessibility of attributes and constructs needed during the computation of the hierarchical layout of a directed graph. All vertices are assigned to a layer. The positions of the vertices can be viewed as a grid (with varying widths per layer).

7.6.18.1 Declaration

```
public class SugiyamaGraph
 extends java.lang.Object implements ICycleRemoverGraph,
    ILayerAssignerGraph, ICrossMinimizerGraph,
    IVertexPositionerGraph, IEdgeDrawerGraph
```

7.6.18.2 Constructors

- **SugiyamaGraph**

```
public SugiyamaGraph(graphmodel.DirectedGraph graph)
```

  - **Description**

    Constructs a new SugiyamaGraph and sets the Graph which is the underlying representation. To fulfill the invariant that all vertices are assigned to a layer, all vertices will be assigned to layer 0.

  - **Parameters**

    * `graph` – the graph used as underlying representation.

7.6.18.3 Methods

- **addEdgeCorner**

```
void addEdgeCorner(SugiyamaGraph.SugiyamaEdge edge, int x, int y,
    int index)
```

  - **Description copied from IEdgeDrawerGraph** (**in 7.6.6, page 143**)

    Adds a new edge corner to the specified edge. The index specifies the position between other edge corners. Every edge corner is connected with the corners with index +/- 1 of it's index. Counting starts at 0 at the endpoint at the source vertex of the edge. End- and startpoint are also counted as corners

  - **Parameters**
    - `edge` – the edge to add a new corner
    - `x` – the x coordinate of the corner
    - `y` – the y coordinate of the corner
    - `index` – the index on the edge of the corner

- **addToFastGraphAccessor**

```
public void addToFastGraphAccessor(graphmodel.FastGraphAccessor
    fga)
```

- **assignToLayer**

```
void assignToLayer(SugiyamaGraph.SugiyamaVertex vertex, int
    layerNum)
```

  - **Description copied from ILayerAssignerGraph** (**in 7.6.8, page 146**)

    Assigns a vertex to a certain layer represented by a number.

  - **Parameters**
    - `vertex` – the vertex to assign to a layer
    - `layerNum` – the layer number to assign a vertex to

- **edgesOf**

```
public java.util.Set edgesOf(SugiyamaGraph.SugiyamaVertex vertex
    )
```

- **getEdgeCorners**

```
java.util.List getEdgeCorners(SugiyamaGraph.SugiyamaEdge edge)
```

- **Description copied from IEdgeDrawerGraph** (**in 7.6.6, page 143**)
  Returns a list of points, which describe the coordinates of the edges
- **Parameters**
  - `edge` – the edge
- **Returns** – the list of points of the corners on the edge

- **getEdgeSet**

```
public java.util.Set getEdgeSet()
```

- **getFastGraphAccessor**

```
public graphmodel.FastGraphAccessor getFastGraphAccessor()
```

- **getHeight**

```
public int getHeight()
```

- **getID**

```
public java.lang.Integer getID()
```

- **getLayer**

```
java.util.List getLayer(int layerNum)
```

- **Description copied from ICrossMinimizerGraph** (**in 7.6.2, page 139**)
  Get all vertices from a certain layer.
- **Parameters**
  - `layerNum` – the layer number to get all vertices from
- **Returns** – a list of all vertices which are on this layer

- **getLayer**

```
int getLayer(SugiyamaGraph.SugiyamaVertex vertex)
```

- **Description copied from ILayerAssignerGraph** (**in 7.6.8, page 146**)
  Get the layer from the vertex

- **Parameters**
  - `vertex` – the vertex to get its layer from
- **Returns** – the layer number from this vertex

- **getLayerCount**

```
int getLayerCount()
```

- **Description copied from ICrossMinimizerGraph** (**in 7.6.2, page 139**)
  Get the amount of layers.
- **Returns** – the amount of layers that contain at least one vertex

- **getLayers**

```
java.util.List getLayers()
```

- **Description copied from ICrossMinimizerGraph** (**in 7.6.2, page 139**)
  Get all layers that contain vertices.
- **Returns** – a list of lists of vertices which are on this layer

- **getLayerWidth**

```
public int getLayerWidth(int layerN)
```

- **getMaxWidth**

```
public int getMaxWidth()
```

- **getName**

```
public java.lang.String getName()
```

- **getReplacedEdges**

```
java.util.Set getReplacedEdges()
```

- **Description copied from IEdgeDrawerGraph** (**in 7.6.6, page 143**)
  Returns the set of replaced edges.
- **Returns** – the set of replaced edges

- **getReversedEdges**

```
java.util.Set getReversedEdges()
```

- **Description copied from IEdgeDrawerGraph** (**in 7.6.6, page 143**)
  Returns the set of all with `reverseEdge(E edge)` reversed edges.
- **Returns** – the set of all reversed edges.

- **getSubgraphs**

```
public java.util.List getSubgraphs()
```

- **getVertexCount**

```
int getVertexCount(int layerNum)
```

- **Description copied from ILayerAssignerGraph** (**in 7.6.8, page 146**)
  Get the number of vertices which are on a certain layer
- **Parameters**
  - `layerNum` – the layer number to get the vertex count from
- **Returns** – the number of vertices which are on this layer

- **getVertexSet**

```
public java.util.Set getVertexSet()
```

- **incomingEdgesOf**

```
public java.util.Set incomingEdgesOf(SugiyamaGraph.
    SugiyamaVertex vertex)
```

- **indegreeOf**

```
public java.lang.Integer indegreeOf(SugiyamaGraph.SugiyamaVertex
     vertex)
```

- **isReversed**

```
boolean isReversed(SugiyamaGraph.SugiyamaEdge edge)
```

- **Description copied from ICycleRemoverGraph** (**in 7.6.4, page 141**)
  Returns true if the specified edge is reversed, false otherwise
- **Parameters**

* `edge` – the edge

– **Returns** – true if the edge is reversed, false otherwise

- **outdegreeOf**

```
public java.lang.Integer outdegreeOf(SugiyamaGraph.
    SugiyamaVertex vertex)
```

- **outgoingEdgesOf**

```
public java.util.Set outgoingEdgesOf(SugiyamaGraph.
    SugiyamaVertex vertex)
```

- **removeEdgeCorner**

```
void removeEdgeCorner(SugiyamaGraph.SugiyamaEdge edge, int index)
```

– **Description copied from IEdgeDrawerGraph** (**in 7.6.6, page 143**)

Removes the corner on the specified edge at the index

– **Parameters**

* `edge` – the edge to remove the corner
* `index` – the index of the corner to remove

- **restoreAllEdges**

```
java.util.Set restoreAllEdges()
```

– **Description copied from IEdgeDrawerGraph** (**in 7.6.6, page 143**)

Deletes the supplement edges, which have been created when an edge was reversed. Adds all reversed edges back to the set of edges and returns them.

– **Returns** – the set of edges, which have been restored.

- **restoreReplacedEdges**

```
java.util.Set restoreReplacedEdges()
```

– **Description copied from IEdgeDrawerGraph** (**in 7.6.6, page 143**)

Deletes all dummy vertices and edges connecting dummy vertices. Adds the replaced edges back to set of edges.

– **Returns** – the set of edges, which has been restored

- **reverseEdge**

```
void reverseEdge(SugiyamaGraph.SugiyamaEdge edge)
```

  - **Description copied from ICycleRemoverGraph** (**in 7.6.4, page 141**)

    Reverses the direction of a sugiyama edge. The underlying edge won't be reversed to avoid inconsistencies in the underlying graph Instead the reversing will be saved in the SugiyamaEdge. If the edge is already reversed it will be reversed again.
  - **Parameters**
    - `edge` – the edge to reverse its direction

- **setLayerY**

```
void setLayerY(int layerN, int y)
```

  - **Description copied from IVertexPositionerGraph** (**in 7.6.10, page 148**)

    Sets the y-coordinate of all vertices on layer Y.
  - **Parameters**
    - `layerN` – the index of the layer
    - `y` – the y-coordinate

- **setX**

```
void setX(SugiyamaGraph.SugiyamaVertex vertex, int x)
```

  - **Description copied from IVertexPositionerGraph** (**in 7.6.10, page 148**)

    Sets the x-coordinate of the specified vertex
  - **Parameters**
    - `vertex` – the vertex to position
    - `x` – the x-coordinate

- **swapVertices**

```
void swapVertices(SugiyamaGraph.SugiyamaVertex first,
    SugiyamaGraph.SugiyamaVertex second)
```

  - **Description copied from ICrossMinimizerGraph** (**in 7.6.2, page 139**)

    Swaps the position of two vertices that are on the same layer
  - **Parameters**
    - `first` – first vertex to change position with
    - `second` – second vertex to change position with

7.6.19 Class SugiyamaGraph.DummyVertex

A supplement vertex which is part of a `SugiyamaGraph.SupplementPath` (in 7.6.23, page 165).

7.6.19.1 Declaration

```
public static class SugiyamaGraph.DummyVertex
 extends graphmodel.DefaultVertex
```

7.6.19.2 Constructors

- **DummyVertex**

```
public DummyVertex()
```

7.6.20 Class SugiyamaGraph.SugiyamaEdge

A wrapper class for directed edges to implement additional functionality to apply the sugiyama layout to the SugiyamaGraph containing them.

7.6.20.1 Declaration

```
public static class SugiyamaGraph.SugiyamaEdge
 extends graphmodel.DirectedEdge
```

7.6.21 Class SugiyamaGraph.SugiyamaVertex

A wrapper class for vertices used in the sugiyama framework. A SugiyamaVertex can be a `DefaultVertex` (in 7.3.14, page 78) or a `SugiyamaGraph.DummyVertex` (in 7.6.19, page 164)

7.6.21.1 Declaration

```
public static class SugiyamaGraph.SugiyamaVertex
 extends graphmodel.DefaultVertex
```

7.6.21.2 Constructors

- **SugiyamaVertex**

```
public SugiyamaVertex()
```

7.6.21.3 Methods

- **isDummyVertex**

```
public boolean isDummyVertex()
```

7.6.22 Class SugiyamaGraph.SupplementEdge

A supplement edge which is part of a `SugiyamaGraph.SupplementPath` (in 7.6.23, page 165).

7.6.22.1 Declaration

```
public static class SugiyamaGraph.SupplementEdge
 extends graphmodel.DirectedEdge
```

7.6.22.2 Constructors

- **SupplementEdge**

  ```
  public SupplementEdge()
  ```

7.6.23 Class SugiyamaGraph.SupplementPath

A supplement path for connecting vertices, which are more than one layer apart. They are stored in the SugiyamaEdge along with the substituted edge.

7.6.23.1 Declaration

```
public static class SugiyamaGraph.SupplementPath
 extends graphmodel.DirectedEdge
```

7.6.23.2 Constructors

- **SupplementPath**

  ```
  public SupplementPath()
  ```

7.6.23.3 Methods

- **getDummyVertices**

  ```
  public java.util.List getDummyVertices()
  ```

  - **Description**

    Returns the list of vertices on the path sorted from source to target excluding the source and target.
  - **Returns** – the list of vertices

- **getEdges**

  ```
  public java.util.List getEdges()
  ```

- **Description**

  Returns the list of edges on the path from source to target

- **Returns** – the edges

- **getLength**

```
public int getLength()
```

  - **Description**

    Returns the number of vertices including source and target.

  - **Returns** – the length of the path

7.6.24 Class SugiyamaLayoutAlgorithm

This class supports a customizable implementation of the Sugiyama-framework. The single stages of the framework can be chosen individually. Additionally this class tries to follow the given constraints, if possible.

7.6.24.1 Declaration

```
public class SugiyamaLayoutAlgorithm
 extends java.lang.Object implements LayeredLayoutAlgorithm
```

7.6.24.2 Constructors

- **SugiyamaLayoutAlgorithm**

```
public SugiyamaLayoutAlgorithm()
```

7.6.24.3 Methods

- **addAbsoluteLayerConstraint**

```
public void addAbsoluteLayerConstraint(AbsoluteLayerConstraint
    constraint)
```

  - **Description**

    Adds an `AbsoluteLayerConstraint` (in 7.6.12, page 150) to the set of constraints which should be followed.

  - **Parameters**

    * `constraint` – the constraint to follow

- **addRelativeLayerConstraint**

```
public void addRelativeLayerConstraint(RelativeLayerConstraint
    constraint)
```

- **Description**

  Adds an `RelativeLayerConstraint` (in 7.6.17, page 155) to the set of constraints which should be followed.

- **Parameters**

  * `constraint` – the constraint to follow

- **getSettings**

```
public parameter.Settings getSettings()
```

- **layout**

```
public void layout(graphmodel.DirectedGraph graph)
```

- **layoutLayeredGraph**

```
void layoutLayeredGraph(graphmodel.LayeredGraph graph)
```

- **Description copied from LayeredLayoutAlgorithm** (**in 7.6.11, page 149**)

  Applies its layout to a graph as in `layout(G graph)` but keeps the notion of layers. The algorithm will assign every vertex a coordinate and every edge a path. Additionally every vertex will be assigned a position in a layer in the LayeredGraph. A possible application is drawing of recursive graphs.

- **Parameters**

  * `graph` – the graph to apply the layout to

- **setCrossMinimizer**

```
public void setCrossMinimizer(ICrossMinimizer minimizer)
```

- **Description**

  Sets the algorithm for cross minimization used when applying the layout

- **Parameters**

  * `minimizer` – the cross minimization algorithm

- **setCycleRemover**

```
public void setCycleRemover(ICycleRemover remover)
```

- **Description**

  Sets the algorithm to remove all cycles for a graph to layout used when applying the layout

- **Parameters**

  * `remover` – the algorithm

- **setEdgeDrawer**

```
public void setEdgeDrawer(IEdgeDrawer drawer)
```

  - **Description**

    Sets the algorithm for edge drawing used when applying the layout

  - **Parameters**

    * `drawer` – the edge drawing algorithm

- **setLayerAssigner**

```
public void setLayerAssigner(ILayerAssigner assigner)
```

  - **Description**

    Sets the algorithm for layer assigning used when applying the layout

  - **Parameters**

    * `assigner` – the layer assign algorithm

- **setVertexPositioner**

```
public void setVertexPositioner(IVertexPositioner positioner)
```

  - **Description**

    Sets the algorithm for vertex positioning used when applying the layout

  - **Parameters**

    * `positioner` – the positioning algorithm

7.6.25 Class SugiyamaPlugin

A plugin for GAns that supplies a layout algorithm based on the Sugiyama-framework.

7.6.25.1 Declaration

```
public class SugiyamaPlugin
 extends java.lang.Object implements plugin.Plugin
```

7.6.25.2 Constructors

- **SugiyamaPlugin**

  ```
  public SugiyamaPlugin()
  ```

7.6.25.3 Methods

- **getEdgeFilter**

  ```
  java.util.List getEdgeFilter()
  ```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

    Returns all by the plugin provided `EdgeFilter` (in 7.1.8, page 35). If none are provided returns `null` or an empty list.
  - **Returns** – the list of provided edge filter

- **getExporter**

  ```
  java.util.List getExporter()
  ```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

    Returns all by the plugin provided `Exporter` (in 7.1.2, page 29). If none are provided returns `null` or an empty list.
  - **Returns** – a list of provided exporter

- **getImporter**

  ```
  java.util.List getImporter()
  ```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)

    Returns all by the plugin provided `Importer` (in 7.1.3, page 30). If none are provided returns `null` or an empty list.
  - **Returns** – a list of provided importer

- **getName**

  ```
  java.lang.String getName()
  ```

- **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)
  Returns the name of the plugin. Uniqueness can't be assumed.
- **Returns** – the name of the plugin

- **getVertexFilter**

```
java.util.List getVertexFilter()
```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)
    Returns all by the plugin provided `VertexFilter` (in 7.1.12, page 40). If none are provided returns `null` or an empty list.
  - **Returns** – the list of provided vertex filter

- **getWorkspaceOptions**

```
java.util.List getWorkspaceOptions()
```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)
    Returns all provided by the plugin `WorkspaceOption` (in 7.1.13, page 41). If none are provided returns `null` or an empty list.
  - **Returns** – The list of provided workspace options

- **load**

```
void load()
```

  - **Description copied from plugin.Plugin** (**in 7.1.6, page 32**)
    Called after all plugins have been constructed. Ïnter-Plugincommunication, like registering of layouts for graphs in other plugins should be executed in here.

7.6.26 Class VertexPositioner

This class takes a directed graph and position its vertices in order to look more clearly. (e.g. position vertices in a row or column)

7.6.26.1 Declaration

```
public class VertexPositioner
 extends java.lang.Object implements IVertexPositioner
```

7.6.26.2 Constructors

- **VertexPositioner**

```
public VertexPositioner()
```

7.6.26.3 Methods

- **positionVertices**

  ```
  void positionVertices(IVertexPositionerGraph graph)
  ```

  - **Description copied from IVertexPositioner** (**in 7.6.9, page 148**)

    Sets absolute coordinates for every vertex in the graph. Before this step the vertices are only positioned relatively to each other (layer and order in one layer)

  - **Parameters**
    - `graph` – input graph

7.7 Package objectproperty

Package Contents *Page*

Classes

7.7.1 Class GAnsObjectProperty

A JavaFX object property which in toString() returns the toString() of the value.

7.7.1.1 Declaration

```
public class GAnsObjectProperty
 extends javafx.beans.property.SimpleObjectProperty
```

7.7.1.2 Constructors

- **GAnsObjectProperty**

  ```
  public GAnsObjectProperty(java.lang.Object parent,java.lang.
      String name)
  ```

  - **Description**

    Constructor.

  - **Parameters**
    - `parent` – object that contains the GAnsObjectProperty.
    - `name` – name that will be given to the SimpleObjectProperty constructor.

7.7.1.3 Methods

- **toString**

```
public java.lang.String toString()
```

7.7.2 Class GAnsProperty

The GAnsProperty is a single property of a vertex or an edge in GAns. The name and value of the property are being stored in JavaFX-Properties. A graph consists out of vertices and edges which contain GAnsProperties, which are linked to the GUI-elements.

7.7.2.1 Declaration

```
public class GAnsProperty
 extends java.lang.Object
```

7.7.2.2 Fields

- `public static final java.lang.String` **name**
  - A string with whom, factories or other elements from the GUI, can reference to the name/identifier of the value.
- `public static final java.lang.String` **value**
  - A string with whom, factories or other elements from the GUI, can reference to the value.
- `public static final java.lang.String` **valueAsString**
  - A string with whom, factories or other elements from the GUI, can reference to the string-representation of the value.

7.7.2.3 Constructors

- **GAnsProperty**

```
public GAnsProperty(java.lang.String name,java.lang.Object value
    )
```

  - **Description**

    Constructor, setting the name and value of the property.

  - **Parameters**
    - `name` – The string will be set as the name of the GAnsProperty.
    - `value` – The value that will be set in the GAnsProperty.

7.7.2.4 Methods

- **addListenerToValue**

  ```
  public void addListenerToValue(javafx.beans.value.ChangeListener
      listener)
  ```

  - **Description**

    Adds a ChangeListener to the value of the property.

  - **Parameters**

    * `listener` – The listener that will be added.

- **getName**

  ```
  public java.lang.String getName()
  ```

  - **Description**

    Returns the name/identifier of the GAnsProperty.

  - **Returns** – The name/identifier of the GAnsProperty.

- **getValue**

  ```
  public java.lang.Object getValue()
  ```

  - **Description**

    Returns the value of the GAnsProperty.

  - **Returns** – The value of the GAnsProperty.

- **getValueAsString**

  ```
  public java.lang.String getValueAsString()
  ```

  - **Description**

    Returns the string-representation of the value from the GAnsProperty.

  - **Returns** – The string-representation of the value from the GAnsProperty.

- **propertyNameProperty**

  ```
  public javafx.beans.property.StringProperty propertyNameProperty
      ()
  ```

- **Description**

  Ensures that the property which contains the name/identifier of the GAnsProperty is not null and always set with the right name.

- **Returns** – The StringProperty which contains the name/identifier of the GAnsProperty.

- **propertyValue**

```
public GAnsObjectProperty propertyValue()
```

  - **Description**

    Ensures that the property which contains the value in the GAnsProperty is not null and always set with the right name.

  - **Returns** – The property which contains the value in the GAnsProperty.

- **propertyValueAsString**

```
public javafx.beans.property.StringProperty
    propertyValueAsString()
```

  - **Description**

    Ensures that the property which contains the string-representation of the value in the GAnsProperty is not null and always set with the right name.

  - **Returns** – The StringProperty which contains the string-representation of the value of the GAnsProperty.

- **removeListenerFromValue**

```
public void removeListenerFromValue(javafx.beans.value.
    ChangeListener listener)
```

  - **Description**

    Removes a ChangeListener from the value of the property.

  - **Parameters**

    * `listener` – The listener that will be removed.

- **setName**

```
public void setName(java.lang.String value)
```

  - **Description**

    Sets the name/identifier of the GAnsProperty.

- **Parameters**
  - `value` – The string will be set as the name of the GAnsProperty.

- **setValue**

```
public void setValue(java.lang.Object value)
```

  - **Description**

    Sets the value and its string-representation of the GAnsProperty.
  - **Parameters**
    - `value` – The value that will be set in the GAnsProperty.