Die Programmierschnittstelle des Tools VinetS

1. Download und Installation des VinetS-Graph-Editors

Das Demoarchiv finden Sie – zusammen mit einer Anleitung zum Herunterladen – auf der Seite **`http://vinets.fh-stralsund.de/demo/index.html`**. Die Datei **`vinets-demo.zip`** enthält die Archive **`vinets.jar`** und **`jdom.jar`** sowie die API-Beschreibung **`javadoc.zip`** der für Algorithmen-Entwickler relevanten Pakete **`de.fhstralsund.vinets.algorithm`**, **`de.fhstralsund.vinets.structure`** und **`de.fhstralsund.vinets.geometry`**.

Für die Arbeit im PC-Labor finden Sie die Datei **`vinets-demo.zip`** auch in meinem Dozentenverzeichnis. Entpacken Sie diese in Ihr Projektverzeichnis und binden Sie die beiden Archive **`vinets.jar`** und **`jdom.jar`** in den Bibliothekspfad Ihres Projektes ein.

Entpacken Sie auch **`javadoc.zip`** ins doc-Verzeichnis.

Kopieren Sie sich außerdem das im folgenden Abschnitt beschriebene Paket **`uebung1`** in Ihr Quellverzeichnis und testen Sie damit Ihre Installation.

2. Einbinden eigener Graphalgorithmen

2.1. Eine einfache Beispielapplikation: Knotenanzahl des Graphen bestimmen

Ein Algorithmus, der auf dem im VinetS-Editor angezeigten Graphen ausgeführt werden soll, muss in einer Java-Klasse implementiert sein, der das `Algorithm`- Interface mit seinen vier Methoden implementiert. Dies könnte beispielsweise so aussehen:

```
package uebung1;

import de.fhstralsund.vinets.algorithm.Algorithm;
import de.fhstralsund.vinets.algorithm.Parameter;
import de.fhstralsund.vinets.structure.Graph;
import de.fhstralsund.vinets.structure.Node;
import de.fhstralsund.vinets.structure.Edge;

public class KnotenAnzahl implements Algorithm {

   public boolean accept(Parameter p) {
      return true;
   }

   public Parameter execute (Parameter p) {
      Graph g = p.getGraph();
      int n = g.countNodes();
      String s = "Der Graph hat " + n + " Knoten.";
      return new Parameter(g, s);
   }

   public String getHint() {
      return "Knotenanzahl bestimmen";
   }

   public String getName() { return "Knoten"; }
}
```

Zusätzlich muss dieser Algorithmus in die Applikation eingebunden werden, dies geschieht durch eine Modifikation der `Start`-Klasse von VinetS wie folgt:

```
package uebung1;

import de.fhstralsund.vinets.control.GraphApplication;

public class Start {

   public static void main(String[] unused){
      GraphApplication GA = GraphApplication.getInstance();
      GA.attachAlgorithm("uebung1.KnotenAnzahl");
   }

}
```

Der Methode **`attachAlgorithm( )`** muss hierbei der vollständige Klassenname des neuen Algorithmus übergeben werden. Sie bewirkt, dass ein Item im Algorithm-Menü mit der Beschriftung erzeugt wird, die von der **`getName()`**-Methode des Algorithmus geliefert wird. Das Menü-Item ist immer dann aktiv, wenn im Editorfenster ein Graph angezeigt wird, auf den der aktuelle Algorithmus anwendbar ist. Dies ist der Fall, wenn die Methode **`accept( )`** **`true`** liefert. Wählt der Nutzer das Item, wird die **`execute( )`**-Methode auf den angezeigten Graphen angewendet. Danach wird der zurückgegebene Parameter, der sich aus einem Graphen und einer Meldung zusammensetzt, entsprechend im Editor- bzw. Meldungsfenster angezeigt.

Der Algorithmus kann dabei den übergebenen Graphen mit den im nächsten Kapitel 3 beschriebenen Operationen der Graph-Datenstruktur auswerten oder verändern. Im Beispiel wird einfach die Größe der Knotenmenge des Graphen abgefragt und als Meldung geliefert, der Graph selbst wird vom Algorithmus nicht verändert.

Soll obiges Beispiel in Ihrer IDE compiliert und ausgeführt werden, muss in den Projekteigenschaften der Bibliothekspfad zu den Archiven `vinets.jar` und `jdom.jar` gesetzt sein. Die beiden oben gegebenen Klassen müssen sich im Quellpfad befinden.

2.2. Kurzdokumentation der Algorithm- und Parameter-Schnittstellen

`public interface de.fhstralsund.vinets.algorithm.Algorithm`

A class implements the interface Algorithm in order to be able to examine or to manipulate the Graph shown by the GraphApplication.

Method Summary	
`boolean`	`accept(Parameter params)` Checks whether the passed Parameter would be accepted by this algorithm as input parameter.
`Parameter`	`execute(Parameter params)` A method that examines the graph passed within the Parameter object. Please note that an algorithm may be invoked multiple times, therefore required initialization should be done for EVERY execute call!
`java.lang.String`	`getHint()` Returns a help text for the user.
`java.lang.String`	`getName()` Returns a String that should occur on the corresponding button of the GraphApplication.

`public class de.fhstralsund.vinets.algorithm.Parameter`

A class to encapsulate a graph and additional parameters/results to be passed from and to algorithms. Additionally a message String may be provided.

Constructor Summary	
`Parameter(Graph theGraph, java.lang.String theMessage)`	
Method Summary	
`Graph`	`getGraph()` Returns the encapsulated Graph object.
`java.lang.String`	`getMessage()` Returns the encapsulated message.
`java.lang.Object`	`getProperty(java.lang.Object key)` Returns the value associated with the passed key.
`void`	`setMessage(java.lang.String s)` Changes the encapsulated message.
`void`	`setProperty(java.lang.Object key, java.lang.Object value)` Adds a new property to be stored inside this Parameter object.

Jedes `Parameter`-Objekt verwaltet intern verschiedene Merkmale in einer `HashMap`.

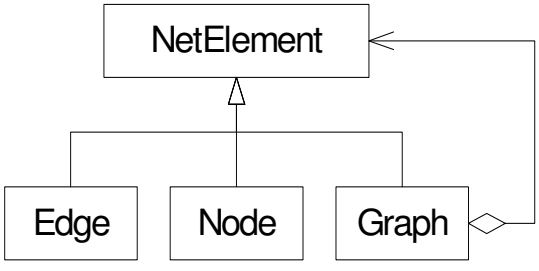
3. Die Graph-Datenstrukturen in VinetS

3.1. Allgemeines

Beim Entwurf des Frameworks wurde Wert auf Modularisierung gelegt. Die Graph-Daten werden in mehreren Schichten – in den Paketen `structure`, `geometry` und `graphics` – gespeichert. Hier und künftig sind die Klassen- und Paketnamen verkürzt wiedergegeben, alles folgende beschreibt Unterpakete von `de.fhstralsund.vinets`.

Beispielgraphen können mit dem Tool entweder interaktiv erzeugt oder aus einer XML-Datei geladen werden. Für die Speicherung wird GraphML – ein standardisiertes XML-Format für Netzwerkdaten – genutzt, vergleiche hierzu **`http://graphml.graphdrawing.org`** .

Grundsätzlich sollten alle Algorithmen nur auf den Schnittstellen arbeiten, so dass hier auf Informationen zur Implementierung der Graph-Klasse verzichtet werden kann. In der Demo-Version von VinetS können gerichtete, ungerichtete und gemischte Graphen und ihre Teile bearbeitet werden – nur die hierzu benötigten Interfaces werden im folgenden erklärt.



Über die Schnittstelle `NetElement` werden grundlegende Eigenschaften aller Bestandteile von Graphen verwaltet, wobei auch eindeutige IDs vergeben werden. Außer einem Namen kann jedes `NetElement` verschiedene Label besitzen werden – diese werden in einer `HashMap` verwaltet und lassen sich in Algorithmen zur Speicherung von Informationen nutzen. Die

wichtigsten Methoden des Interface `NetElement`, die für alle Knoten, Kanten und Graphen verfügbar sind, sind die folgenden:

`public interface de.fhstralsund.vinets.structure.NetElement`

Method Summary	
`void`	`clearLabels()` Clears all labels of this NetElement.
`NetElementGeometry`	`getGeometry()` Returns the geometry of this NetElement.
`Object`	`getID()`
`Object`	`getLabel(Object key)` Returns the label value associated with the passed key or `null`.
`int`	`getIntLabel(Object key)` Returns the integer label value associated with the passed key. (Is equivalent to `((Integer)getLabel(key)).intValue()`)
`String`	`getName()` Returns the name of this NetElement.
`NetElement-Representation`	`getRepresentation()` Returns the graphic representation of this NetElement.
`Object`	`removeLabel(Object key)` Removes a property from this NetElement.
`void`	`setGeometry(NetElementGeometry geom)` Assign a new position and/or size to this NetElement.
`void`	`setLabel(Object key, Object value)` Adds a new property to be stored as a label of this NetElement or changes the value of an existing property.
`void`	`setIntLabel(Object key, int value)` Adds or changes an integer value as a label of this NetElement. (Is equivalent to `setLabel(key, new Integer(value))`)
`void`	`setName(String name)` Sets a human readable name for this NetElement.

3.2. Die Strukturschicht

Graphalgorithmen betreffen in der Regel die Struktur des Graphen, d.h. Algorithmen verwenden vorwiegend die Methoden der drei Interfaces `Node` (für Knoten), `Edge` (für gerichtete und ungerichtete Kanten) sowie `Graph`, die alle von `NetElement` abgeleitet sind.

Kanten-Objekte verwalten Referenzen auf ihren Start- und Zielknoten, wobei die Reihenfolge der beiden Knoten bei ungerichteten Kanten zufällig ist. Es gibt die folgenden Methoden:

`public interface de.fhstralsund.vinets.structure.Edge`

Method Summary	
`Node`	`getOtherEnd(Node v)` returns the opposite end vertex of this Edge
`Node`	`getSource()`
`Node`	`getTarget()`
`boolean`	`isDirected()`
`boolean`	`isUndirected()`

Knoten-Objekte verwalten ihre inzidenten Kanten (getrennt nach herein- und herausführenden sowie ungerichteten Kanten), sie stellen Iteratoren zum Durchlaufen dieser Kantenlisten bereit.

`public interface de.fhstralsund.vinets.structure.Node`

Method Summary	
`int`	`degree()` Returns the number of undirected edges of this Node.
`java.util.Iterator`	`inArcs()`
`java.util.Iterator`	`incidentEdges()`
`int`	`indegree()` Returns the number of incoming arcs of this Node.
`java.util.Iterator`	`outArcs()`
`int`	`outdegree()` Returns the number of outgoing arcs of this Node.
`java.util.Iterator`	`undirectedEdges()`

Graph-Objekte dienen einerseits als Container für ihre Knoten und Kanten, die sie verwalten, und andererseits als Factory zum Erzeugen neuer Knoten und / oder Kanten. (Das Interface `Link` ist dabei eine Verallgemeinerung von `Edge`.)

`public interface de.fhstralsund.vinets.structure.Graph`

Method Summary	
`java.lang.Object`	`clone()` Returns a deep copy of this Graph.
`int`	`countEdges()` Returns the number of edges of this Graph.
`int`	`countNodes()` Returns the number of nodes of this Graph.
`Link`	`createEdge(Node start, Node end)` Constructs and returns a new edge of the appropriate type.
`Link`	`createEdge(Node start, Node end, boolean directed)` Constructs and returns a new directed or undirected edge.
`Node`	`createNode()` Constructs and returns a new node.
`Node`	`createNode(java.lang.String name)` Constructs and returns a new node.
`java.util.Iterator`	`edges()` Returns an Iterator to traverse all edges of this Graph.
`boolean`	`isDirected()`
`boolean`	`isMixed()` A mixed graph may contain some undirected and some directed edges.
`boolean`	`isUndirected()`
`java.util.Iterator`	`nodes()` Returns an Iterator to traverse all nodes of this Graph
`boolean`	`remove(NetElement n)` Removes a NetElement from this graph.

Das Ergebnis eines Algorithmus kann sich auf die Struktur des Graphen auswirken, dann wird dies in der Zeichnung sichtbar. Eine andere Möglichkeit besteht darin, den Namen, die Geometrie oder die graphische Ausprägung von Knoten zu verändern.

3.3. Die Graphikschicht

Hier sollen nur die für die Anzeige von Resultaten nützlichen Methoden angegeben werden, insbesondere kann die Farbe der Darstellung von Knoten und / oder Kanten geändert werden.

`public class de.fhstralsund.vinets.graphics.NetElementRepresentation`

Constructor Summary	
`NetElementRepresentation(NetElement elem)`	
Method Summary	
`java.lang.Object`	`clone()`
`boolean`	`equals(java.lang.Object o)`
`java.awt.Color`	`getColor()`
`void`	`setColor(java.awt.Color color)`

Hierbei ist zu beachten, dass von den Factory-Methoden der `Graph`-Klasse zunächst nur die Strukturschicht eines Knotens / einer Kante erzeugt wird. Soll in einem Algorithmus die Geometrie und/oder die Repräsentation für einen *neu erzeugten* Knoten **`v`** verändert werden, müssen erst die Geometrie und die Repräsentation für ihn bereitgestellt werden und zwar mit dem Aufruf **`GraphApplication.getInstance.getDeployer().deployNode(v)`**.

3.4. Die Geometrieschicht

Größe und Position eines `NetElement` werden im Koordinatensystem des Graphen angegeben, wobei stets $0 \le x \le MAX\_X$ und $0 \le y \le MAX\_Y$ gelten muss.

`public class de.fhstralsund.vinets.geometry.NetElementGeometry`

Field Summary		
`static double`	`EPS`	The precision.
`static double`	`MAX_X`	The maximum X coordinate.
`static double`	`MAX_Y`	The maximum Y coordinate.

Constructor Summary
`NetElementGeometry()` Constructs a NetElementGeometry at position (0,0) with width and height being 0 as well.
`NetElementGeometry(double x, double y)` Constructs a NetElementGeometry at the given position with width and height being 0.
`NetElementGeometry(double x, double y, double width, double height)` Constructs a NetElementGeometry with the given position and size.
`NetElementGeometry(java.awt.geom.Point2D origin)` Constructs a NetElementGeometry at the given position with width and height being 0.
`NetElementGeometry(java.awt.geom.Point2D origin, java.awt.geom.Point2D size)` Constructs a NetElementGeometry with the given position and size.

Method Summary	
java.lang.Object	clone()
static boolean	equals(double one, double two) This Method should be used to compare two double values for equality.
boolean	equals(java.lang.Object o)
double	getHeight()
java.awt.geom.Point2D	getPosition()
java.awt.geom.Dimension2D	getSize()
double	getWidth()
double	getX()
double	getY()
void	setHeight(double value)
void	setPosition(java.awt.geom.Point2D pos)
void	setSize(java.awt.geom.Dimension2D size)
void	setWidth(double value)
void	setX(double value)
void	setY(double value)
java.lang.String	toString()

Inhaltsverzeichnis: