GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Forschungsbezogenes Praktikum

im Studiengang "Angewandte Informatik"

Entwicklung eines Plugins für die 3D-Modellierungsplatform GroIMP

Dominick Leppich

Institut für Informatik

12. Juli 2017

Georg-August-Universität Göttingen
Institut für Informatik

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌍 www.informatik.uni-goettingen.de

Betreuer: Prof. Dr. Winfried Kurth

Inhaltsverzeichnis

Einleitung

In diesem Praktikum beschäftige ich mich mit dem Entwickeln eines Plugins für die Software GroIMP. Da hierzu sehr wenig Dokumentation vorhanden ist, erkläre ich in diesem Praktikumsbericht alle Schritte die notwendig sind, um GroIMP auf dem Arbeitssystem zu installieren, in der Entwicklungsumgebung *Eclipse* [1] einzurichten und ein Plugin von Grund auf neu zu schreiben. Weiterhin gehe ich in einigen Beispielen darauf ein, wie das Plugin in die bestehende GroIMP Architektur eingefügt werden kann.

GroIMP ist eine 3D-Modellierungsplatform, mit welcher auch das Wachstum von Pflanzen und anderen wachsenden Strukturen simuliert und visualisiert werden kann. Die Regeln für das Wachstum werden in sogenannten Wachstumsgrammatiken [2] definiert, welche sich aus Lindenmayer-Systemen [3] entwickelt haben.

Zur Realisierung dieser Wachstumsgrammatiken wurde die Programmiersprache `XL` entwickelt. Zusätzlich können in `XL` Java Befehle direkt ausgeführt werden [4]. Auf diese Weise ist es möglich beliebige Java Bibliotheken für eine Wachstumssimulation zu verwenden, ohne sich auf eine vordefinierte Menge an Hilfsmethoden festlegen zu müssen.

Sowohl innerhalb von GroIMP als auch auf der Internetpräsenz sind viele Beispielprojekte zu finden. Darunter befindet sich auch das Beispielprojekt Drachenkurve[1], mit welcher die Kurve nach beliebig vielen Wachstumsschritten simuliert werden kann.

„Die Drachenkurve ist ein fraktales Objekt, das ähnlich wie die Koch-Kurve und die Hilbert-Kurve durch Ersetzung erzeugt wird.“ [5]

In Abbildung 1.1 ist eine Drachenkurve nach 9 Simulationsschritten innerhalb der GroIMP Software gezeigt.

[1] `http://wwwuser.gwdg.de/~groimp/grogra.de/gallery/AlgorithmicBeauty/dragoncurve.html`

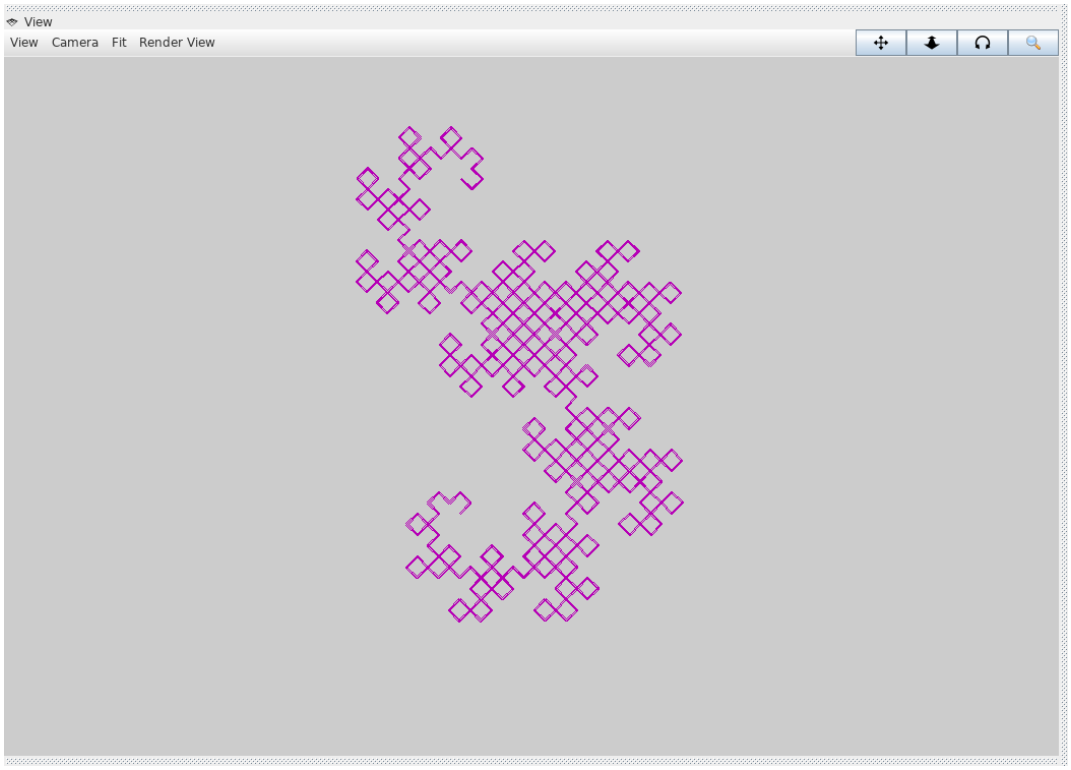

Abbildung 1.1: GroIMP Simulation der Drachenkurve nach 9 Iterationsschritten

Eclipse-Setup

In diesem Kapitel wird die Einrichtung von GroIMP in Eclipse erklärt. Hierzu muss zuerst eine aktuelle Version von GroIMP heruntergeladen werden, welche dann im Anschluss als Projekt in Eclipse importiert werden kann. Zuletzt wird erklärt, wie GroIMP aus Eclipse heraus gestartet werden kann.

2.1 Download von GroIMP

Auf der Sourceforge-Seite[1] von GroIMP kann die aktuellste Version von GroIMP heruntergeladen werden.

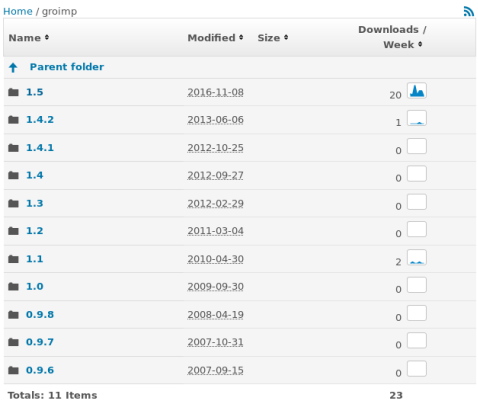

Abbildung 2.1: Auflistung der aktuellen GroIMP Versionen

Auf dieser Liste wird dann die aktuellste Version ausgewählt und auf der nächsten Seite das Archiv `GroIMP-X.X-src.zip` heruntergeladen. Dieses Archiv enthält den gesamten Quellcode von GroIMP und muss für den späteren Import in Eclipse an einen beliebigen Ort entpackt werden.

[1] `https://sourceforge.net/projects/groimp/files/groimp/`

2.2 Plugins

GroIMP verwendet die Programmiersprache Java und ist damit bereits von Haus aus eine platformunabhängige Software. Das Programm besteht aus vielen einzelnen Plugins, die zusammen die Modellierungsplatform GroIMP bilden. Jedes dieser Plugins liegt in einem eigenen Ordner und ist selbst ein eigenständiges Projekt, welches als solches in Eclipse importiert werden kann.

Einige dieser Plugins sind zwingend erforderlich um die Anwendung starten zu können, andere hingegen sind Erweiterungen die zusätzliche Funktionalität bereitstellen.

Diese Architektur erlaubt es bei korrekter Verwendung die einzelnen Plugins recht isoliert voneinander zu entwickeln und damit die Wartbarkeit zu erhöhen. Auch wäre es möglich unerwünschte Erweiterungen einfach zu deaktivieren, indem die Plugins aus dem GroIMP Verzeichnis entfernt werden, wobei auf Abhängigkeiten geachtet werden muss. Neue Funktionen können der Software einfach als Plugins hinzugefügt werden.

Das *Hauptplugin* von GroIMP trägt den Namen `Project-Core`. Dieses Plugin enthält die ausführbare Klasse `Main` im Package `de.grogra.pf.boot` und wird zum Starten von GroIMP ausgeführt. Während des Startvorgangs werden vorhandene Plugins automatisch gefunden und geladen.

2.3 Import in Eclipse

Am einfachsten ist der Import aller Plugins in Eclipse, wobei es theoretisch ausreichen würde nur das Hauptplugin, das zu schreibende Plugin und alle dafür benötigten Plugins zu importieren.

Hierfür ist es sinnvoll einen neuen Workspace anzulegen, um nicht mit GroIMP-fremden Projekten Verwirrung zu stiften. Dies geschieht entweder beim Start von Eclipse oder nachträglich manuell über `File` → `Switch workspace`. Nachdem in einen anderen Workspace gewechselt wurde, ist die Liste der Projekte leer.

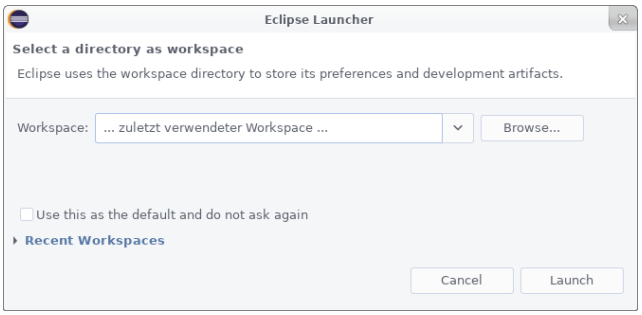

Abbildung 2.2: Einstellen des Workspace beim Starten von Eclipse

Über `File` → `Open Projects from File System...` können jetzt alle GroIMP Projekte importiert werden. Dazu wechselt man über `Directory...` ins Wurzelverzeichnis aller Plugins (dies sollte das entpackte Quellcode-Archiv von GroIMP sein) und bestätigt seine Auswahl mit `OK`. Wenn alles korrekt geklappt hat, zeigt Eclipse eine Liste von gefundenen Java-Projekten in einer Liste im gleichen Dialogfenster an. In dieser Liste müssen nun alle Einträge ausgewählt und der Dialog anschließend über `Finish` bestätigt werden.

Abbildung 2.3: Auszug der zu importierenden GroIMP Projekte in Eclipse

Wenn der Import erfolgreich war sind jetzt alle Plugins von GroIMP als Java-Projekte im `Package-Explorer` von Eclipse aufgelistet.

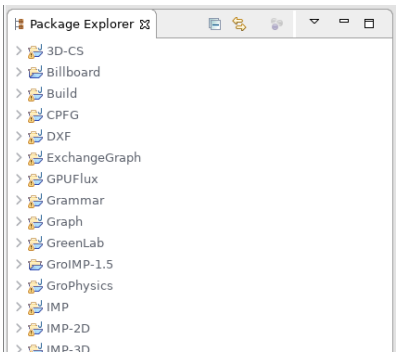

Abbildung 2.4: Auszug der Projekte im Package-Explorer in Eclipse

2.4 Starten von GroIMP

Um zu prüfen ob alles korrekt funktioniert kann GroIMP jetzt gestartet werden. Hierzu muss das Hauptplugin `Platform-Core` innerhalb von Eclipse ausgeführt werden. In diesem Projekt liegt im Ordner `src` das Package `de.grogra.pf.boot` mit der ausführbaren Klasse `Main`, welche geöffnet werden muss.

Jetzt kann GroIMP über das Symbol mit dem grünen Pfeil oder über das Menü unter `Run` → `Run` gestartet werden. Ein gewöhnliches Starten schlägt allerdings mit folgender Meldung fehl, weil GroIMP davon ausgeht, dass es bereits korrekt kompiliert wurde und sich alle anderen Plugins im kompilierten Zustand am richtigen Ort befinden.

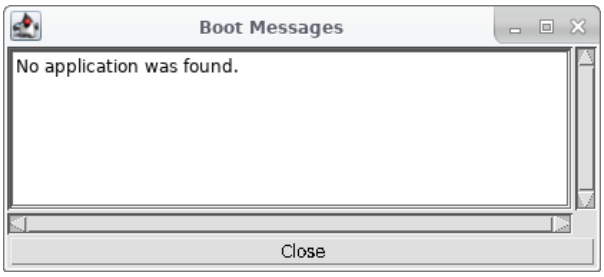

Abbildung 2.5: Fehlermeldung beim *normalen* Starten von GroIMP aus Eclipse

Während der Entwicklung mit Eclipse ist die Ordnerstruktur jedoch eine andere. Die Klasse `Main` kann hierfür mit einem speziellen Schalter `--project-tree` gestartet werden. Das bewirkt, dass die Klasse nicht versucht die Plugins aus kompilierten Archiven zu starten sondern die Plugins in der Ordnerstruktur sucht, wie sie bei der Eclipse-Entwicklung vorliegt.

Hierzu muss über das Menü `Run` → `Run Configurations` rechts im Reiter `Arguments` die Zeichenfolge `--project-tree` unter `Program arguments:` eingetragen werden. Mit einem Klick auf `Apply` wird diese Änderung bestätigt und GroIMP kann nun problemlos gestartet werden.

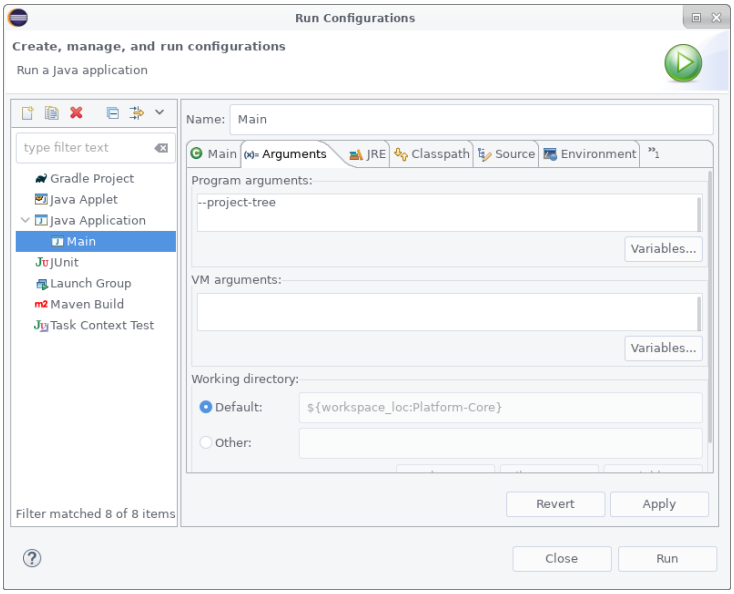

Abbildung 2.6: Einstellen der Startparameter von GroIMP

Plugin Architektur

Dieses Kapitel erläutert die Plugin Architektur in GroIMP. Es wird erklärt wie die Dateien eines Plugins aufgebaut sind und wie man ein neues Plugin erstellen kann, dies geschieht exemplarisch mit einem neuen Plugin namens `MyPlugin`. Dieses Plugin wird dann in Eclipse importiert und es werden Beispiele für die Anbindung des eigenen Plugins an GroIMP gezeigt.

3.1 Dateistruktur

Damit GroIMP das eigene Plugin beim Starten korrekt laden kann, müssen einige Richtlinien eingehalten werden.

Jedes Plugin liegt in einem eigenen Verzeichnis an gleicher Stelle, an der auch die übrigen Plugins liegen (nicht der `plugin` Ordner, sondern das entpackte Quellcode Verzeichnis!).

Die Ordnerstruktur eines Plugins folgt dabei diesem Schema:

- MyPlugin
  - `build.xml`
  - `src`
    - `plugin.xml`
    - `plugin.properties`
    - Java Quellcode Dateien
    - Weitere Dateien (zum Beispiel Grafiken)
  - `build`
    - Kopie der `plugin.xml`
    - Kopie der `plugin.properties`
    - Kompilierte Java Klassen
    - Kopierte Weitere Dateien aus `src`
  - `lib`
    - Benötigte Bibliotheken als `jar`-Dateien

Abbildung 3.1: Dateistruktur des Plugins

3.2 Buildfile

Um das Plugin später mit einer neuen Version von GroIMP verteilen zu können, muss eine `build.xml`-Datei für das Programm *Ant* erzeugt werden, welches dann automatisiert eine Release Version von GroIMP erzeugen kann. Für den Import des neuen Plugins in Eclipse ist diese Datei nötig.

```
<?xml version = "1.0" encoding="UTF-8"?>
<project name="MyPlugin" default="compile">
  <description>
    Ant build file for MyPlugin
  </description>

  <import file="../Build/buildproject.xml"/>

  <target name="-projects">
    <antcall target="-addproject">
      <param name="proj" value="MyPlugin"/>
    </antcall>
  </target>
</project>
```

Listing 3.1: Minimale `build.xml` Datei

Für eine Veröffentlichung des Plugins ist ein detaillierteres Buildfile nötig, wenn weitere Abhängigkeiten zu anderen Plugins bestehen, was aber bei der Entwicklung zunächst nicht notwendig ist und deshalb an dieser Stelle nicht weiter erläutert wird. Informationen hierzu sind im SourceForge Wiki Eintrag [6] über die Erstellung eines eigenen Plugins nachzulesen.

3.3 plugin.xml

Auf jeden Fall jedoch muss die `plugin.xml` Datei angelegt werden, welche GroIMP alle wichtigen Informationen über das Plugin liefert und welche beim Start von GroIMP automatisch gelesen wird, um dieses Plugin zu laden.

3.3.1 Beispiel

```
<?xml version = "1.0" encoding="UTF-8"?>
<plugin
  id="de.grogra.myplugin"
  version="1.0"
  xmlns="http://grogra.de/registry">

  <library file="mylib.jar" prefixes="{bar.lib.my,foo.lib.my}"/>
  <library file="MyPlugin.jar" prefixes="{de.grogra.myplugin}"/>

  <import plugin="de.grogra.pf"/>
  <import plugin="de.grogra.pf.ui.swing"/>

  <registry>
  </registry>
</plugin>
```

Listing 3.2: Eine einfache `plugin.xml` Datei

Sowohl die `id` als auch die `version` können frei gewählt werden. Bei der `id` wird jedoch empfohlen den Namenskonventionen von Java zu folgen. Der XML Namespace `xmlns="http://grogra.de/registry"` muss so wie im Beispiel übernommen werden.

3.3.2 Externe Bibliotheken

Es müssen alle externen Bibliotheken angegeben werden. Beispielhaft wurde hier eine fiktive Bibliothek `mylib.jar` eingefügt, die jedoch im weiteren Verlauf nicht mehr verwendet wird. Bibliotheken, die andere Bibliotheken benötigen, müssen hinter diesen stehen.

Mit dem Attribut `file` wird angegeben, wie die Jar-Datei der Bibliothek heißt. Externe Bibliotheken liegen im Verzeichnis `lib` innerhalb des Plugins (siehe Abbildung 3.1).

Das Attribut `prefixes` ist eine mit Kommas getrennte Liste aller Java-Packages, die aus diesen Bibliotheken verwendet werden.

Das eigene Plugin muss hierbei als letzter Eintrag auch aufgeführt werden und den selben Namen haben wie das Plugin, in diesem Fall also `MyPlugin.jar`.

3.3.3 Verwendete GroIMP Plugins

Als nächstes müssen alle GroIMP Plugins importiert werden, die für die Ausführung des eigenes Plugins benötigt werden. In diesem Beispiel werden die Plugins `de.grogra.pf` und `de.grogra.pf.ui.swing` verwendet, um eine einfache Kommunikation mit GroIMP zu ermöglichen.

Der Name des zu importierenden Plugins entspricht hierbei der `id`, die in der jeweiligen `plugin.xml` Datei definiert ist.

3.3.4 Registry

Im `registry`-Element kann die GroIMP Registry modifiziert werden. Es können Einträge eingefügt oder bereits bestehende Einträge verändert werden.

Leider sind Umgang und Aufbau der Registry weder dokumentiert noch sind im SourceForge-Wiki [6] Hinweise dazu verfügbar. Um mögliche Beispiele zu finden kann man sich an den `plugin.xml` Dateien der existierenden Plugins orientieren und versuchen diese zu adaptieren.

Die Registry von GroIMP ist baumartig aufgebaut und kann sowohl beim Start während des Ladevorgangs der Plugins als auch zur Laufzeit mit Inhalten gefüllt werden. Beim Start jedes Plugins werden die Inhalte des `registry` Elements der `plugin.xml` in die GroIMP Registry eingefügt [7].

Die Wurzel der Registry wird mit `/` angesprochen. Jedes Element im Baum kann beliebig viele Kindelemente haben. Ein Kindelement `foo`, welches direkt an der Wurzel eingefügt wird, wird dann mit `/foo` angesprochen.

Mit `ref` Elementen innerhalb des `registry` Elements wird angegeben, an welcher Stelle in der Registry Modifikationen ausgeführt werden sollen. Für jede Ebene in der Registry wird ein eigenes geschachteltes `ref` Element verwendet, der Name der Kindelements im Baum wird mit dem Attribut `name` festgelegt. Soll beispielsweise in der Registry unter `/foo/bar` eine Änderung stattfinden, wird dies durch geschachtelte `ref` Elemente erreicht.

```
<registry>
  <ref name="foo">
    <ref name="bar">
      ...
    </ref>
  </ref>
</registry>
```

Listing 3.3: Registry Zugriff bei `/foo/bar`

In einem laufenden GroIMP Projekt kann eine XL Konsole geöffnet werden, indem im Menü `Panels→RGG Panels` auf `XL Console` geklickt wird. In dieser Konsole kann dann mittels

```
de.grogra.util.Utils.dumpTree(workbench().getRegistry())
```

die aktuelle Projekt-Registry auf der Konsole (nicht der GroIMP Konsole!) ausgegeben werden (in Abbildung 3.2 wird eine Ausgabe der Eclipse Konsole gezeigt). Die gesamte GroIMP Registry kann mithilfe des folgenden Befehls angezeigt werden:

```
de.grogra.util.Utils.dumpTree(workbench().getRegistry().getParentRegistry())
```

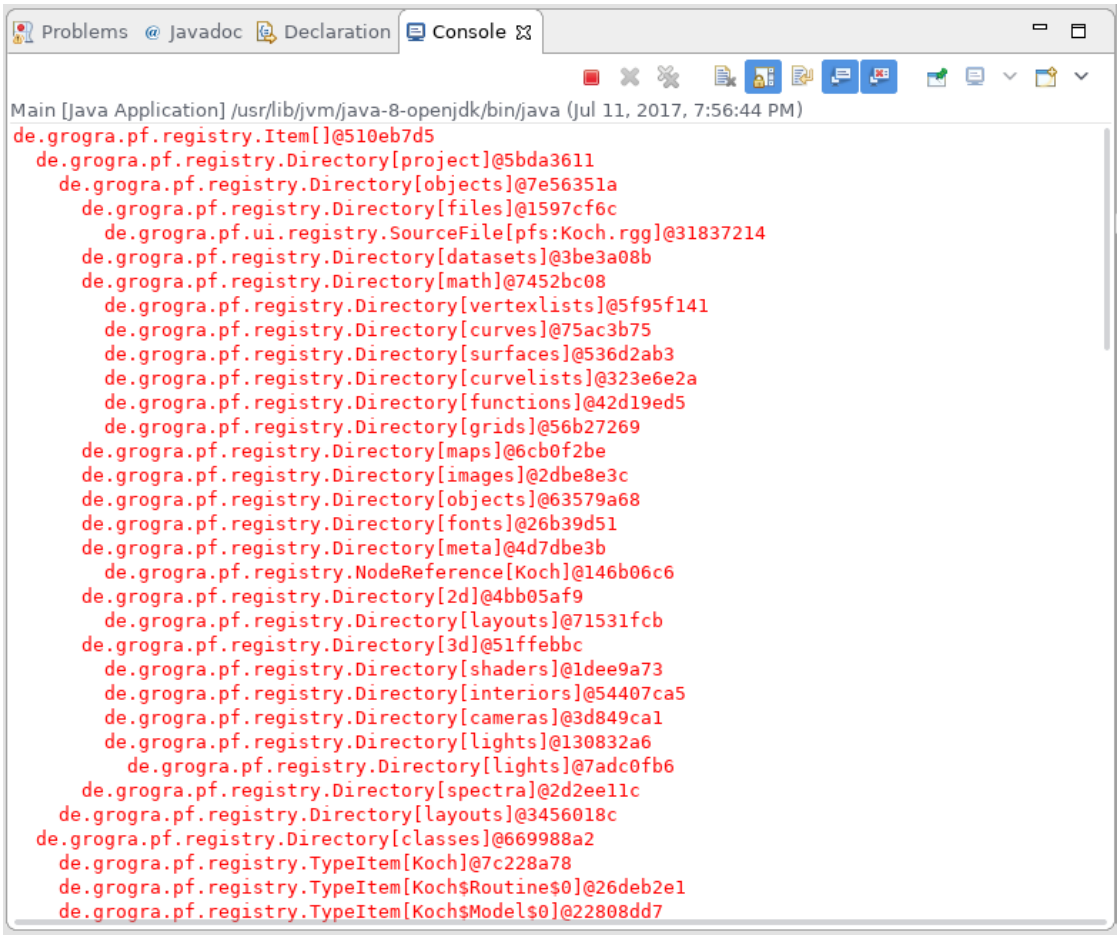

Abbildung 3.2: Eclipse Konsolenausgabe der Projekt-Registry des GroIMP Beispielprojektes `Koch`

Am Beispiel von `MyPlugin` werden hier kurz zwei Möglichkeiten der Integration in GroIMP demonstriert.

3.3.5 Beispiel - Menüeintrag zum Aufrufen einer statischen Methode

Es soll für dieses Beispiel eine einfache statische Methode `myTest` in unserem Plugin aufgerufen werden.

```
<?xml version = "1.0" encoding="UTF-8"?>
<plugin
  ...

  <registry>
    <ref name="ui">
      <ref name="commands">
        <command name="mytest" run="de.grogra.myplugin.MyPlugin.myTest"/>
      </ref>
    </ref>
    <ref name="hooks">
      <ref name="complete">
        <ref name="project">
          <insert target="/workbench/menu/src/file/save" resolveLinks="false">
            <link source="/ui/commands/mytest"/>
          </insert>
        </ref>
      </ref>
    </ref>
  </registry>
</plugin>
```

Listing 3.4: Einfügen eines Menüeintrages für die statische Methode `myTest`

Es muss zunächst ein `Command` Element erzeugt werden, um die Aktion, die beim Klicken auf den Menüeintrag ausgeführt werden soll, zu definieren. In diesem Beispiel wird ein `Command` namens `mytest` erzeugt, welches die Methode `de.grogra.myplugin.MyPlugin.myTest` aufruft.

Jetzt muss dieses `Command` noch mit einem Menüeintrag in der Registry verknüpft werden. Unter dem Pfad `/hooks/complete/project` wird dazu ein neuer Eintrag im Menü mit dem `insert` Element erzeugt, der beim Anklicken des Menüeintrags das zuvor angelegte `Command` unter `/ui/commands/mytest` ausführen soll.

3.3.6 Beispiel - Menüeintrag zum Erzeugen eines GroIMP Panels

Im zweiten Beispiel soll über einen Menüeintrag ein eigenes Panel in GroIMP erzeugt werden.

```
<?xml version = "1.0" encoding="UTF-8"?>
<plugin
  ...

  <registry>
    <ref name="ui">
      <ref name="panels">
        <panel name="mypanel">
          <exists name=".available" ref="/project"/>
          <object expr="de.grogra.myplugin.MyPlugin.createPanel">
            <var name="context"/>
            <vars/>
          </object>
        </panel>
      </ref>
    </ref>
  </registry>
</plugin>
```

Listing 3.5: Einfügen eines Menüeintrages für das Erzeugen eines eigenen Panels `MyPanel`

Das Panel soll im Menü unter `Panels` aufgelistet werden und muss daher in der Registry unter `/ui/panels` mit einem Namen (in diesem Fall `mypanel`) abgelegt werden.

Zum Erzeugen des Panels wird die Methode `de.grogra.myplugin.MyPlugin.createPanel` ausgeführt, welche dafür verantwortlich ist ein eigenes Panel zu instantiieren.

Damit dieser Menüeintrag nur bedingt angezeigt wird, kann mit dem `exists` Element eine Bedingung an das Anzeigen des Panel Eintrags gebunden werden.

Zum Erzeugen des Panels wird in GroIMP weiterhin der `Context` des Projektes benötigt (um darüber zum Beispiel Zugriff auf die aktuelle `Workbench` zu erhalten).

Der weitere Eintrag `<vars/>` wird auch zwingend benötigt, kann von mir aber nicht genau erklärt werden.

3.4 plugin.properties

Für die Namensgebung der Menüeinträge und generell der Definition von Benennungen ist die Datei `plugin.properties` verantwortlich.

```
pluginName = MyPlugin
provider = grogra.de

/ui/panels/mypanel.Name = Create MyPanel
/ui/panels/mypanel.Icon = `icon filesystems/folder`

/ui/commands/mytest.Name = MyTest
```

Listing 3.6: Eine einfache `plugin.properties` Datei

Mit `pluginName` wird der Name des Plugins festgelegt, unter dem das Plugin in GroIMP (zum Beispiel beim Laden) angezeigt wird.

Weiterhin werden die Menüeinträge, die zuvor in den Beispielen in der Registry eingefügt wurden, mit einem Namen versehen. Die Elemente werden über den Pfad in der Registry angesprochen und haben ein Attribut `Name`, welches mit = gesetzt werden kann.

Weiterhin können Menüeinträgen über das Attribut `Icon` kleine Icons hinzugefügt werden.

Verfügbare Icons liegen im `build` Verzeichnis des Plugins `Platform` im Unterordner `icons`.

3.5 Import in Eclipse

Das neu angelegte Plugin kann wie zuvor in Abschnitt 2.3 importiert werden. Damit der Quellcode innerhalb von Eclipse kompiliert werden kann, müssen noch die Projektabhängigkeiten korrekt gesetzt werden.

In Eclipse wird hierzu über einen Rechtsklick auf das importierte `MyPlugin` Projekt das Einstellungsfenster mit einem Klick auf `Properties` geöffnet (siehe Abbildung 3.3).

In diesem müssen unter `Java Build Path` im Reiter `Projects` die benötigten Projekte eingetragen werden. Über einen Klick auf `Add...` öffnet sich ein Auswahlfenster, in welchem die Projekte `Platform` und `Platform-Swing` ausgewählt werden müssen (siehe Abbildung 3.4).

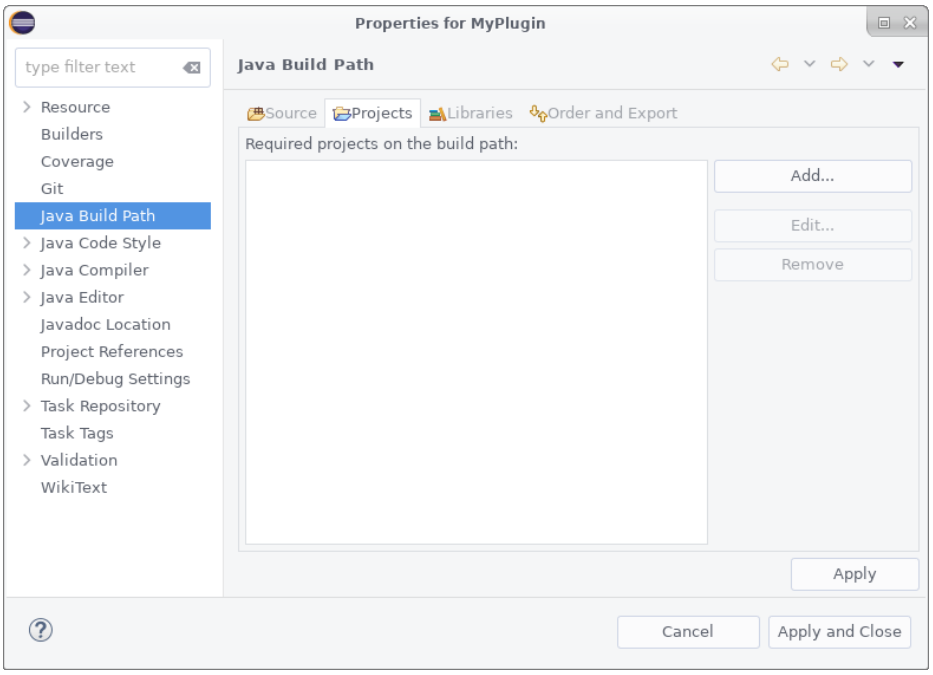

Abbildung 3.3: Properties des eigenen `MyPlugin` Plugins

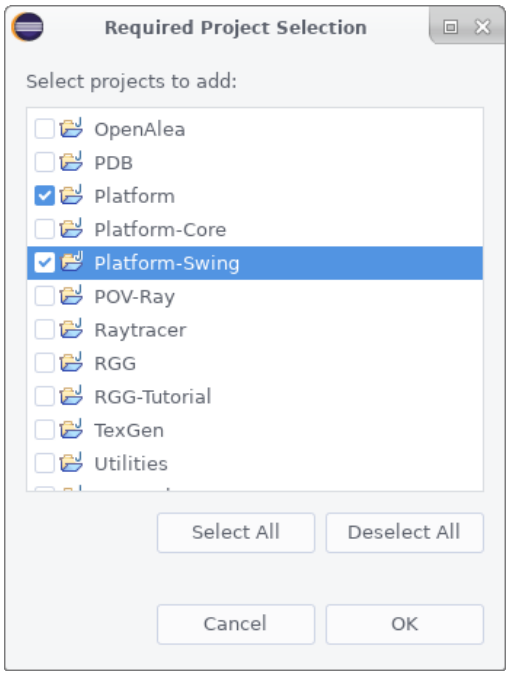

Abbildung 3.4: Properties des eigenen `MyPlugin` Plugins

Falls `Java Build Path` nicht angezeigt wird, hat Eclipse das Projekt nicht als Java Projekt importiert (was passieren kann, falls im `src` Verzeichnis noch keine Quellcode Dateien liegen oder die `build.xml` Datei fehlt).

Um das zu beheben kann das Projekt aus Eclipse entfernt werden (ohne die Dateien auf der Festplatte zu löschen!). Danach müssen die `build.xml` und Quellcode Dateien erstellt und im Plugin Ordner die versteckte Datei `.project` Datei von Eclipse gelöscht werden.

3.6 Java Implementation

Jetzt müssen noch Methoden im Quellcode des Plugins implementiert werden.

Hierfür wird im `src` Verzeichnis des Packages `de.grogra.myplugin` eine Java Datei `MyPlugin.java` erstellt. Für das zweite Beispiel wird weiterhin eine Datei `MyPanel.java` angelegt.

3.6.1 Beispiel - Menüeintrag zum Aufrufen einer statischen Methode

Um eine einfache statische Methode (wie in Beispiel 3.3.5) auszuführen, kann folgender Quellcode verwendet werden.

```
package de.grogra.myplugin;

import de.grogra.pf.registry.Item;
import de.grogra.pf.ui.Context;

public class MyPlugin {
        public static void myTest(Item item, Object info, Context ctx) {
                System.out.println("myTest");
        }
}
```

Listing 3.7: Statische Methode `myTest` in `MyPlugin.java`

Der Rückgabewert und die Parameter der Methode müssen wie im Beispiel übernommen werden, damit die Methode aus GroIMP über den Menüeintrag so aufgerufen werden kann.

Hier diente die statische Methode `exportCompiled` des Plugins `RGG` aus der Klasse `de.grogra.rgg.model.CompilationFilter` als Vorlage [7].

3.6.2 Beispiel - Menüeintrag zum Erzeugen eines GroIMP Panels

Auch zum Erzeugen eines Panels muss zunächst eine statische Methode angelegt werden, welches dieses Panel instantiiert (siehe Beispiel 3.3.6). Es gibt mehrere Möglichkeiten eine Instanz eines Panels zu erzeugen.

In diesem Beispiel wurde eine Klasse `MyPanel` erzeugt, die von `PanelSupport` aus dem Package `de.grogra.pf.ui.swing` im Plugin `Platform-Swing` erbt.

Diese Basisklasse bietet automatisch alle Funktionalitäten für GroIMP Panel, wie zum Beispiel das automatische Andocken an GroIMP Komponenten innerhalb der grafischen Oberfläche.

Eine statische Methode `createPanel` innerhalb der Klasse `MyPlugin` erzeugt dann ein Objekt dieses Panels. Diese statische Methode wurde zuvor in der Registry an einen Menüeintrag gekoppelt.

```
package de.grogra.myplugin;

import de.grogra.pf.ui.*;
import de.grogra.pf.ui.swing.WindowSupport;
import de.grogra.util.Map;

public class MyPlugin {
        public static Panel createPanel(Context ctx, Map params) {
                MyPanel panel = (MyPanel) new MyPanel(ctx.getWorkbench()).initialize((
                    WindowSupport) ctx.getWindow(), params);
                return panel;
        }
}
```

Listing 3.8: Statische Methode `createPanel` zum Erzeugen einer Instanz von `MyPanel`

Das Panel selbst muss innerhalb des Konstruktors den `super`-Konstruktor der Klasse `PanelSupport` mit einem `SwingPanel` Argument aufrufen.

Testweise enthält das Panel einfach ein Swing Label mit Beschriftung `MyLabel`. Dieses kann dem Panel nicht wie von Swing gewohnt mittels `add` hinzugefügt werden. Die Basisklasse `PanelSupport` bietet hier die Möglichkeit mittels `getComponent().getContentPane()` auf das zugehörige Swing Panel zuzugreifen.

```
package de.grogra.myplugin;
```

```
import javax.swing.JLabel;

import de.grogra.pf.ui.Workbench;
import de.grogra.pf.ui.swing.*;

public class MyPanel extends PanelSupport {
        public MyPanel(Workbench workbench) {
                super(new SwingPanel(null));

                JLabel testLabel = new JLabel("MyLabel");

                ((SwingPanel) getComponent()).getContentPane().add(testLabel);
        }
}
```

Listing 3.9: Konstruktor von `MyPanel`

Literaturverzeichnis

[1] "Eclipse Website," https://eclipse.org/, letzter Zugriff: 12. Juli 2017.

[2] "GroIMP Website," http://www.grogra.de/software/groimp, letzter Zugriff: 8. Juli 2017.

[3] "Wikipedia: Lindenmayer-Systeme," https://de.wikipedia.org/wiki/Lindenmayer-System, letzter Zugriff: 12. Juli 2017.

[4] "Grogra Website," http://www.grogra.de, letzter Zugriff: 8. Juli 2017.

[5] "Wikipedia: Drachenkurve," https://de.wikipedia.org/wiki/Drachenkurve, letzter Zugriff: 12. Juli 2017.

[6] "Erzeugen eines eigenen Plugins," https://sourceforge.net/p/groimp/wiki/Creating%20an%20own%20plugin/, letzter Zugriff: 8. Juli 2017.

[7] "Email Verkehr mit Ole Kniemeyer (Entwickler)."