j-Algo

The Algorithm
Visualisation Tool

Entwicklerhandbuch

Inhaltsverzeichnis

0.1 Einleitung

Dieses Handbuch soll künftigen Entwicklern von **j-Algo** helfen, sich schnell mit der Struktur der Software auseinanderzusetzen. **j-Algo** ist eine Software, die sich mit der Visualisierung von Algorithmen beschäftigt. Sie soll dazu dienen, verschiedene Algorithmen zu veranschaulichen um sie so Studenten und anderen Interessierten verständlicher zu machen. Die Anwendung basiert auf einer Plugin-Struktur, die es ermöglicht, einzelne Module, die jeweils einen Algorithmus oder ein Themengebiet abdecken können, in das Programm zu integrieren und zu laden.

Sowohl **j-Algo** als auch die einzelnen Module entstanden im Rahmen des externen Softwarepraktikums im Studiengang Informatik der TU Dresden in Zusammenarbeit mit dem Lehrstuhl Programmierung. Die implementierten Module orientieren sich daher an den Lehrveranstaltungen „Algorithmen und Datenstrukturen“ sowie „Programmierung“ im Grundstudium Informatik an der TU Dresden. Das Einsatzgebiet soll vor allem die Vorlesung und das studentische Lernen zu Hause umfassen.
j-Algo ist eine freie Software, die beliebig oft kopiert werden darf.

0.2 Technische Hinweise

0.2.1 Systemvoraussetzungen

Folgende minimale Systemanforderungen werden für den reibungslosen Einsatz von **j-Algo** benötigt:

- IBM-kompatibler PC
- Mindestens 64 MB RAM
- Windows 98(SE)/ME/2000/XP , Linux SuSE/Red Had
- Java 2 Platform Standard Edition 5.0 (siehe: http://java.sun.com/)
- Maus und Tastatur
- Monitor mit einer Auflösung von mindestens 800x600

0.2.2 Installation

Windows

Entpacken Sie nach dem Herunterladen das ZIP-komprimierte Archiv in einen Ordner Ihrer Wahl. In diesem Ordner finden Sie eine Datei namens „j-algo.bat“. Öffnen Sie diese Datei mit einem Doppelklick, und das Programm wird gestartet.

Unix

Entpacken Sie nach dem Herunterladen das TGZ-komprimierte Archiv in einen Ordner Ihrer Wahl. In diesem Ordner finden Sie eine Datei namens „j-algo.sh“. Öffnen Sie die Konsole und starten sie mittles `sh j-algo.sh` das Programm.

0.2.3 Deinstallation

Der komplette Programmordner kann jederzeit gefahrlos von der Festplatte gelöscht werden.

0.3 CVS-Zugang

j-Algo ist als Projekt bei SourceForge registriert und gehostet. Es gibt 2 Arten, auf das CVS-Repository des Projektes zuzugreifen.

1. Lesezugriff. Als Beobachter des Projektes kann jeder auf das Projekt lesend zugreifen. Die Zugangsdaten sind:
   Verbindungsmethode: `pserver`
   Host: `cvs.sourceforge.net`
   Repository-Pfad: `/cvsroot/j-algo`
   Login: `anonymous`

2. Vollzugriff. Als registrierter Entwickler bei SourceForge und als eingetragenes Projektmitglied bei **j-Algo** kann das CVS unter folgenden Zugangsdaten im Vollzugriff erreicht werden:
   Verbindungsmethode: `extssh`
   Host: `cvs.sourceforge.net`
   Repository-Pfad: `/cvsroot/j-algo`
   Login: <SourceForge-Login>
   Passwort: <SourceForge-Passwort>
   Um als Projektmitglied eingetragen zu werden, wenden Sie sich bitte an den Projekt-Administrator. Dessen Kontaktdaten sind auf der SourceForge-Seite zugänglich.

Achtung: Wird das Projekt im Rahmen des Software-Praktikums an der TUD weiterentwickelt, gelten andere Bedingungen für den CVS-Zugang. Diese sind beim zuständigen Betreuer des Praktikums zu erfragen.

0.4 Entwickeln unter Eclipse

Natürlich steht es jedem Entwickler frei, eine Programmierumgebung seiner Wahl zu benutzen. Da jedoch der Großteil der **j-Algo** -Entwickler unter Eclipse programmiert, und diese Plattform einige komfortable Features besitzt, sollen hier die wichtigsten Einstellungen für diese Umgebung erläutert werden. Für andere Programmierumgebungen gelten sie sinngemäß.
Da **j-Algo** die Java-Version 1.5 verwendet, ist eine Eclipse-Version 3.1 oder höher erforderlich.

Das Projekt kann in der CVS-Ansicht von Eclipse ausgecheckt werden. Ab jetzt sind zwar alle nötigen Daten (Quellcodes, etc.) auf dem Rechner. Allerdings müssen noch einige Einstellungen vorgenommen werden, damit das Projekt kompiliert und gestartet werden kann:

- Unter den Projekteinstellungen->Info->Text file encoding muss UTF-8 eingestellt werden. Dies garantiert reibungslose Unterstützung von Umlauten auf verschiedenen Betriebssystemen.

- Unter Projekteinstellungen->Java Build Path müssen jetzt einige Einstellungen für den ClassPath des Projektes vorgenommen werden:
  Unter Source darf nur der Ordner <PROJEKTORDNER>`/src` stehen. Andere Ordner sind zu entfernen.

- Als „Default Output Folder“ ist <PROJEKTORDNER>`/bin` anzugeben.

- Unter „Libraries“->„Add JARs...“ ist <PROJEKTORDNER>`/extlibs/jh.jar` hinzuzufügen. Dies ist die nötige Bibliothek für das Hilfe-System.

- Unter „Libraries“->„Add Class Folder...“ sind folgende Ordner hinzuzufügen:
  <PROJEKTORDNER>`/runtime` (für die Erkennung der installierten Module)
  <PROJEKTORDNER>`/res/main` (für die Ressourcen zum Hauptprogramm)
  sowie alle verfügbaren Modulordner unter <PROJEKTORDNER>`/res/modules/`, also z.B. <PROJEKTORDNER>`/res/modules/testModule` (für die Ressourcen der einzelnen Module)

- Als nächstes werden die jUnit-Bibliotheken benötigt. Weil Eclipse diese bereits eingebaut hat, ist die einfachste Variante, diese hinzuzufügen, folgendermaßen:
  Projekteinstellungen schliessen, Workspace neu kompilieren lassen, und dann im View „Problems“ einen der vielen Fehler auswählen, die im Zusammenhang mit jUnit gebracht werden. Beim Öffnen des gewählten Source-Files zeigt Eclipse im Editor bei den entsprechenden Imports Fehler an. Drücken Sie genau dort auf das rote Kreuz, und Ihnen wird die Option angeboten „Add jUnit libraries“. Wählen Sie diese aus, schließen das Source-File, und fertig.
  Jetzt sollte im View „Problems“ kein Fehler mehr angezeigt werden.

- Nun muss noch eine Startkonfiguration erstellt werden, und dann sind wir fertig:
  Unter dem Menüpunkt Run->Run... erstellen Sie eine neue Konfiguration vom Typ „Java Application“, vergeben einen sinnvollen Namen und wählen vom **j-Algo** -Projekt als „Main-Class“ `org.jalgo.main.JAlgoMain` aus.

Jetzt ist das Projekt kompilierbar und das Programm kann gestartet werden.

0.5 Projektstruktur

Es folgt ein kurzer Überblick über die bestehende Struktur des Projektes, so dass der Entwickler weiß, welche Teile er verändern darf, und welche er besser unangetastet lassen sollte ...

Das Projekt fasst mehrere Ordner und einige „lose“ Dateien. Der Reihe nach:
Der Ordner `bin` fasst die kompilierten Klassen. Sein Inhalt kann gelöscht werden, er wird bei jedem kompilieren neu erstellt. (Hinweis: Dieser Ordner gehört nicht unter die Versionskontrolle!)
Der Ordner `doc` fasst die Projektdokumentation. Dies sind die Dateien zum Entwicklerhandbuch, zum Benutzerhandbuch, sowie einige Dateien, die gewisse aufgetretene Probleme und evtl. Abhilfen schildern.
Im Ordner `examples` sind Beispieldateien für jedes Modul enthalten. Der komplette Ordner wird später in der Distribution enthalten sein.
Im Ordner `extlibs` liegen Bibliotheken, die Fremdcode enthalten. Dies ist derzeit nur die Laufzeitbibliothek des Hilfesystems. Der komplette Ordner wird später in der Distribution enthalten sein.
Der Ordner `relicts` fasst Codeteile und Ressourcendateien, welche derzeit nicht mehr verwendet werden. Sie wurden trotzdem aufgehoben, weil sie teilweise Funktionalität enthalten, die zu implementieren mal begonnen wurde, die jedoch nie ausgereift waren und daher derzeit nicht verwendet werden. Vielleicht bringen Sie einen Nutzen, wenn der Entwickler Ideen sucht.
Im Ordner `res` liegen alle Ressourcendateien geordnet nach Programmteilen.
Der Ordner `runtime` enthält leere, aber notwendige Dateien für die Laufzeit. Sie sind Teil der Pluginstruktur, und ermöglichen das Erkennen der installierten Module.
Im Ordner `src` schließlich ist der Quellcode enthalten. Die Paketstruktur ist in Abbildung X ersichtlich.
TODO: Abbildung Paketstruktur.

[12pt]article a4wide, listings, url [latin1]inputenc

J-Algo
Module Programmers Manual

Stephan Creutz Michael Pradel Alexander Claus

26. März 2006

basicstyle=,language=Java,showstringspaces=false

0.6 Introduction to jAlgo

J-Algo was developed to provide multiple module support. Each module should cover one topic (e.g. tree algorithms or EBNF). For this reason we created a simple interface. This interface is described in the sections below.
The software is using the following toolkits: SWT, JFace and Draw2d (see section **??**), thus you have to use it too. But it might be possible to write an adapter which makes it possible to use e.g. Swing or something else.

0.7 Implementing a module

To implement a new J-Algo module, create a directory with your modules name in /source/org/jalgo/module. This directory has to contain at least two classes: ModuleConnector.java implementing the interface IModuleConnector and ModuleInfo.java implementing the interface IModuleInfo.
In order to help you to write a new module, there is an minimalistic module called testModule. It is a correctly implemented J-Algo module, but does nothing. You can use it as a skeleton for any new module.

0.7.1 IModuleConnector

This interface establishes a connection to the main program. You have to implement the methods listed below:

[frame=single,caption=IModuleConnector] public interface IModuleConnector

/** * This method is invoked, after the user loaded a saved file * for the module. * * @param data the loaded file consisting of the module * header, which was added by the main program * before saving (e.g. including with which * module the file is associated) and the data * for the module; put the data in here */ public void setDataFromFile(ByteArrayInputStream data);

/** * This method is invoked, when the user wants to save the * state of the module. * * @return a stream with the data from the module, that has * to be stored in a file after the main program * added the module header (e.g. including with * which module the file is associated) to it */ public ByteArrayOutputStream getDataForFile();

/** * This method will be invoked, if the user clicked the * print-button (or chose to print in any other way) * The module will call a print dialog and manage the * printing. * * NOTE! Printing is currently not supported by the J-Algo * main program. */ public void print();

/** * Get the Menu from the module */ public SubMenuManager getMenuManager();

/** * Get the ToolBar from the module */ public SubToolBarManager getToolBarManager();

/** * Get the StatusLine from the module */ public SubStatusLineManager getStatusLineManager();

/** * Get a class with all module information (name, * description, version, ...) */ public IModuleInfo getModuleInfo();

/** * This method is invoked, when module or program are * intended to be closed. * Here the user can be asked for saving his work. * If this method returns false, the closing of module/ * program is ignored. * * @return true, if module is ready to be closed, * false otherwise */ public boolean close();

0.7.2 ModuleConnector from testModule

Here is a concrete example for a class implementing the *IModuleConnector* interface.

[frame=single,caption=IModuleInfo interface] public class ModuleConnector implements IModuleConnector

private ModuleInfo moduleInfo;

private ApplicationWindow appWin; private Composite comp; private SubMenuManager menuManager; private SubToolBarManager toolBarManager; private SubStatusLineManager statusLineManager;

public ModuleConnector(ApplicationWindow appWin, Composite comp, SubMenuManager menu, SubToolBarManager tb, SubStatusLineManager sl)

moduleInfo = new ModuleInfo();

this.appWin = appWin; this.comp = comp; this.menuManager = menu; this.toolBarManager = tb; this.statusLineManager = sl;

public void run() System.err.println("testModule is running");

public void setDataFromFile(ByteArrayInputStream data)

public ByteArrayOutputStream getDataForFile() return null;

public void print()

public SubMenuManager getMenuManager() return menuManager;

public SubToolBarManager getToolBarManager() return toolBarManager;

public SubStatusLineManager getStatusLineManager() return statusLineManager;

public IModuleInfo getModuleInfo() return moduleInfo;

public boolean close() return true;

0.7.3 IModuleInfo

The class which implements this interface provides some basic information about the module. The class yields the name, version, author(s), license, description and a logo. Furthermore it holds the information about open files.

[frame=single,caption=IModuleInfo interface] import org.eclipse.jface.resource.ImageDescriptor

public interface IModuleInfo public String getName(); public String getVersion(); public String getAuthor(); public String getDescription(); public ImageDescriptor getLogo(); public String getLicense();

/** * Get the filename of the currently opened file. * @return filename */ public String getOpenFileName(); /** * Set the filename of the currently opened file. * @param string filename */ public void setOpenFileName(String string);

0.7.4 ModuleInfo from testModule

Here is a concrete example for a class implementing the $IModuleInfo$ interface.

[frame=single,caption=Sample ModuleInfo Code] public class ModuleInfo implements IModuleInfo

public String getName() return "testModule";
public String getVersion() return "0.1";
public String getAuthor() return "Your Name";
public String getDescription() return ä module for testing purposes";
public ImageDescriptor getLogo() return ImageDescriptor.createFromFile(null, "pix/new.gif");

public String getLicense() return "GPL";
public String getOpenFileName() return null;
public void setOpenFileName(String string)

If this is too easy for you, just have a look at the existing modules, namely ÄVL-trees", "Dijkstra-algorithm", SSyntax diagrams and EBNF".

0.8 Bind the module to the main program

Binding a new module to the main program is really easy: You only have to add two lines into the class /src/org/jalgo/main/JalgoMain.java.
You should find the method addKnownModules: [frame=single,caption=JalgoMain.addKnownModules()] public void addKnownModules() try knownModules.add(ModuleConnector.class); knownModules.add(org.jalgo.module.testModule.ModuleConnector.class); //Add a new ModuleConnector here!! catch (Exception e) e.printStackTrace(); knownModuleInfos.add(new ModuleInfo()); knownModuleInfos.add(new org.jalgo.module.testModule.ModuleInfo()); //Add a new ModuleInfo here!!

Please use the complete path to your classes!

0.9 Known bugs

- The method *publicvoidprint*() is unused and senseless in the moment.

0.10 Reporting bugs

If you find a bug please be so kind to drop us an email (j-algo-development@lists.sourceforge.net).

0.11 See also

To program a module you need information about SWT, JFace and Draw2d. This can be found at http://www.eclipse.org.
More detailed information can be found in the source code.