



Whitepaper zur Literaturverwaltung

Liabolo

„Distribution und Installation von Liabolo“
von Thorsten Schlörmann

erstellt im Rahmen der Projektgruppe
„Virtuelle Organisation - Open Source“

unter der Leitung von
Prof. Uwe Schneidewind und Hendrik Eggers (Lehrbeauftragter)

Inhaltsverzeichnis

1	Einleitung	1
2	Plattformspezifische Verteilung und Installation	1
2.1	Windows	1
2.2	Linux	3
2.2.1	Red Hat Package Manager	3
2.2.2	Debian Paketverwaltung	4
3	Plattformunabhängige Verteilung und Installation	6
3.1	Quellcode als Zip-Datei oder tarball	6
3.2	make und ant	7
3.2.1	make	7
3.2.2	ant	8
3.3	Java Webstart	9
4	Programmstruktur	9
4.1	Installationsort	10
4.2	Aufbau des Installationsverzeichnisses	10
4.3	Lage der Userdaten	10
5	Schluss	11

1 Einleitung

In der Entwicklungsphase eines Softwareprojekts gelangt man irgendwann an den Punkt, wo eine erste lauffähige Version des Programms vor der Veröffentlichung steht. Dann stellen sich häufig Fragen, über die im vorherigen Entwicklungszeitraum wenig nachgedacht wurde, nämlich wie das Programm zum Benutzer gelangen und in welcher Form dies geschehen kann. Auch bei der Installation der Software müssen nicht nur die Besonderheiten und Möglichkeiten des Zielbetriebssystems, sondern auch die unterschiedlichen Kenntnisse der Benutzer im Umgang mit Computern berücksichtigt werden und nach Mitteln gesucht werden, die Installation für den Benutzer zu erleichtern. Im Folgenden sollen hier mögliche Vorgehensweisen bei der Distribution und Installation von Liabolo vorgestellt werden.

2 Plattformspezifische Verteilung und Installation

2.1 Windows

Benutzer des Betriebssystems Windows sind es im Allgemeinen gewöhnt, Software mittels eines Installationsprogramms auf dem System zu installieren. Solch ein Hilfsprogramm, oft auch „Wizard“ genannt, führt den Benutzer Schritt für Schritt durch die Installation, indem die benötigten Informationen abgefragt werden, kleine Hilfstexte gegeben werden oder bereits voreingestellte sinnvolle Angaben vorgeschlagen werden, welche der Benutzer einfach übernehmen oder nach seinen Vorstellungen ändern kann. Danach kopiert es die benötigten Dateien auf das System und nimmt evtl. notwendige Einträge in die Windows-Registry vor, mit dessen Hilfe System und Userdaten verwaltet werden.

Java Programme, zu denen ja auch Liabolo zählt, brauchen nicht auf die oben beschriebene Weise installiert werden; sie laufen nämlich streng genommen nicht unter Windows selbst, sondern innerhalb der einer *Java Runtime Environment* (JRE). Diese JRE muss bereits auf dem System installiert sein und dient sozusagen als Vermittlungsschicht zwischen dem Betriebssystem (in diesem Fall Windows) und dem Java Programm. Java Programme werden demnach nicht in eine windows-native Maschinensprache übersetzt, sondern in die Pseudo-Maschinensprache „Bytecode“, die von der JRE als virtuelle Maschine ausgeführt wird.

Aus diesem Grund ist es für Java Programme ausreichend, einfach auf das System (mit vorhandener JRE) kopiert zu werden und mittels des

```
java 'Programmname'
```

Kommandos gestartet werden, wobei 'Programmname' die Java-Klasse mit der main()-Methode darstellt.

Eine weitere Möglichkeit ist es, alle Klassen und sonstigen Dateien in eine „Jar¹ Datei“ zusammenzufassen und diese Datei durch das Kommando

```
java -jar 'Programmname.jar'
```

zu starten, wobei 'Programmname.jar' die erstellte Jar-Datei darstellt.

Da jedoch solche Vorgehensweisen, wie oben bereits beschrieben, für einige Windows-Nutzer eher ungewohnt sind, ist es auch für Java Programme überlegenswert, ein Installationsprogramm zu verwenden. Eine Möglichkeit wäre hier der *Advanced Installer*², dessen einfache Basisfunktionen kostenlos einsetzbar sind (siehe Abbildung 1).

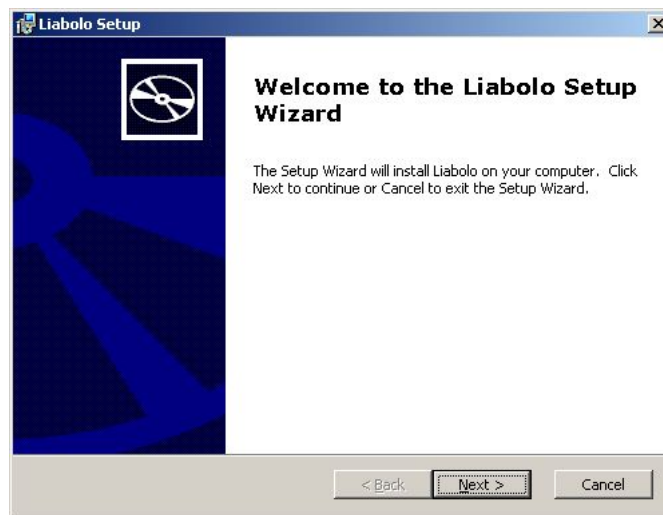


Abbildung 1: Durch den *Advanced Installer* erstellter Installations Wizard

¹Java Archive

²<http://www.advancedinstaller.com/>

Momentan wird Liabolo unter Windows mittels einer Batch-Datei namens „start.bat“ gestartet, hinter der sich eigentlich nur den Aufruf

```
java -jar liabolo.jar
```

verbirgt. Auch hier könnte man, um dem Windows-User entgegenzukommen, den Aufruf in eine „Exe-Datei“ packen. Das ließe sich z.B. mit dem *Xenoage Java Exe Starter*³ realisieren.

2.2 Linux

Die plattformunabhängige Einsetzbarkeit von Java Programmen bringt mit sich, dass diese auf allen Systemen auf die gleiche Art gestartet werden können, nämlich — wie im vorangegangenen Abschnitt beschrieben — durch das von der JRE gelieferte Programm „java“. Somit kann einfach das komplette Programmverzeichnis (oder eine Jar Datei) auf das Linux-System kopiert werden und gestartet werden. Dem Großteil der Linux-User wird diese einfache Vorgehensweise keine Probleme bereiten bzw. bereits bekannt sein. Doch auch für Linux gibt es komfortable Installationshilfen in Form von sogenannten Paketmanagern.

Dabei wird aus dem Programm, der Dokumentation, den Konfigurationsdateien, usw. ein Paket „geschnürt“, welches von einem Paketmanager verwaltet werden kann. Dadurch wird die Installation, das Update und das Entfernen von Software erleichtert. Es haben sich insbesondere zwei Paketverwaltungen in der Linuxwelt durchgesetzt: der Paketmanager von *Red Hat* und die Paketverwaltung von *Debian*. Diese beiden sollen hier kurz vorgestellt werden.

2.2.1 Red Hat Package Manager

Der *Red Hat Package Manager* (RPM) wurde im Rahmen der Red Hat⁴ Linux Distribution entwickelt und ist mittlerweile von vielen anderen Distributionen als Paketverwaltung übernommen worden.

RPM-Pakete tragen üblicherweise den Namen des Programms, die Versionsnummer und eine Releasenummer im Paketnamen, gefolgt von '.rpm' als Dateiendung, z.B. `liabolo-1.0-1.rpm`. Die Verwaltung der Pakete erfolgt

³<http://www.xenoage.com/jestart/>

⁴<http://www.redhat.com/>

durch das Konsolen-Programm 'rpm', für das auch graphische Frontends existieren.

Durch den Befehl

```
rpm -i 'Paketname'
```

wird das Programm, welches sich im Paket 'Paketname' befindet, installiert. 'rpm' überprüft dabei, ob Abhängigkeiten zu anderen nicht installierten Programmen bestehen und meldet dies ebenso wie Konflikte, die bei der Installation entstehen, z.B. wenn eine Datei oder gar das zu installierende Programm bereits vorhanden ist.

Pakete werden durch

```
rpm -e 'Paketname'
```

wieder entfernt, wobei 'rpm' prüft, ob noch installierte Programme von dem zu entfernenden abhängen und mittels

```
rpm -U 'Paketname'
```

können bereits installierte Pakete durch neuere „upgedatet“ werden.^[1]

Möchte man Informationen zu einem Paket kann man diese durch

```
rpm -q 'Paketname'
```

erhalten und

```
rpm -qa
```

zeigt alle auf dem System installierten Pakete.

Die Beschreibung der RPM-Paketerstellung ist ein wenig umfangreicher und soll im Rahmen dieser Arbeit nicht behandelt werden.

2.2.2 Debian Paketverwaltung

Die Linux Distribution *Debian GNU/Linux*⁵ entwickelte ihrerseits eine eigene Paketverwaltung. Grundlage dafür ist das Programm 'dpkg', das sozusagen das Pendant zum oben beschriebenen 'rpm' ist. Im Laufe der Zeit haben sich

⁵<http://www.debian.org/>

dazu Frontends gebildet, `'dselect'` und `'apt'`⁶. `dselect` ist eine textbasierte Benutzungsoberfläche und war bis zur Debian-Version 2.2 das wichtigste Werkzeug, um Pakete zu verwalten. Mittlerweile wurde es von `'apt'` abgelöst, das allerdings ein Kommandozeilen-Tool ist.⁷

Debian-Pakete haben die Endung `'deb'` und werden folgendermaßen mit `dpkg` installiert:

```
dpkg -i 'Paketname'
```

Das Entfernen erfolgt mit

```
dpkg -r 'Paketname'
```

Die Option `'-r'` entfernt nur Programmdateien, mit `'-P'` werden zusätzlich auch die Konfigurationsdateien gelöscht.

Heutzutage wird allerdings eher `apt` benutzt. `apt` unterteilt sich in mehrere kleine Programme. Zum Installieren ist `'apt-get'` wichtig:

```
apt-get install 'Programmname'
```

`apt-get` benötigt allerdings einen Hinweis auf die Quellen, wo die Pakete zu finden sind. Diese werden in eine Datei namens `'sources.list'` eingetragen. Solche Quellen können sich lokal auf der Festplatte oder CD-ROM befinden, aber auch auf einem entfernten FTP-Server. Das Programm `'apt-setup'` bietet Hilfe beim Eintragen neuer Quellen.

`apt-get install` überprüft auch auf Abhängigkeiten von Paketen und installiert nicht vorhandene, benötigte Pakete gleich mit.

Pakete können mit

```
apt-get remove 'Programmname'
```

entfernt werden mitsamt der Pakete zu denen Abhängigkeiten bestanden (sofern diese nicht noch von anderen Paketen benötigt werden).

`apt` bietet noch eine Vielzahl von Funktionen, die hier nicht weiter besprochen werden können. Bei Interesse empfiehlt sich ein Blick in das „APT HOWTO“ [2]. Auch das Erstellen von Debian-Paketen kann hier nicht behandelt werden.

⁶ „A Package Tool“

⁷ als Benutzungsoberfläche gibt es das Programm `'aptitude'`

3 Plattformunabhängige Verteilung und Installation

3.1 Quellcode als Zip-Datei oder tarball

Die Offenlegung des Quellcodes ist ja die zentrale Eigenschaft von Open Source Projekten⁸. Und die Distribution eines Programms durch den Quellcode, anstatt ausführbarer Binaries, stellt vielleicht die ehrlichste und ursprünglichste Weise dar. Der Benutzer kann — entsprechendes „Know-How“ vorausgesetzt — Einblick den in Code nehmen und diesen selbst in ein ausführbares Programm übersetzen.

In der Linux-Welt werden dazu sogenannte „tarballs“ eingesetzt. Die Quelltexte werden durch das Archivierungswerkzeug `'tar'` in eine Datei zusammengefasst und mittels des Programms `'gzip'` komprimiert. Dabei entsteht eine Datei mit der Endung `'tar.gz'` oder `'tgz'`, die tarball genannt wird.

Weit verbreitet sind auch die „Zip-Archive“. Die Quelltexte werden durch ein Zip-Programm zu einer Datei zusammengefasst und komprimiert. Die entstehende Datei trägt die Endung `'zip'`.

In beiden Fällen besorgt sich der Benutzer die komprimierte Quellcodedatei und entpackt diese. Anschließend übersetzt er mit einem Compiler den Sourcecode und erhält so das lauffähige Programm.

Leider ist nicht jeder Benutzer gewillt oder in der Lage, obige Schritte durchzuführen. Auch kann es beim Übersetzen des Quellcodes zu Fehlern kommen, die der Benutzer oft nicht selbst beheben kann. Aus diesem Grund werden neben dem Sourcecode auch Archive mit bereits übersetzten Programmen angeboten.

Wie bereits in Abschnitt 2.1 erwähnt, handelt es sich bei Liabolo um ein Java-Programm und kann daher, auch in bereits übersetzter Form, einfach entpackt und ausgeführt werden. Auf diese Art ist es auf jedem System, auf dem auch eine JRE lauffähig ist, einfach einsetzbar.

⁸Das wird ja schon bereits durch den Namen Open Source ausgedrückt

3.2 make und ant

Bei größeren Programmen wird der gesamte Quellcode nicht mehr in einer Datei gehalten, sondern auf mehrere Dateien verteilt. Das entspricht dem Konzept der Modularisierung, wo Teilprobleme in einzelne Module zusammengefasst werden und auch dem Klassenkonzept der objektorientierten Programmierung, wo die Problemwelt durch Objekte dargestellt wird, die verschiedenen Objektklassen zugehörig sind. Dadurch lassen sich z.B. bei der Wartung der Software einzelne Komponenten leichter verändern oder austauschen.

Ein „Build-Tool“ hilft beim kompilieren des Quellcodes. Es verwaltet die Quelldateien und deren Abhängigkeiten untereinander. Alle notwendigen Operationen zum Kompilieren des Quellcodes können mit einem Build-Tool zusammengefasst werden und somit der gesamte Prozess der Quellcodeübersetzung auf einen einzigen Schritt reduzieren werden. Die hier vorgestellten Werkzeuge 'make' und 'ant' stellen solche Build-Tools dar.

3.2.1 make

make ist das ältere der beiden Tools. Es wurde bereits früh unter Unix, insbesondere für die Programmiersprache C, eingesetzt. Unter Linux wird allgemein „GNU make“ benutzt, das aber im Wesentlichen mit dem traditionellen make übereinstimmt.

make benötigt eine Steuerungsdatei, ein sogenanntes „Makefile“, das die notwendigen Informationen zum Erstellen des Programms enthält. Ein Makefile besteht aus Regeln, die folgende Form haben[3]:

```
Ziel:      Abhängigkeit(en)
           Kommando
           Kommando
           ...
```

Hier wird ein einfaches Makefile gezeigt, das nur aus einer einzigen Regel besteht. Es wird aus einigen Tex-Files mit dem 'latex' Kommando ein Dokument erzeugt[3]:

```
book.dvi: ch01.tex ch02.tex ch03.tex ch04.tex ch05.tex \
          ch06.tex ch07.tex ch08.tex ch09.tex book.tex
          latex book.tex
```

Ruft man nun `make` ohne Argument auf, so nimmt es das erste Ziel („Target“) des Makefile (in diesem Fall gibt es nur eins, nämlich `'book.dvi'`) und führt die darin enthaltenen Kommandos aus.

Diese Kommandos sind allerdings plattformabhängig; nun wird mit `ant` ein Build-Tool vorgestellt, das plattformunabhängig einsetzbar ist.

3.2.2 ant

`ant` ist ein Projekt der Apache Gruppe⁹ und steht für „A Neat Tool“. `ant` war ursprünglich Teil des Tomcat-Projekts, doch man erkannte seine Nützlichkeit und formte ein eigenständiges Programm daraus^[4]. Es basiert auf Java und kann, wie Java selbst auch, plattformunabhängig benutzt werden¹⁰.

Genauso wie `make` benötigt auch `ant` eine Steuerungsdatei (bzw. Builddatei), die im XML-Format gehalten ist. Eine Builddatei muss ein Wurzelelement namens *project* enthalten. Innerhalb des Wurzelelements werden die Ziele(*target*) beschrieben, die auch einen Namen tragen und von anderen targets abhängig sein können. Die Ziele wiederum bestehen aus Aufgaben (*tasks* genannt), die ausgeführt werden müssen, um das Ziel zu erfüllen.^[5]

Hier ein (gekürztes) Beispiel aus dem Apache Ant User Manual^[6]:

```
<project name="MyProject" default="dist" basedir=".">

  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist" location="dist"/>

  <target name="init">
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
    description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>
</project>
```

⁹<http://ant.apache.org>

¹⁰Als Javaprogramm benötigt `ant` natürlich eine JRE.

```
</target>

</project>
```

Auch Liabolo besitzt ein Ant-Buildfile, das leider zu umfangreich ist, um es in diesem Whitepaper abzudrucken, dennoch lohnt sich sicherlich eine Einsichtnahme. Die Builddatei von Liabolo („build.xml“) befindet sich auf der oberen Ebene des Quellcodeverzeichnis.

3.3 Java Webstart

Java Webstart ist Teil der JRE und somit bereits vorhanden, wenn die JRE auf dem System installiert ist. Es erlaubt dem Softwareentwickler, sein Programm auf einem Webserver zu installieren, so dass Benutzer das Programm über einen Webbrowser starten können. Es muss also auf Benutzerseite keine Installation vorgenommen werden und Entwickler können ihre Applikationen für den Anwender unmittelbar und unkompliziert nutzbar machen.

Die Anwendung von Java Webstart ist für den User einfach. Er braucht lediglich die URL der Startdatei (JNLP¹¹-Datei) der Applikation im Browser einzugeben. Eine andere Möglichkeit ist die Benutzung des `javaws`-Programm, das sich im JRE-Verzeichnis befindet. Beispiele für Webstart-Anwendungen sind unter <http://java.sun.com/products/javawebstart/demos.html> zu finden.

Serverseitig muss ein Jar-File mit der Applikation vorhanden sein und eine entsprechende JNLP-Datei. Zusätzlich muss der Webserver so eingestellt werden, dass er als MIME-Typ `application/x-java-jnlp-file` für JNLP-Datei ausgibt. Der genaue Prozess der Einrichtung einer Webstart-Anwendung und das Erstellen einer JNLP-Datei kann der Dokumentation über Java Webservices entnommen werden[7].

4 Programmstruktur

Ob Liabolo auch via Webstart verfügbar sein wird, steht noch nicht fest. Auf jeden Fall gibt es aber eine installierbare Version. In diesem abschließenden Abschnitt wird beschrieben, wo auf der Festplatte sich Liabolodateien befinden bzw. befinden werden.

¹¹Java Network Launching Protocol

4.1 Installationsort

Der Installationsort von Liabolo kann vom Benutzer frei gewählt werden. Bestimmte Orte bieten sich traditionsgemäß an:

- **Windows:**
C:\ Liabolo *oder*
C:\ Programme\ Liabolo
- **Linux:**
/opt/liabolo *oder*
/usr/local/liabolo

4.2 Aufbau des Installationsverzeichnis

Ist Liabolo ins gewünschte Verzeichnis entpackt worden, bietet sich dem User dort folgende Verzeichnisstruktur:

- **bin**
Hier liegt die Jar-Datei von Liabolo
- **config**
Hier liegen die Konfigurationsdatei, sowie XSL-Dateien, die für den Export benötigt werden.
- **exit**
Die eXist-Datenbank befindet sich in diesem Verzeichnis.
- **lib**
In diesem Verzeichnis liegen alle externen Bibliotheken, von denen Liabolo Gebrauch macht.
- **log**
Hier erstellt Liabolo die Log-Dateien, die Ereignisse während der Benutzung von Liabolo aufzeichnen.
- Außerdem befindet sich eine Kopie der GPL („LICENSE.txt“), sowie Startskripte für Windows („start.bat“) und Linux („start.sh“) im Verzeichnis.

4.3 Lage der Userdaten

Liabolo soll auch im Mehrbenutzerbetrieb laufen. Dazu müssen alle benutzerspezifischen Daten im Heimverzeichnis des Benutzer gehalten werden.

Speziell betrifft das die individuellen Datenbankeinträge (alle .dbx Dateien im data-Verzeichnis von exist) sowie die Konfigurationsdatei 'config.xml'. Zur Zeit ist das bei Liabolo leider noch nicht der Fall. Alle oben genannten Dateien werden noch global im Liabolo-Installationsverzeichnis gehalten. Das soll so schnell wie möglich geändert werden.

5 Schluss

Java Programme werden häufig nur als Jar-Datei oder Zip-Archiv mit den lauffähigen Klassen verbreitet. Für Linuxsysteme kann durchaus an dieser Praxis festgehalten werden. Für viele Windows-User könnte jedoch diese Form eher ungewohnt sein, daher wäre der Einsatz eines Installationswizards als Alternative eine Überlegung wert.

Zusätzlich müssen gerade im Rahmen eines Open Source Projekt die Quelltexte angeboten werden.

Ob sich der Einsatz von Java Webstart lohnt, sollte noch zur Diskussion gestellt werden. Für den ernsthaften Einsatz von Liabolo ist sicherlich eine lokale Installation die bessere Lösung. Bei der Benutzung über Webstart muss der Benutzer ständig online sein und es stellt sich die Frage, wie schnell die Kommunikation von User und Programm über das Internet erfolgen kann.

Literatur

- [1] Edward C. Bailey. Maximum RPM. Red Hat, Inc. 2000. <http://www.rpm.org/max-rpm/>
- [2] APT HOWTO. <http://www.debian.org/doc/manuals/apt-howto/index.en.html>
- [3] Unixwerkzeuge: Make und Makefiles. Die Linuxfibel. <http://www.linuxfibel.de/make.htm>
- [4] Frequently Asked Questions. The Apache Ant Project. <http://ant.apache.org/faq.html>
- [5] Softwareentwicklung mit Ant. Java User Group Deutschland 2002. <http://www.java.de/article/articleview/63/1/1>
- [6] Apache Ant User Manual. The Apache Ant Project. <http://ant.apache.org/manual/>
- [7] Java Web Start 1.4.2 Developer Guide. <http://java.sun.com/j2se/1.4.2/docs/guide/jws/developersguide/contents.html>