TUDIIR Toolkit Overview

David Urbansky, Klemens Muthmann
TU Dresden, Department of Systems Engineering, Chair Computer Networks, IIR Group, Germany

July 26, 2010

Contents

Chapter 1

Introduction

Internet Information Retrieval (IIR) is a research domain in computer science that is concerned with the retrieval, extraction, classification, and presentation of information from the Internet. This toolkit provides functionality which is often needed to perform IIR tasks such as crawling, classification, and extraction of various information types.

1.1 What is TOOLKIT?

1.2 What is TOOLKIT NOT?

1.3 License

The complete source code is licensed under the Apache License 2.0. All source files should include the following license snippet at the very top.

```
Copyright 2010 David Urbansky, Klemens Muthmann
Licensed under the Apache License, Version 2.0 (the "License"); you may not
use this file except in compliance with the License. You may obtain a copy of
the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

1.4 Alternative and Complimentary Toolkits

Some functionalities of this toolkit are covered in other libraries. Before you start using TOOLKIT you might want to take a look at these alternatives. The objective of TOOLKIT is to create new functionalities or improve existing ones, we do not intend to reinvent the wheel. For example, TOOLKIT has no functionality for part-of-speech tagging (PoS tagging) since there are many toolkits that do this task pretty accurate already. In case you do not find the functionality you are looking for in this toolkit, you probably will in one of the following toolkits.

1. **AlchemyAPI** [?] is a commercial web-service that can be used via several programming languages. The service offers named entity recognition, text classification, language identification,

concept tagging, keyword extraction, content scraping, and web page cleaning. The service comes in 4 variants: free, basic, professional, and metered.

2. **Apache Mahout** [?] is a Java-based machine learning library. Its main features are collaborative filtering, user and item based recommenders, (fuzzy k-means clustering, mean shift clustering, latent dirichlet process allocation, singular value decomposition, parallel frequent pattern mining, complementary naive bayes classifier, and a random forest decision tree based classifier. The library is licensed under the Apache Software license.

3. **Balie** [?] is a Java-based information extraction library. Its main features are language identification, tokenization, sentence boundary detection, and named entity recognition (using dictionaries). The library is licensed under the GNU GPL and supports English, German, French, Romanian, and Spanish as input languages.

4. **ContentAnalyst** [?] is a commercial platform for text analytics. The platform's main features are concept search, dynamic clustering, near-duplicate document identification, automatic summarization, text classification, and latent semantic indexing.

5. The **Dragon Toolkit** [?] is a Java-based development package for information retrieval and text mining. Its main features are text classification, text clustering, text summarization, and topic modeling.

6. **FreeLing** [?] is a natural language processing library written in C++. Its main features are Text tokenization, sentence splitting, morphological analysis, sSuffix treatment, retokenization of clitic pronouns, flexible multiword recognition, contraction splitting, probabilistic prediction of unkown word categories, named entity detection, recognition of dates, numbers, ratios, currency, and physical magnitudes, PoS tagging, chart-based shallow parsing, named entity classification, WordNet based sense annotation and disambiguation, Rule-based dependency parsing, and nominal correference resolution. It is licensed under GPL and supports Spanish, Catalan, Galician, Italian, English, Welsh, Portuguese, and Asturian as languages. An online demo is available under `http://garraf.epsevg.upc.es/freeling/demo.php`.

7. **GATE** [?] is a Java-based text mining and processing framework. The framework itself comes with few text processing features but many plugins can be used and chained into a text engineering pipeline. The framework is licensed under the GNU Lesser General Public License.

8. The **Illinois Cognitive Computation Group** [?] has a list of ready to use programs for semantic role labeling, text chunking, named entity tagging, named entity discovery, PoS tagging, unsupervised rank aggregation, and named entity similarity metrics.

9. **Julie NLP** [?] is a Java-based toolkit of UIMA based text processing components. The toolkit can be used for semantic search, information extraction, and text mining. The Toolkit is licensed under the Common Public License.

10. **Language Computer** [?] provides commercial products for sentence splitting, tokenization, PoS tagging, named entity recognition, co-reference resolution, attribute extraction, relationship extraction, event extraction, question answering, and text summarization.

11. **Lingo3G** [?] is a text clustering engine that organizes text collections into hierarchical clusters. The software is commercial but [?] offers an open source alternative for text clustering algorithms written in Java. The algorithms integrate with other programming or scripting languages such as PHP, Ruby, and C# too.

12. **LingPipe** [?] is a text processing toolkit using computational linguistics. LingPipe is written in Java. Its main features are topic classification, named entity recognition, clustering, PoS tagging, sentence detection, spelling correction, database text mining, string comparisons, interesting phrase detection, character language modeling, chinese word segmentation, hyphenation and syllabification, sentiment analysis, language identification, singular value decomposition, logistic regression, expectation maximization, and word sense disambiguation. LingPipe is available under a free license for academic use and several commercial licenses.

13. **Mallet** [?] is a Java-based toolkit for statistical natural language processing. Its main features are text classification, sequence tagging (PoS tagging), topic modeling, and numerical optimization. The toolkit is licensed under the Common Public License.

14. **MinorThird** [?] is a Java-based toolkit for text processing. Its main features are annotating text, named entity recognition, and text classification. The toolkit is licensed under the BSD license.

15. **MontyLingua** [?] is a Python and Java-based toolkit for natural language processing (English only). Its main features are tokenization, PoS tagging, lemmatization, and natural language summarization. The toolkit is free for non-commercial use and licensed under the MontyLingua version 2.0 License.

16. **MorphAdorner** [?] is a Java-based command line program for text processing. Its main features are language recognition, lemmatization, name recognition, PoS tagging, noun pluralization, sentence splitting, spelling standardization, text segmentation, verb conjugation, and word tokenization. The program is licensed under a NCSA style license.

17. **NaCTeM Software Tools** [?] are programs for natural language processing and text mining that are made available by the National Centre for Text Mining. The programs include functionality for PoS tagging, syntactic parsing, named entity recognition, sentence splitting, text classification, and sentiment analysis.

18. **NLTK** [?] is a Python-based natural language processing toolkit. Its main features are tokenization, stemming, PoS tagging, text classification, and syntactic parsing. The toolkit is licensed under the Apache 2.0 license.

19. **OpenCalais** [?] is a web service that performs named entity recognition, fact and event extraction. The web service is free for commercial and non-commercial use but limited to 50,000 transactions a day. A professional plan is available too including more transactions and an service license agreement.

20. The **RASP System** [?] is a C and Lisp-based toolkit for natural language processing (English only). Its main features are tokenization, PoS tagging, lemmatization, morphological analysis, and grammar-based parsing. The toolkit is free for non-commercial use and licensed under the RASP System License.

21. The **Rosette Linguistic Platform** [?] is a software suite that can perform name translation, name matching, named entity recognition, morphological analysis, and language identification. The suite works for 55 European, Asian, and Arabic languages. The software is a commercial product.

22. **Stanford NLP** [?] is a set of Java-based natural language processing libraries. Their main features are PoS tagging, named entity recognition, Chinese word segmentation, and classification. The software distributions are licensed under the GNU Public License.

23. **SRILM - The SRI Language Modeling Toolkit** [?] is a C++-based toolkit for language modeling. Its main features are speech recognition, statistical tagging and segmentation, and machine translation. The toolkit is free for non-commercial use and licensed under the SRILM Research Community License.

24. **TextAnalyst** [?] is a commercial text processing software offering text summarization, semantic information retrieval, meaning extraction, and text clustering.

25. **VisualText** [?] is a natural language processing software that addresses named entity recognition, text indexing, text filtering, text classification, text grading, and text summarization.

26. **WEKA** [?] is a Java-based machine learning and data mining library. The library contains a large set of machine learning algorithms such as Support Vector Machines, Neural Networks, Naive Bayes, k-nearest neighbor for but not limited to (text) clustering, (text) classification, and regression. The library is licensed under the GNU General Public License.

Chapter 2

Installation and making it work

The TUDIIR toolkit is managed using Subversion (SVN) (see 3). It is build and tested automatically on a weekly basis using Apache Maven and Hudson CI. Bugs might be reported using the Mantis bugtracker. The project encoding must be set to UTF-8. To support unavailable libraries we manage our own Maven repository using Sonatype Nexus. How to use these components is explained in the next sections.

At the moment each of the systems manages its own user base so you need to register with each one individually. When you start working with the toolkit you might consider sending an E-Mail to the administrator to get access to Mantis, Nexus and Hudson. Provide your name, the name of your advisor and the reason why you need to work with the toolkit and you will get your logins.

2.1 Building the toolkit using Apache Maven

Apache **Maven needs to be installed** before building the toolkit. How to do this is explained here: Maven installation. There is one necessary manual step before you can start building the toolkit. You need to add our Nexus repository to your local maven settings. Do this by **locating or creating the settings.xml file in your local home folder:** `%YOUR_HOME_FOLDER%/.m2/settings.xml` and adding the following content:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
<mirrors>
    <mirror>
      <!--This sends everything else to /public -->
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>http://www.effingo.de/nexus/content/groups/public</url>
    </mirror>
  </mirrors>
  <profiles>
    <profile>
      <id>nexus</id>
      <!--Enable snapshots for the built in central repo to direct -->
      <!--all requests to nexus via the mirror -->
      <repositories>
        <repository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </repository>
      </repositories>
     <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
    <!--make the profile active all the time -->
    <activeProfile>nexus</activeProfile>
  </activeProfiles>
  <servers>
    <server>
      <id>nexus</id>
      <username>your-username</username>
      <password>your-password</password>
      <filePermissions>664</filePermissions>
      <directoryPermissions>775</directoryPermissions>
      <configuration></configuration>
    </server>
  </servers>
</settings>
```

You need to set your username and your password in the server section. This can be obtained by sending an E-Mail to klemens.muthmann@tu-dresden.de stating who is your advisor and why you need to work with the toolkit. After completing the installation perform the following steps to build the toolkit using Maven:

1. Check out the code from SVN!
2. Open your favorite command line.
3. Change to the toolkits root folder.
4. Type `mvn clean install` to start the build process.

There also is an Eclipse plugin, that allows you to issue maven build from within Eclipse. It is quite beta so be careful with it. To use the toolkit together with the plugin in your own projects you need to update the plugins index (This index stores which libraries are available through Maven) and update it after you have successful run `maven install`. This can be achieved by choosing `Window` → `Show View` → `Other` → `Maven` → `Maven Repositories` inside Eclipse. A View appears showing all Maven artifact repositories available to the m2eclipse. Right klick `Local Repositories` → `Local repository` and choose `Rebuild Index`.

2.2 Regularly builds and tests using Hudson CI

Currently the toolkit is build automatically every week using Hudson CI. Be careful to check in only working code or Hudson will send you and your advisor an E-Mail about broken code. If this happens try to fix your code as fast as possible and check in again. You can get details about the problem by logging into Hudson. You can get a login by sending an E-Mail stating your advisor and why you nood to work with the toolkit to klemens.muthmann@tu-dresden.de

2.2.1 The Continuous Integration Game

Hudson supports a game that rewards people commiting code that improves the toolkit and punishing people breaking it. The leader (the person having the most points) is highly valued by the toolkit commiters community. The rules for the game are explained in detail in the next section.

Rules The rules of the game are:

- -10 points for breaking a build
- 0 points for breaking a build that already was broken
- +1 points for doing a build with no failures (unstable builds gives no points)
- -1 points for each new test failures
- +1 points for each new test that passes
- Adding/removing a HIGH priority PMD warning = -5/+5. Adding/removing a MEDIUM priority PMD warning = -3/+3. Adding/removing a LOW priority PMD warning = -1/+1.
- Adding/removing a violation = -1/+1. Adding/removing a duplication violation = +5/-5.
- Adding/removing a HIGH priority findbugs warning = -5/+5. Adding/removing a MEDIUM priority findbugs warning = -3/+3. Adding/removing a LOW priority findbugs warning = -1/+1
- Adding/removing a compiler warning = -1/+1.
- Checkstyle Plugin. Adding/removing a checkstyle warning = -1/+1.

2.3 Hello Toolkit - Your first application using the TUDIIR Toolkit

Project creation: Create a new Maven Project using the New Project wizard of Eclipse (See Fig. 2.1, 2.2 and 2.3).

Setting correct encoding On Windows and Mac encoding is set by default to some platform specific format. Since we do a lot of string processing correct and consistent character encoding is very important. Therefore we need to set encoding to UTF-8, which is the standard format that should be used by modern software. To do this right click on your new project. Choose `Properties` and set `Text file encoding` on the right side to `UTF-8` if not already set.

Adding the toolkit dependency to the project: Right click on the new project. In the context menu that appears choose *Maven → Add Dependency* (See Fig. 2.4 and 2.5). A search interface appears (See 2.6). If you followed the steps in Section 2.1 and installed the toolkit to your local Maven repository, you can type *toolkit* in the search interface (See Fig. 2.7 and add the dependency with a double click on the *de.tud.inf.rn.iir toolkit* entry.

Note: If you need to add further dependencies in the future you can use the same steps. The Maven plugin adds the dependency to your pom.xml file, which should look like Fig. 2.8.

Configuring your project Since Maven by default still uses Java 1.4 (very conservative), but the toolkit depends on Java 1.6 (very visionary) you need to configure the Maven Java compiler plugin to use Java 1.6 by adding the following markup to your pom.xml (add it after the description element):

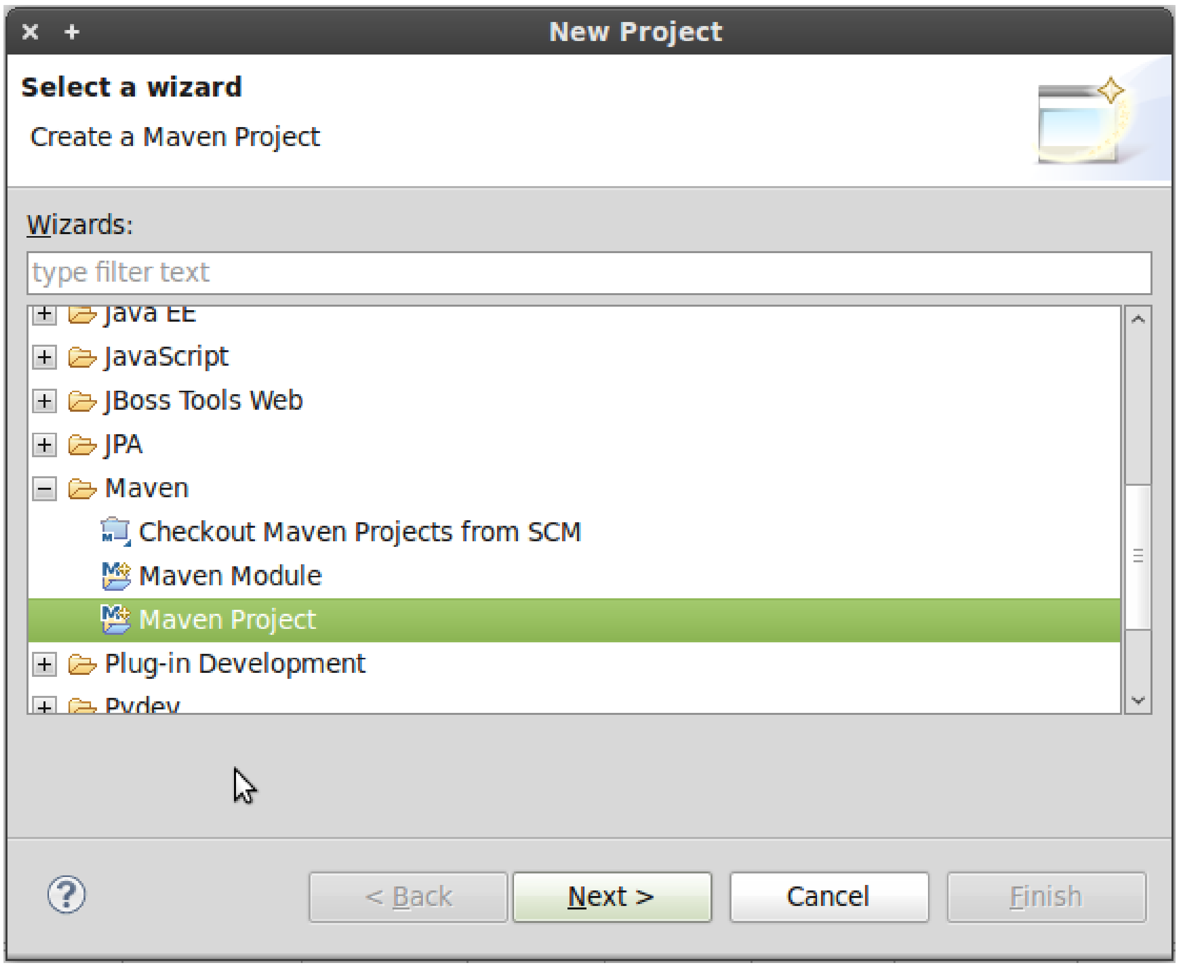
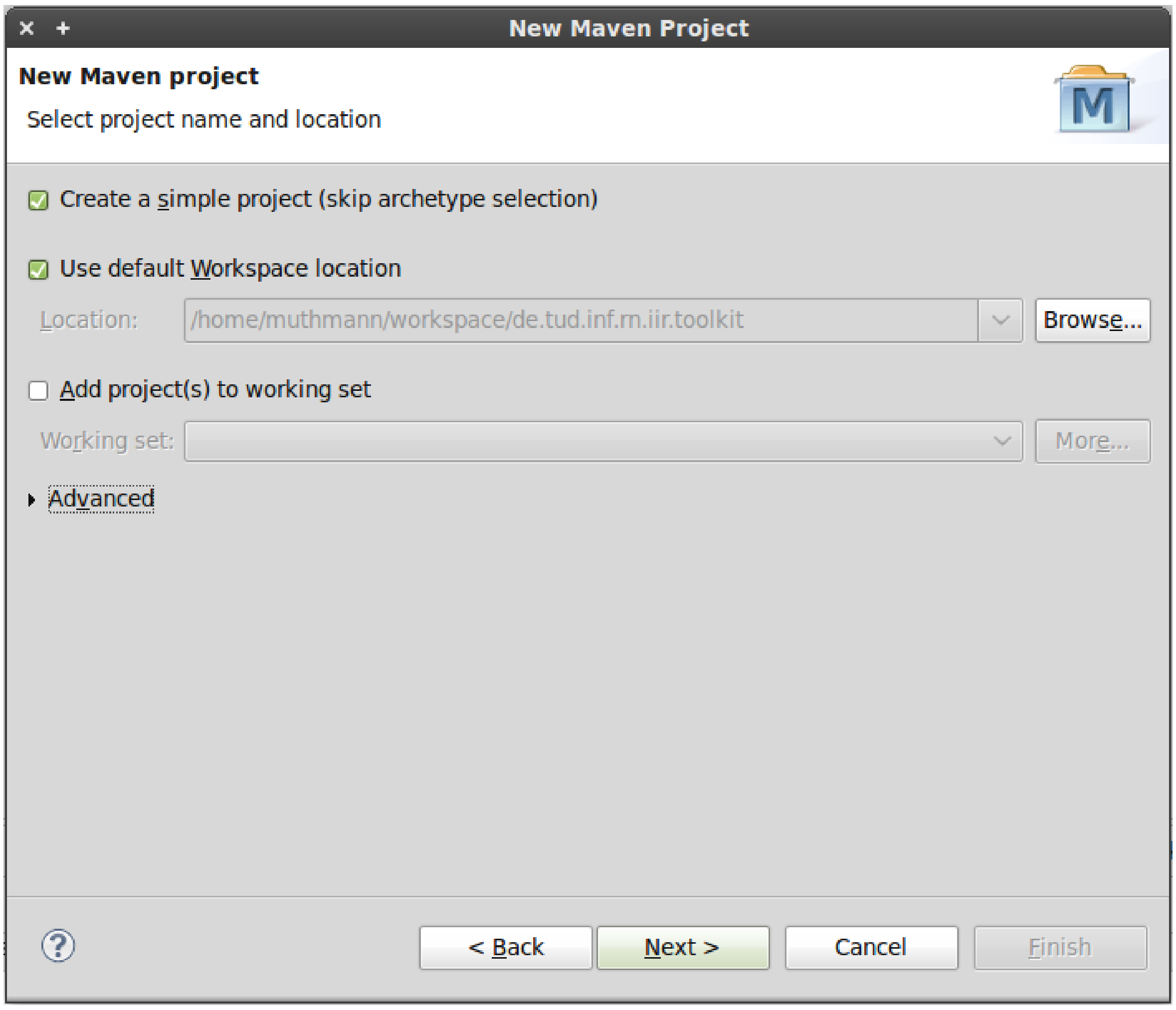


Figure 2.1: Create a new Maven Project.



New Maven Project

New Maven project

Select project name and location

[✓] Create a simple project (skip archetype selection)

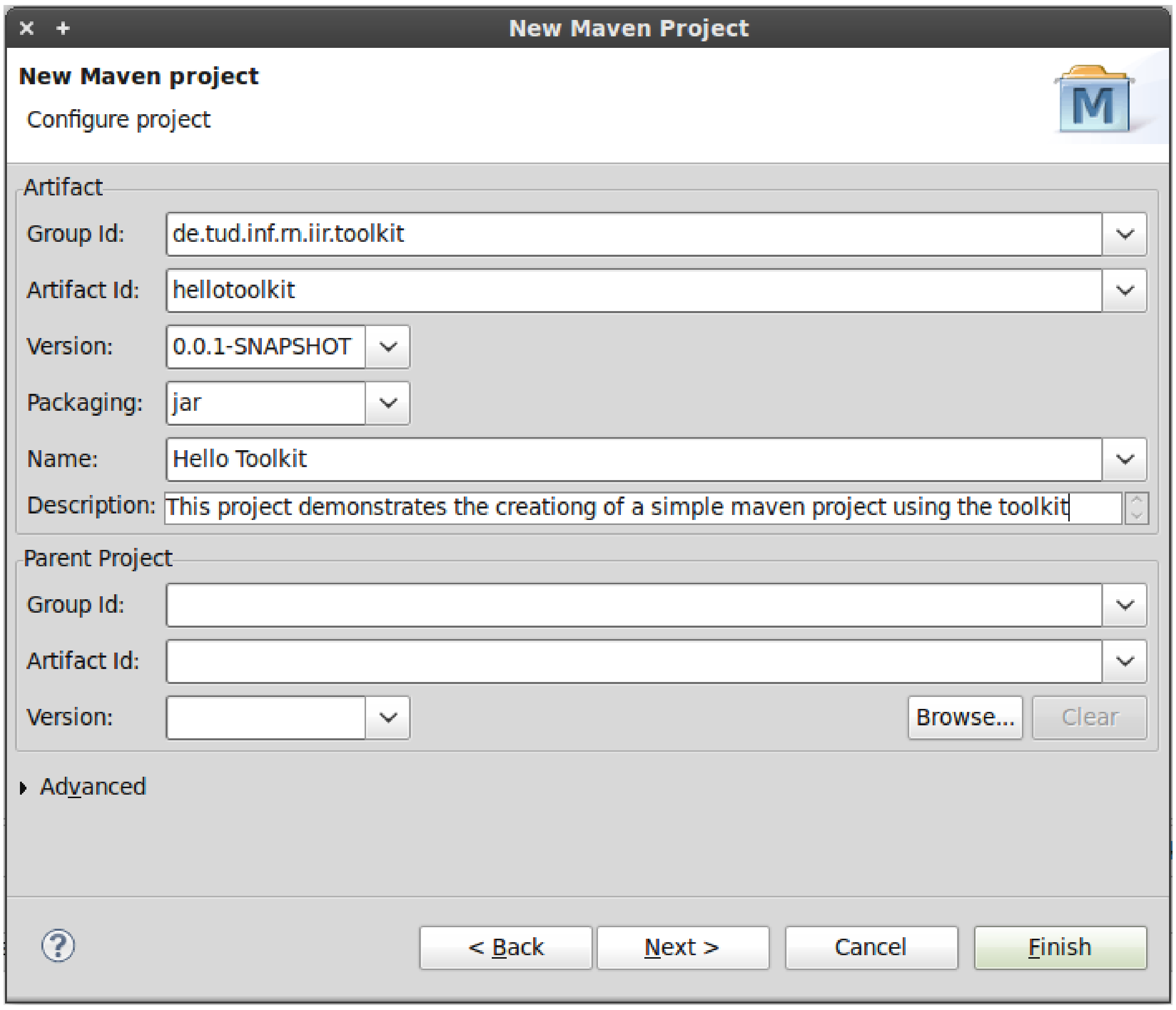
[✓] Use default Workspace location

Location: /home/muthmann/workspace/de.tud.inf.rn.iir.toolkit

[ ] Add project(s) to working set

Working set:

Figure 2.2: Choose to create a simple Maven Project.


New Maven Project

New Maven project

Configure project

Artifact

Group Id:	de.tud.inf.rn.iir.toolkit
Artifact Id:	hellotoolkit
Version:	0.0.1-SNAPSHOT
Packaging:	jar
Name:	Hello Toolkit
Description:	This project demonstrates the creationg of a simple maven project using the toolkit

Parent Project

Group Id:	
Artifact Id:	
Version:	

Figure 2.3: Enter detail information about your new Maven Project

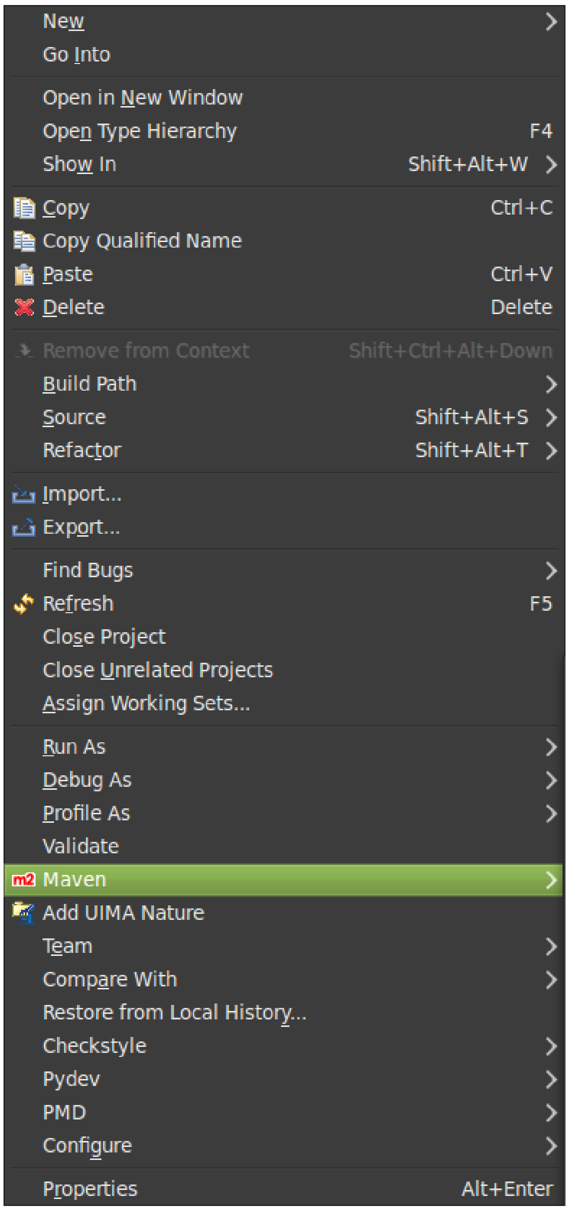


Figure 2.4: Maven Project Context Menu. Choose *Maven*

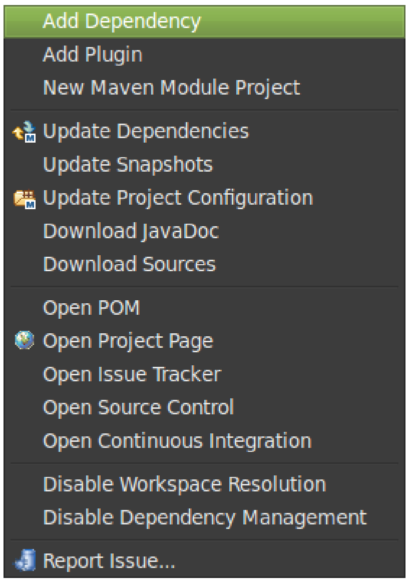


Figure 2.5: Create a new Maven project

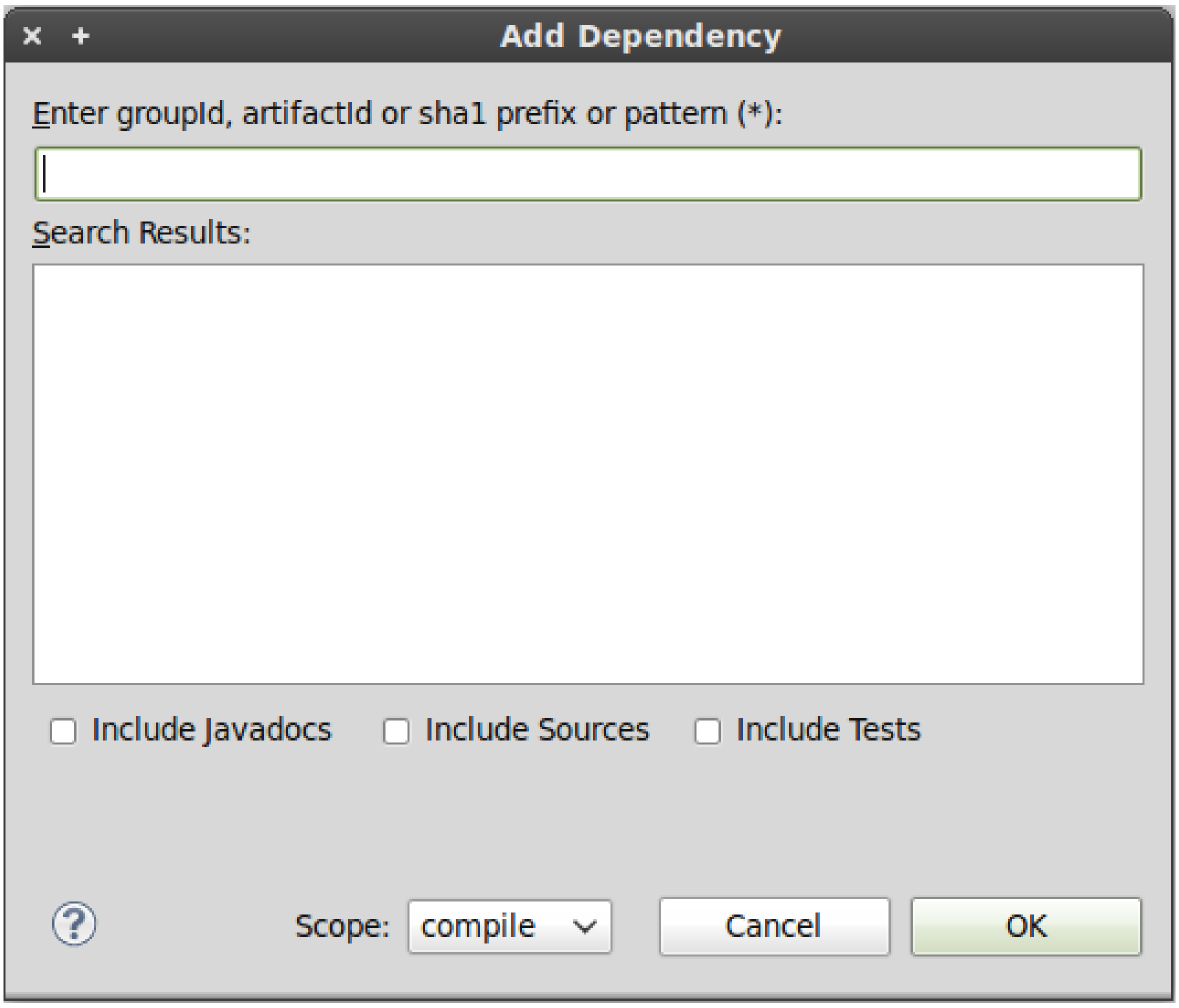
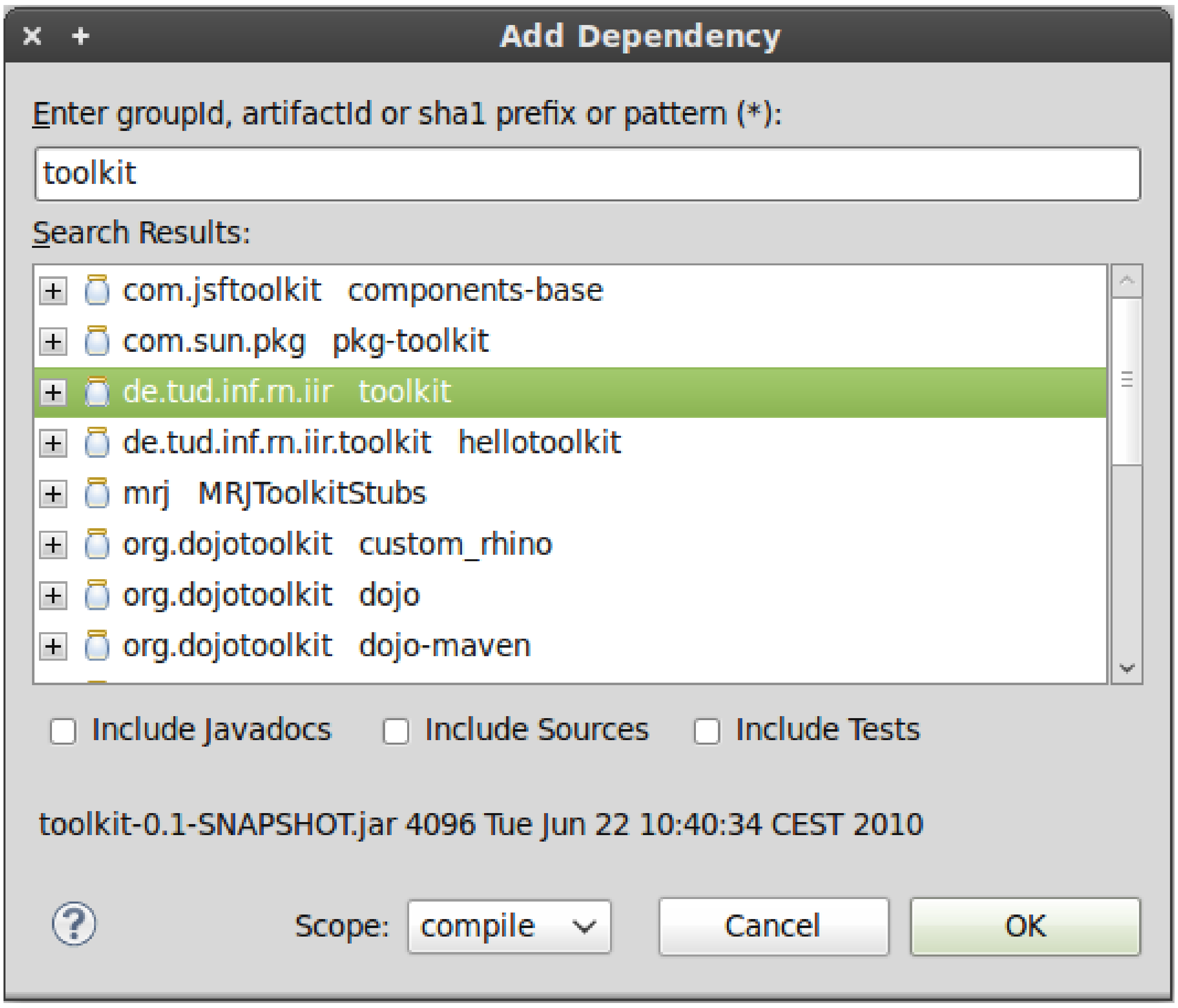


Figure 2.6: Search interface for Maven dependencies.



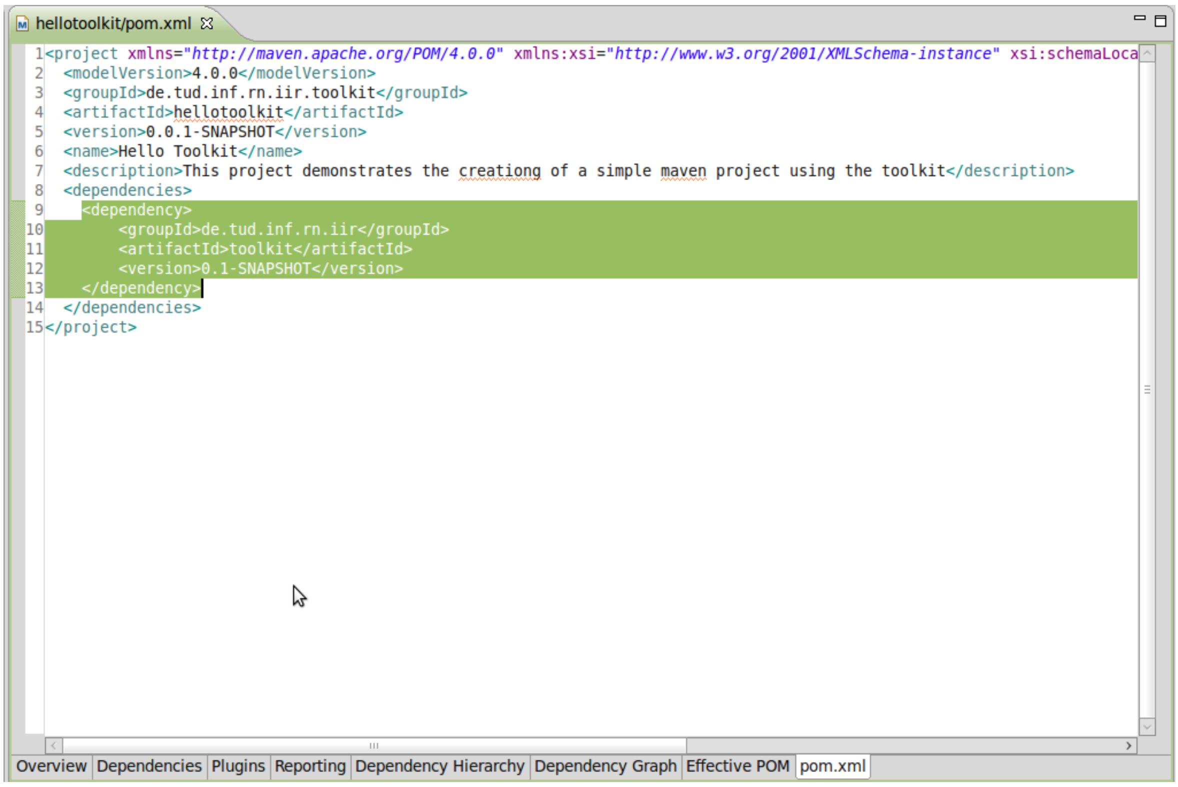
toolkit

com.jsftoolkit components-base
com.sun.pkg pkg-toolkit
de.tud.inf.rn.iir toolkit
de.tud.inf.rn.iir.toolkit hellotoolkit
mrj MRJToolkitStubs
org.dojotoolkit custom_rhino
org.dojotoolkit dojo
org.dojotoolkit dojo-maven

toolkit-0.1-SNAPSHOT.jar 4096 Tue Jun 22 10:40:34 CEST 2010

Scope: compile

Figure 2.7: Search results for *toolkit*



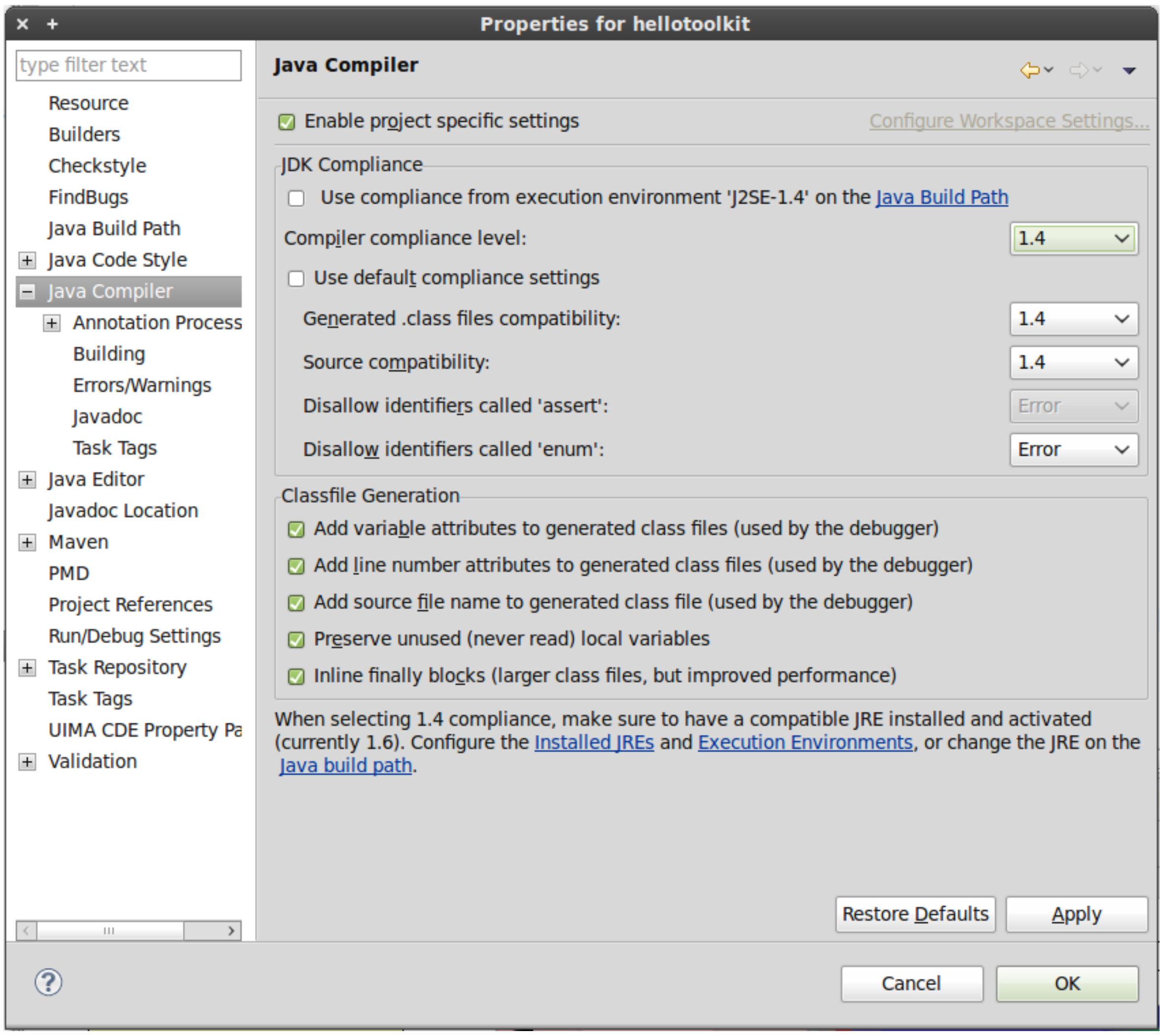
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLoca
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.tud.inf.rn.iir.toolkit</groupId>
  <artifactId>hellotoolkit</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Hello Toolkit</name>
  <description>This project demonstrates the creationg of a simple maven project using the toolkit</description>
  <dependencies>
    <dependency>
        <groupId>de.tud.inf.rn.iir</groupId>
        <artifactId>toolkit</artifactId>
        <version>0.1-SNAPSHOT</version>
    </dependency>
  </dependencies>
</project>
```

Figure 2.8: POM with added dependency on the TUD IIR Toolkit.

```
<build>
 <plugins>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <version>2.3.1</version>
   <configuration>
    <compilerArgument>-Xlint:all</compilerArgument>
    <showWarnings>true</showWarnings>
    <source>1.6</source>
    <target>1.6</target>
    <compilerArguments>
     <encoding>UTF-8</encoding>
    </compilerArguments>
    <showDeprecation>true</showDeprecation>
    <verbose>true</verbose>
    <encoding>UTF-8</encoding>
   </configuration>
  </plugin>
 </plugins>
</build>
```

The same step is necessary to tell Eclipse that it should use Java 1.6 instead of 1.4. For this purpose open the project properties via the context menu for example. In the tree to the left choose *Java Compiler* and change all three entries to 1.6. This is shown in Fig. 2.9 and Fig. 2.10.

Writing your first toolkit code Now you can start to write your first code. Your project will already contain the default Maven directory structure. Do not change this structure since Maven



Properties for hellotoolkit

Java Compiler

[✓] Enable project specific settings

Configure Workspace Settings...

JDK Compliance

[ ] Use compliance from execution environment 'J2SE-1.4' on the Java Build Path

Compiler compliance level: 1.4

[ ] Use default compliance settings

Generated .class files compatibility: 1.4

Source compatibility: 1.4

Disallow identifiers called 'assert': Error

Disallow identifiers called 'enum': Error

Classfile Generation

[✓] Add variable attributes to generated class files (used by the debugger)

[✓] Add line number attributes to generated class files (used by the debugger)

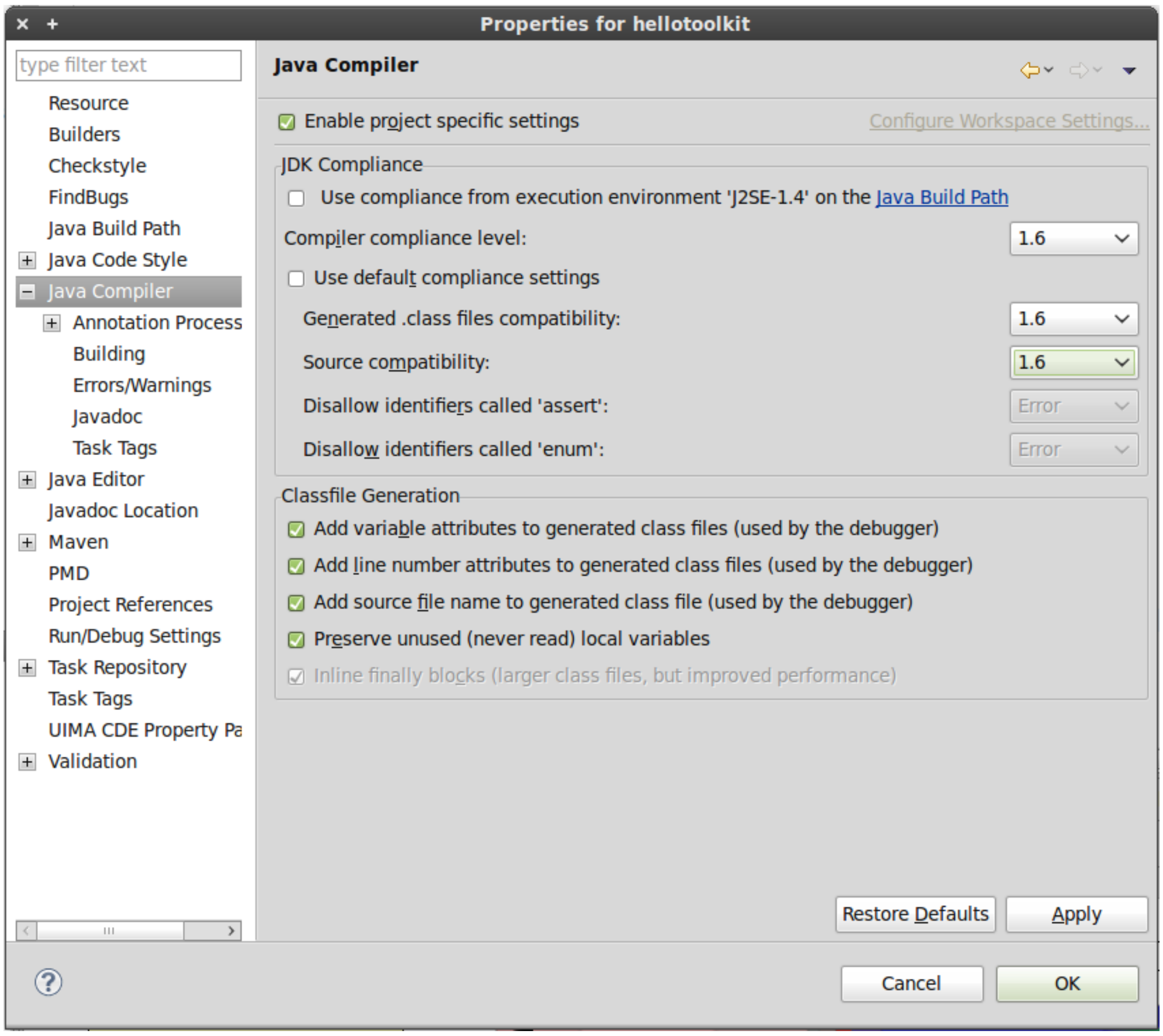
[✓] Add source file name to generated class file (used by the debugger)

[✓] Preserve unused (never read) local variables

[✓] Inline finally blocks (larger class files, but improved performance)

When selecting 1.4 compliance, make sure to have a compatible JRE installed and activated (currently 1.6). Configure the Installed JREs and Execution Environments, or change the JRE on the Java build path.

Figure 2.9: Eclipse uses Java 1.4 by default for Maven projects



Properties for hellotoolkit

type filter text

- Resource
- Builders
- Checkstyle
- FindBugs
- Java Build Path
- Java Code Style
- Java Compiler
  - Annotation Process
  - Building
  - Errors/Warnings
  - Javadoc
  - Task Tags
- Java Editor
- Javadoc Location
- Maven
- PMD
- Project References
- Run/Debug Settings
- Task Repository
- Task Tags
- UIMA CDE Property Pa
- Validation

Java Compiler

✓ Enable project specific settings — Configure Workspace Settings...

JDK Compliance

☐ Use compliance from execution environment 'J2SE-1.4' on the Java Build Path

Compiler compliance level:	1.6

☐ Use default compliance settings

Generated .class files compatibility:	1.6
Source compatibility:	1.6
Disallow identifiers called 'assert':	Error
Disallow identifiers called 'enum':	Error

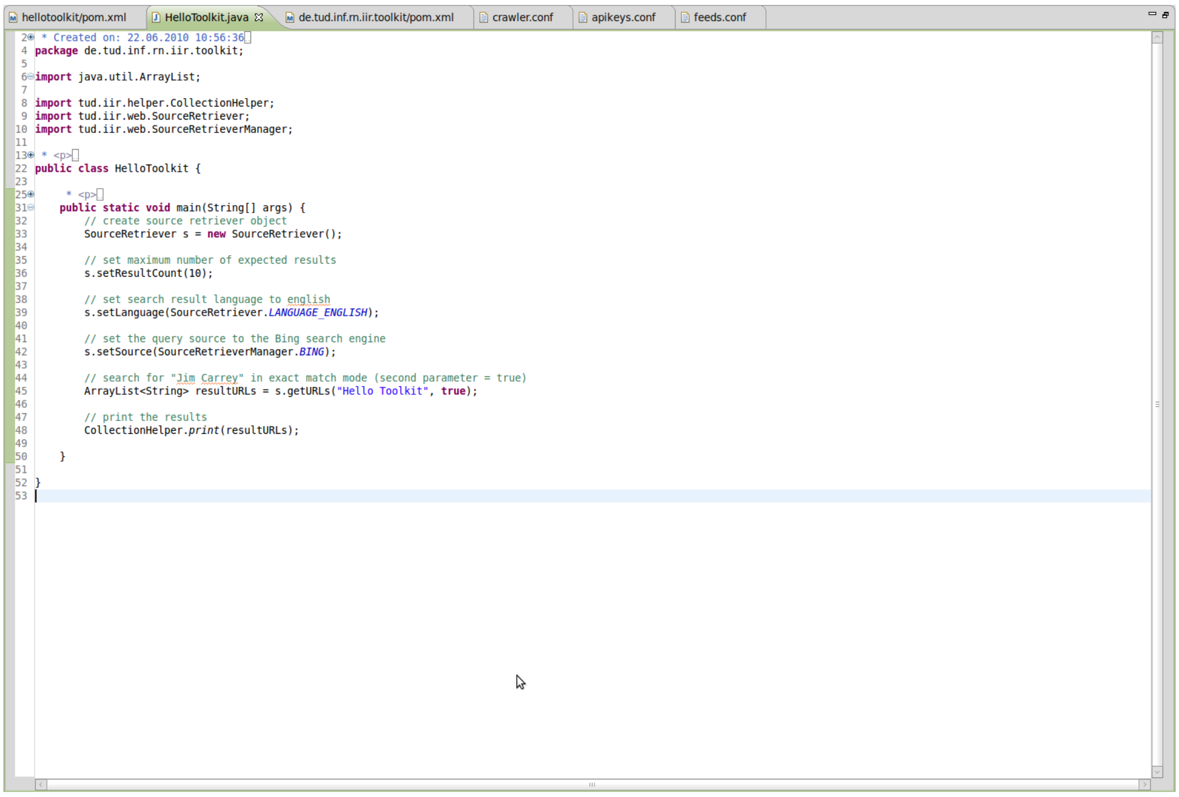
Classfile Generation

✓ Add variable attributes to generated class files (used by the debugger)
✓ Add line number attributes to generated class files (used by the debugger)
✓ Add source file name to generated class file (used by the debugger)
✓ Preserve unused (never read) local variables
✓ Inline finally blocks (larger class files, but improved performance)

Restore Defaults | Apply

Cancel | OK

Figure 2.10: configure Eclipse to use Java 1.6



```
 * Created on: 22.06.2010 10:56:36
package de.tud.inf.rn.iir.toolkit;

import java.util.ArrayList;

import tud.iir.helper.CollectionHelper;
import tud.iir.web.SourceRetriever;
import tud.iir.web.SourceRetrieverManager;

 * <p>
public class HelloToolkit {

     * <p>
    public static void main(String[] args) {
        // create source retriever object
        SourceRetriever s = new SourceRetriever();

        // set maximum number of expected results
        s.setResultCount(10);

        // set search result language to english
        s.setLanguage(SourceRetriever.LANGUAGE_ENGLISH);

        // set the query source to the Bing search engine
        s.setSource(SourceRetrieverManager.BING);

        // search for "Jim Carrey" in exact match mode (second parameter = true)
        ArrayList<String> resultURLs = s.getURLs("Hello Toolkit", true);

        // print the results
        CollectionHelper.print(resultURLs);

    }

}
```

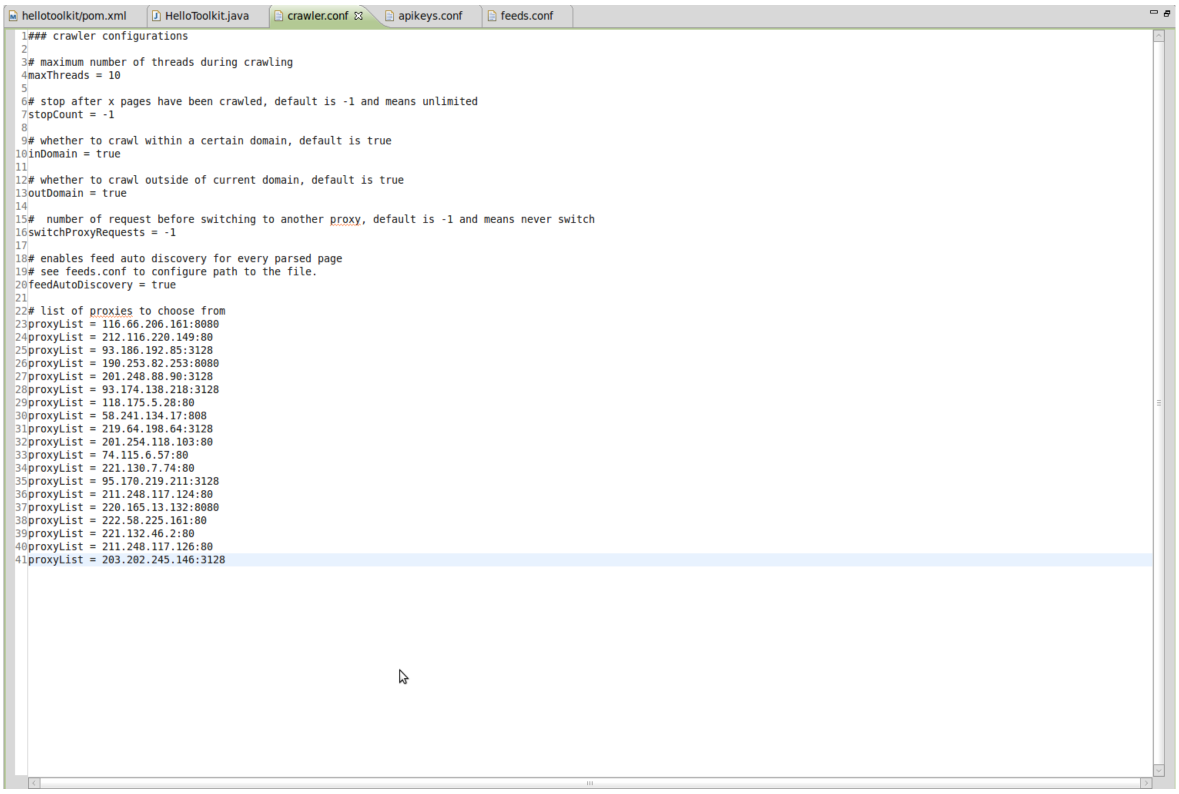
Figure 2.11: "Hello Toolkit" Code

depends on it[1]. As usual we will start with a very simple "Hello World" application. Create a new package `de.tud.inf.rn.iir.toolkit` and a new class `HelloToolkit` containing a `main` method. In this main method you can add actual toolkit code. We used the first example as described in Section 6.0.5 and search Bing for the term "Hello Toolkit". The example is also shown in Fig. 2.11. For the code to work you need three additional files that are already in the config folder in the toolkit project. These are “crawler.conf”, “apikeys.conf” and “feeds.conf”. You need to copy them to src/main/resources/config. The folder src/main/resources usually contains all resources that are not Java files but are required by your code. All files are shown in Fig. 2.12, 2.13 and 2.14. Fig. 2.15 shows the final directory and file structure of your project. With the structure from Fig. 2.15 you can run your project. Just open your projects context menu again, choose *Run As* → *Maven Clean*, then open context menu again choose *Run As* → *Maven Install* and finally choose *Run As* → *Java Application*.

2.4 Reporting issues using Mantis

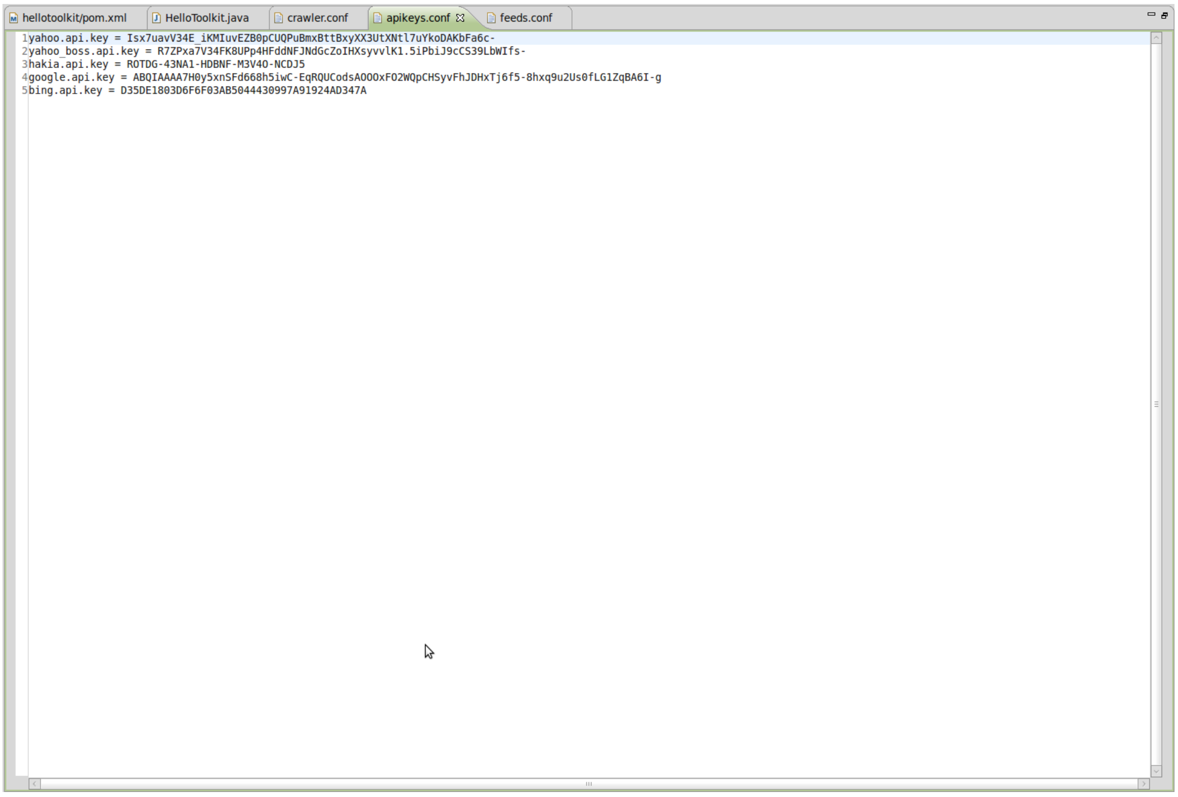
To report issues with the toolkit or to view issues assigned to you, you need to register with Mantis. To do this send an E-Mail to the administrator at klemens.muthmann@tu-dresden.de and provide your name, your advisor and the reason why you are working on the toolkit (i.e. thesis topic or research fellow). You will receive an E-Mail (not automatically so it might take some time) providing a link were you can set your password. If you successfully set a password for your account you can login on: Mantis.

[1]Of course you can change Mavens behaviour via the pom.xml but this requires additional configuration not covered by this document. Refer to the Maven documentation under `http://maven.apache.org/` for further information.



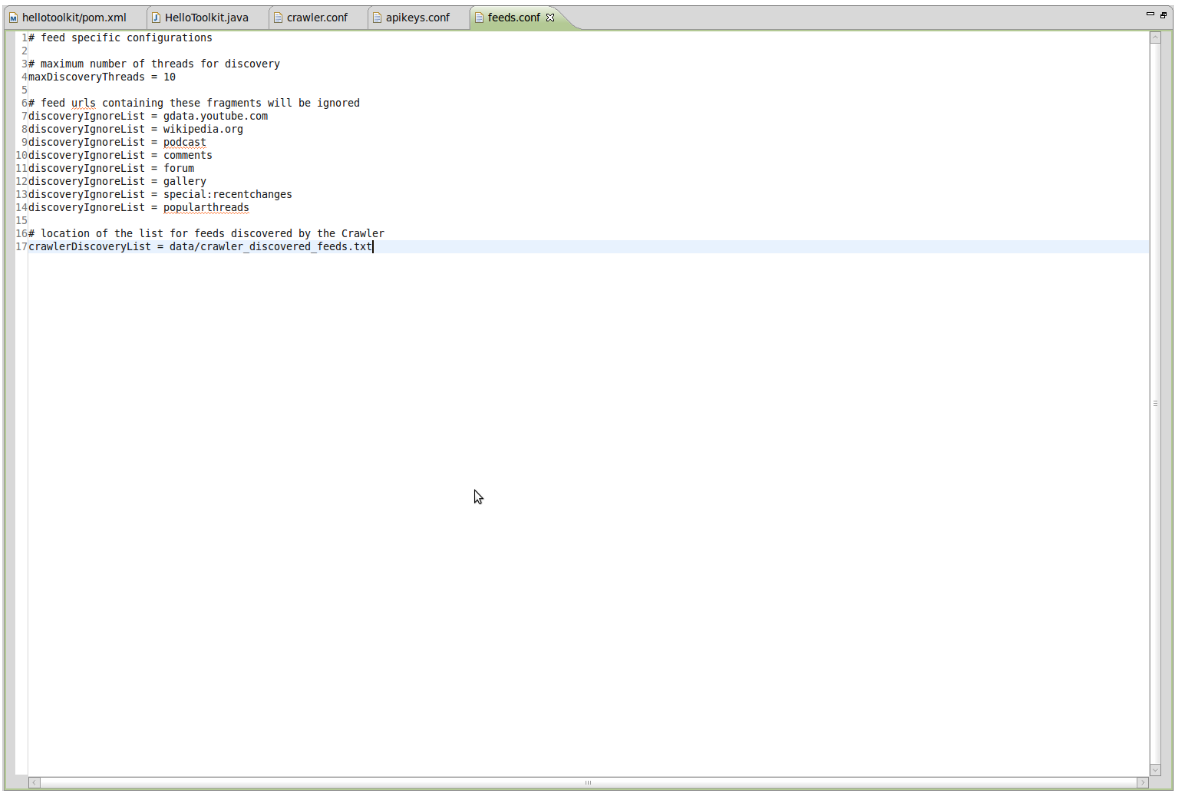
```
### crawler configurations

# maximum number of threads during crawling
maxThreads = 10

# stop after x pages have been crawled, default is -1 and means unlimited
stopCount = -1

# whether to crawl within a certain domain, default is true
inDomain = true

# whether to crawl outside of current domain, default is true
outDomain = true

#  number of request before switching to another proxy, default is -1 and means never switch
switchProxyRequests = -1

# enables feed auto discovery for every parsed page
# see feeds.conf to configure path to the file.
feedAutoDiscovery = true

# list of proxies to choose from
proxyList = 116.66.206.161:8080
proxyList = 212.116.220.149:80
proxyList = 93.186.192.85:3128
proxyList = 190.253.82.253:8080
proxyList = 201.248.88.90:3128
proxyList = 93.174.138.218:3128
proxyList = 118.175.5.28:80
proxyList = 58.241.134.17:808
proxyList = 219.64.198.64:3128
proxyList = 201.254.118.103:80
proxyList = 74.115.6.57:80
proxyList = 221.130.7.74:80
proxyList = 95.170.219.211:3128
proxyList = 211.248.117.124:80
proxyList = 220.165.13.132:8080
proxyList = 222.58.225.161:80
proxyList = 221.132.46.2:80
proxyList = 211.248.117.126:80
proxyList = 203.202.245.146:3128
```

Figure 2.12: Create a new Maven project



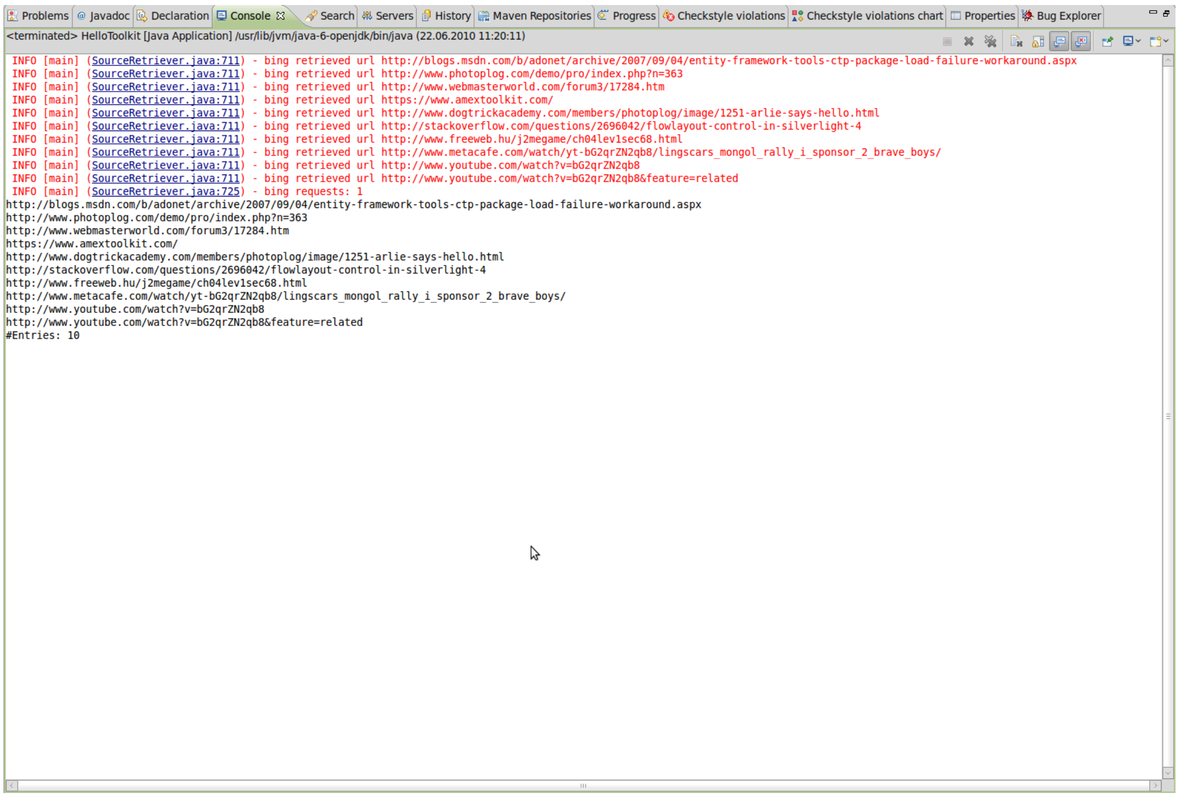
```
yahoo.api.key = Isx7uavV34E_iKMIuvEZB0pCUQPuBmxBttBxyXX3UtXNtl7uYkoDAKbFa6c-
yahoo_boss.api.key = R7ZPxa7V34FK8UPp4HFddNFJNdGcZoIHXsyvvlK1.5iPbiJ9cCS39LbWIfs-
hakia.api.key = ROTDG-43NA1-HDBNF-M3V4O-NCDJ5
google.api.key = ABQIAAAA7H0y5xnSFd668h5iwC-EqRQUCodsAOOOxFO2WQpCHSyvFhJDHxTj6f5-8hxq9u2Us0fLG1ZqBA6I-g
bing.api.key = D35DE1803D6F6F03AB5044430997A91924AD347A
```

Figure 2.13: Create a new Maven project



```
# feed specific configurations

# maximum number of threads for discovery
maxDiscoveryThreads = 10

# feed urls containing these fragments will be ignored
discoveryIgnoreList = gdata.youtube.com
discoveryIgnoreList = wikipedia.org
discoveryIgnoreList = podcast
discoveryIgnoreList = comments
discoveryIgnoreList = forum
discoveryIgnoreList = gallery
discoveryIgnoreList = special:recentchanges
discoveryIgnoreList = popularthreads

# location of the list for feeds discovered by the Crawler
crawlerDiscoveryList = data/crawler_discovered_feeds.txt
```

Figure 2.14: Create a new Maven project



```
<terminated> HelloToolkit [Java Application] /usr/lib/jvm/java-6-openjdk/bin/java (22.06.2010 11:20:11)
 INFO [main] (SourceRetriever.java:711) - bing retrieved url http://blogs.msdn.com/b/adonet/archive/2007/09/04/entity-framework-tools-ctp-package-load-failure-workaround.aspx
 INFO [main] (SourceRetriever.java:711) - bing retrieved url http://www.photoplog.com/demo/pro/index.php?n=363
 INFO [main] (SourceRetriever.java:711) - bing retrieved url http://www.webmasterworld.com/forum3/17284.htm
 INFO [main] (SourceRetriever.java:711) - bing retrieved url https://www.amextoolkit.com/
 INFO [main] (SourceRetriever.java:711) - bing retrieved url http://www.dogtrickacademy.com/members/photoplog/image/1251-arlie-says-hello.html
 INFO [main] (SourceRetriever.java:711) - bing retrieved url http://stackoverflow.com/questions/2696042/flowlayout-control-in-silverlight-4
 INFO [main] (SourceRetriever.java:711) - bing retrieved url http://www.freeweb.hu/j2megame/ch04lev1sec68.html
 INFO [main] (SourceRetriever.java:711) - bing retrieved url http://www.metacafe.com/watch/yt-bG2qrZN2qb8/lingscars_mongol_rally_i_sponsor_2_brave_boys/
 INFO [main] (SourceRetriever.java:711) - bing retrieved url http://www.youtube.com/watch?v=bG2qrZN2qb8
 INFO [main] (SourceRetriever.java:711) - bing retrieved url http://www.youtube.com/watch?v=bG2qrZN2qb8&feature=related
 INFO [main] (SourceRetriever.java:725) - bing requests: 1
http://blogs.msdn.com/b/adonet/archive/2007/09/04/entity-framework-tools-ctp-package-load-failure-workaround.aspx
http://www.photoplog.com/demo/pro/index.php?n=363
http://www.webmasterworld.com/forum3/17284.htm
https://www.amextoolkit.com/
http://www.dogtrickacademy.com/members/photoplog/image/1251-arlie-says-hello.html
http://stackoverflow.com/questions/2696042/flowlayout-control-in-silverlight-4
http://www.freeweb.hu/j2megame/ch04lev1sec68.html
http://www.metacafe.com/watch/yt-bG2qrZN2qb8/lingscars_mongol_rally_i_sponsor_2_brave_boys/
http://www.youtube.com/watch?v=bG2qrZN2qb8
http://www.youtube.com/watch?v=bG2qrZN2qb8&feature=related
#Entries: 10
```

Figure 2.15: Create a new Maven project

Chapter 3

Toolkit Structure

The TUDIIR Toolkit is managed using subversion. The top level folder structure follows the usual subversion layout using trunk for the main development, branches for parallel development and tags to mark specific versions. The trunk is located in our SVN[1]. The folder structure looks as follows.

```
toolkit
 |- config
 |- data
    |- knowledgeBase
    |- models
    |- test
 |- documentation
    |- handout
    |- javadoc
    |- other
 |- exe
 |- libs
 |- src
    |- main
       |- java
    |- test
       |- java
 |- dev
```

3.1 Config Folder

The config folder contains configuration files for several components of the toolkit. The files are explained in the following sections.

3.1.1 apikeys.conf

The api keys that are used by the toolkit components are specified here. You may need to apply for API keys at the provider's page.

```
yahoo.api.key =
yahoo_boss.api.key =
hakia.api.key =
google.api.key =
bing.api.key =
```

[1] https://141.76.40.86/svn-students/iircommon/toolkit/trunk

3.1.2 classification.conf

Classification settings for the tud.iir.classification.page.ClassifierManager.

```
# percentage of the training/testing file to use as training data
page.trainingPercentage = 80

# create dictionary on the fly (lowers memory consumption but is slower)
page.createDictionaryIteratively = false

# alternative algorithm for n-gram finding (lowers memory consumption but is slower)
page.createDictionaryNGramSearchMode = true

# index type of the classifier:
# 1: use database with single table (fast but not normalized and more disk space needed)
# 2: use database with 3 tables (normalized and less disk space needed but slightly slower)
# 3: use lucene index on disk (slow)
page.dictionaryClassifierIndexType = 1
```

3.1.3 crawler.conf

Crawler settings for the tud.iir.web.Crawler. All settings can be set in Java code as well.

```
# maximum number of threads during crawling
maxThreads = 10

# stop after x pages have been crawled, default is -1 and means unlimited
stopCount = -1

# whether to crawl within a certain domain, default is true
inDomain = true

# whether to crawl outside of current domain, default is true
outDomain = true

#  number of request before switching to another proxy, default is -1 and means never switch
switchProxyRequests = -1

# list of proxies to choose from
proxyList = 83.244.106.73:8080
proxyList = 83.244.106.73:80
proxyList = 67.159.31.22:8080
```

3.1.4 db.conf

Database settings for the tud.iir.persistence.DatabaseManager.

```
db.type = mysql
db.driver = com.mysql.jdbc.Driver
db.host = localhost
db.port = 3306
db.name = toolkitdb
db.username = root
db.password = rootpass
```

3.1.5 general.conf

General settings used by the tud.iir.control.Controller.

3.2 Data Folder

The data folder contains files that are used during runtime of several components.

3.2.1 knowledgeBase

The knowledge base folder contains the OWL ontology files used for the extraction tasks in the tud.iir.extraction package.

3.2.2 models

The models folder contains learned models that can be reused.

3.2.3 Temp Folder

The temp folder is not part of the repository. Some functions may however create this folder and write temporary data.

3.2.4 test

The test folder contains data that is used for running jUnit tests.

3.3 Documentation Folder

The documentation folder contains help files to understand the toolkit. This very document is located in the handout folder and a Javadoc can be found there too.

3.4 Exe Folder

The exe folder contains all runnable jar files in separate folders including a sample script to run the program and a readme.txt that explains the run options.

3.5 Libs Folder

The libs folder contains all referenced libs used by the toolkit.

3.6 Src Folder

The src folder contains all source files of the toolkit. You may need to put the log4j.properties file here in order to use custom logging settings. Alternatively you include the config folder in the class path.

Chapter 4

Conventions

4.1 Coding Standards

To keep the code readable and easy to understand for other developers, we use the following coding guidelines.

1. All text is written in (American) English (color instead of colour).
2. Variables and method names should be camel cased and not abbreviated (*computeAverage* instead of *comp_avg*).
3. There must be a space after a comma.
4. There must be a line break after opening { braces.
5. Static fields must be all uppercase. There should be an _ for longer names (*STATIC_FIELD*).
6. Each class must have a comment including the author name.
7. Methods with very simple, short code do not need to have comments (getters and setters) all other methods should have an explaining comment with @param explanation and @return.
8. Avoid assignments (=) inside if and while conditions.
9. Statements after conditions should always be in braces ({})

The following listing shows an example class with applied coding standards. Please also have a look at [?] for a quick overview of best practices.

```
/**
 * This is just an example class.
 * It is here to show the coding guidelines.
 *
 * @author Forename Name
 */
public class ExampleClass implements Example {

        // this field holds all kinds of brackets
        private static final char[] BRACKET_LIST = {'(', ')'};

        /**
         * This is just and example method.
         *
         * @param timeString A string with a time.
         * @return True if no error occurred, false otherwise.
         */
        public boolean computeAverageTime(String timeString) {
                if (hours < 24 && minutes < 60 && seconds < 60) {
                        return true;
                } else {
                        return false;
                }
        }
}
```

4.2 Eclipse Plugins for better Coding

It is a very good practice to install the following eclipse plugins to check ones own code before committing:

1. CodeFormatter is a simple file that can be loaded into Eclipse to format the source code correctly. Go to Eclipse ▶ Window ▶ Preferences ▶ Java ▶ Code Style ▶ Formatter and import the "tudiir_eclipse_formatter.xml" from the dev folder. Pressing Control+F formats the source code of a selected class.

2. Checkstyle [1] checks styles of the code, whether JavaDoc comments are set etc. After installing you should go to the preferences section of Checkstyle in Enclipse and load the "checkstyle_config.xml" from the dev folder.

3. PMD[2] tells you what is wrong with your code in terms of common violations, forgotten initializations and much more. After installing you should go into the preferences of PMD and load the ruleset file "pmd_ruleset" from the dev folder.

4. FindBugs[3] is similar to PMD but focuses on severe errors only. After installing you don't need to configure anything.

Checkstyle, PMD, and FindBugs can be initiated by right clicking a package or class file and selecting the plugin. The violations will be shown so that you can eliminate them.

Using these plugins raises the chances to win in the continuous integration game as described in Section 2.2.1.

4.2.1 Tests

To guarantee that all components work as expected, we use jUnit tests. Before major check-ins to the repository, all jUnit Tests must run successfully. Run the tud.iir.control.AllTests.java to make sure all components work correctly. After finishing a new component, new testing code must be written.

[1] `http://eclipse-cs.sourceforge.net/downloads.html`
[2] `http://pmd.sourceforge.net/eclipse/`
[3] `http://findbugs.cs.umd.edu/eclipse/`

Chapter 5

Toolkit Functionality

5.1 Classification

5.1.1 Text Classification

Text classification is the process of assigning one or more categories to a given text document. There are several types of text classification which are shown in Figure 5.1. The TOOLKIT can classify text in a single category, multiple categories, or in a hierarchical manner.

The text classification components are built from scratch and do not rely on external libraries such as Weka. In this section we will explain which features can be used for the classification, the basic theory of the classifiers, and how the performance of a classifier can be evaluated. In each section, we will describe theory and how the classification components can be used programmatically.

Classification Type

The TOOLKIT supports simple single category classification, tagging, and hierarchical classification. Each classifier needs to know in which type it has to perform classification. This information is stored in the $tud.iir.classification.page.evaluation.ClassificationTypeSetting$ object which is then passed to the classifier. Please read the Javadoc of that class for more detailed information.

Features

Features are the input for a classifier. In text classification we have a long string as an input from which we can derive several features. All of the TOOLKIT classifiers work with n-grams. N-grams are sets of tokens of the length n. The TOOLKIT can preprocess text with character or word-level n-grams:

1. **Character level n-grams** use each character of the string as a token. For example, from the string "It is sunny" we can create the following set of 3-grams: $it, ti, is, is, ss, su, sun, unn, nny$. The number of n-grams in a set can be calculated as $ngrams = numberOfTokens - n + 1$. In our example that means $9 = 11 - 3 + 1$.

2. **Word level n-grams** use each word (separated with white space) of the string as a token. For example, from the string "It is so nice and sunny today" we can create the following set of 3-grams: $Itisso, issonice, soniceand, niceandsunny, andsunnytoday$. The number of n-grams in a set can be calculated as $ngrams = numberOfTokens - n + 1$. In our example that means $5 = 7 - 3 + 1$. If you want to have single words as features for the text classifier you can simply use unigrams or bigrams which are n-grams with $n = 1$ or $n = 2$ respectively.

The document preprocessor allows you to create a set of n-grams with different length too. For example, you can create all 2-grams, 3-grams, and 4-grams for the given input text and use them as features for the classifier.

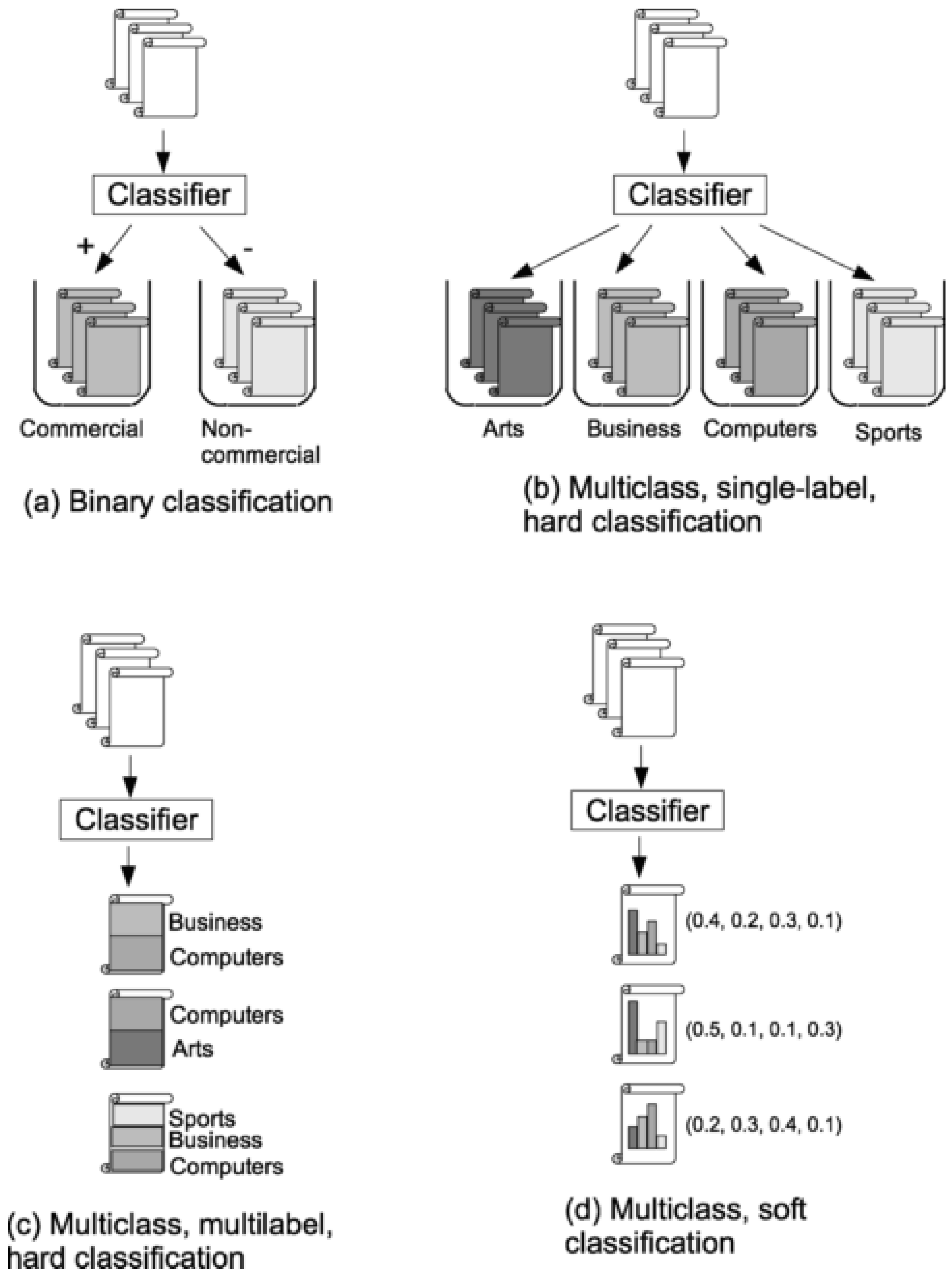


Figure 5.1: Types of classification[?].

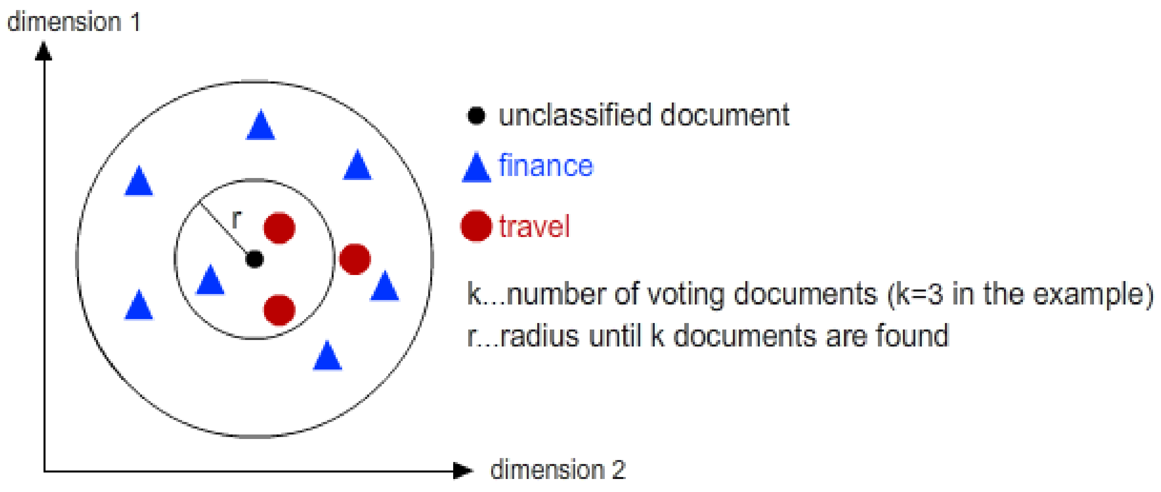

dimension 1
dimension 2
r
● unclassified document
▲ finance
● travel
k...number of voting documents (k=3 in the example)
r...radius until k documents are found

Figure 5.2: A simple KNN example.

Sometimes you do not want to have all n-grams to be features for the classifier. You can simply disallow certain features by putting them in a stop word list. These n-grams will then be ignored for the classification.

All these settings are stored in a *tud.iir.classification.page.evaluation.FeatureSetting* object which is passed to the classifier. Please read the Javadoc of that class for more detailed information.

Text Classifiers

The text classifiers perform the actual classification task by calculating the most relevant category (or categories) for the input document. Two text classifiers are implemented, a dictionary based classifier and a k-nearest neighbor classifier. Both are explained in more detail in the following paragraphs.

K-Nearest Neighbor Text Classifier The KNN classifier uses the n-grams of the training documents to place them in a high dimensional vector space. The dimensions of the space equal the total number of available n-grams. Each training document is therefore a vector in that highly dimensional space. A new, unclassified document is now put into that vector space and by using distance function the k nearest neighbors are found for that document. Each of these neighbors votes with its own class, the more votes for one class the more likely that the new document belongs to that class.

Figure 5.2 shows a simple example of how the KNN classifier works. We limited the dimension to two for easier understanding. In this scenario we want to classify the given document (black dot in the middle) into one of the two categories “finance” (blue triangles) or “travel” (red dots). We calculate the distance between the new document and all training documents and consider the votes of the nearest three. In the example, two of these three document vote for “travel” which would let us classify our input document into that class.

The distance between two documents is calculated as shown in Equation 5.1 where $d1$ and $d2$ are the two documents. The shorter the distance, the more similar the documents.

$$distance(d1, d2) = \frac{1}{numberOfMatchingNGrams} \tag{5.1}$$

Dictionary-Based Classifier The dictionary-based classifier[1] learns how probable each n-gram is for each given category and assigns the most probable category (or categories) to the input document.

[1]This classifier won the first Research Garden (`http://www.research-garden.de`) competition where the goal was to classify product descriptions into 8 different categories.

A dictionary is built at training stage by counting and normalizing the co-occurrences of one n-gram and a category. The dictionary might then look as shown in Table 5.1 where each column is a category (finance, travel, and science) and each row is an n-gram. In each cell we now have the learned relevance for each n-gram and category $relevance(ngram, category)$. The sum of the relevances in each row must add up to one.

Table 5.1 shows an example dictionary matrix. The n-gram “money” is more likely to get the category “finance” ($relevance(money, finance) = 0.6$) than “science” ($relevance(money, science) = 0.25$) while the n-gram “beach” is most likely to appear in the category “travel” ($relevance(beach, travel) = 0.85$.

Table 5.1: N-Gram dictionary with relevances for categories.

n-gram	finance	travel	science
money	0.6	0.15	0.25
beach	0.1	0.85	0.05
paper	0.3	0.2	0.5

To classify a new document, we again create all n-grams, look up the relevance scores in the dictionary and assign the categories with the highest probability. The probability for each category and given document is calculated as shown in Equation 5.2 where $N_{Document}$ is the set of n-grams for the given document.

$$CategoryProbability(category, document) = \sum_{n \in Nurl} relevance(n, category) \tag{5.2}$$

The dictionary can be stored in memory, in an embedded H2 database, or in a client/server MySQL database. These settings can be made in the classification.conf file in the conf folder (see Section 3.1).

Evaluation

In order to find out which classifier works best with which feature settings, you can evaluate these combinations. The *tud.iir.classification.page.ClassifierManager* has the *learnBestClassifier* method to run the evaluation on all given classifiers with the given evaluation setting object *tud.iir.classification.page.Evaluation* See Section 5.1.1 for more details of how to perform the evaluation programmatically.

The output of the evaluation will be three csv files that hold information about the combinations of classifier, dataset, training percentage, and the final performance for the combination. The performance is measured in precision, recall, and F1. The three files are stored in the data/temp folder and hold the following information.

averagePerformancesDatasetTrainingFolds.csv This file holds the performance measures for each classifier, averaged over all given datasets, training percentages, and folds in the cross validation.

averagePerformancesTrainingFolds.csv This file holds the performance measures for each classifier and dataset combination, averaged over all given training percentages and folds in the cross validation.

averagePerformancesFolds.csv This file holds the performance measures for each classifier, dataset, and training percentage combination, averaged over all given folds in the cross validation.

Best Practices

5.2 Extraction

Fact Extraction

Date Extraction

Named Entity Recognition

5.3 Retrieval

Web Crawling

Web Information Retrieval

Feed Retrieval

5.4 Preprocessing

5.4.1 Sentence Splitting

5.4.2 Tokenization

5.4.3 Creating N-Grams

5.4.4 Noun Pluralization and Singularization

Chapter 6

More Example Usages

Learning to use a toolkit often does not work by reading the Javadoc. Therefore, we provide a simple example how to setup your first project using the toolkit, eclipse and maven. Afterwards we provide a short overview of some important modules and how you can use them. All examples are intended as entry points to facilitate the first use of the toolkit. Many components are too mighty to be explained exhaustively in small code snippets, make sure to read the Javadoc and more importantly the actual code for further information.

6.0.5 Source Retriever

The source retriever is a module that can query a number of sources such as search engines and web pages with terms and retrieve matching results.

Basic Features

Basic features are:

- Query the Google search engine (unlimited queries, top 64 results only).
- Query Yahoo search engine (5000 queries per IP and day, top 1000 results).
- Query Bing search engine (unlimited)
- Query Hakia search engine.
- Query Twitter.
- Query Google Blog search.
- Query Textrunner web page.

Some of the search APIs require API keys which must be specified in the config/apikeys.conf file. See 3.1.1 for more information.

How To

The following code snippet shows how to initialize the source retriever and get a list of (English) URLs from the Bing search engine for the exact search "Jim Carrey".

```
// create source retriever object
SourceRetriever s = new SourceRetriever();

// set maximum number of expected results
s.setResultCount(10);
```

```

// set search result language to english
s.setLanguage(SourceRetrieve.LANGUAGE_ENGLISH);

// set the query source to the Bing search engine
s.setSource(SourceRetrieverManager.BING);

// search for "Jim Carrey" in exact match mode (second parameter = true)
ArrayList<String> resultURLs = s.getURLs("Jim Carrey", true);

// print the results
CollectionHelper.print(resultURLs);
```

6.0.6 Web Crawler

The web crawler can be used to crawl domains or just retrieve the cleansed HTML document of a single web page.

Basic Features

Basic functionalities include:

- Download and save contents of a web page.
- Automatically crawl in- and/or outbound links from web pages.
- Use URL rules for the crawling process.
- Extract title, description, keywords and body content of a web page.
- Remove HTML, SCRIPT and CSS tags.
- Find a sibling page of a given URL.
- Switch proxies after a certain number of requests to avoid being blocked.

How To

The following code shows how to instantiate a simple crawler that starts at http://www.dmoz.org and follows all in- and outbound links. The URL of each crawled page is printed to the screen. The crawler will use 10 threads, changes the proxy after every third request and stops after having crawled 1000 pages. Instead of setting the parameters using the code, we can also specify them in the config/crawler.conf file. See 3.1.2 for more information.

```
// create the crawler object
Crawler c = new Crawler();

// create a callback that is triggered for every crawled page
CrawlerCallback crawlerCallback = new CrawlerCallback() {
        @Override
        public void crawlerCallback(Document document) {
                // TODO do something with the page
                System.out.println(document.getDocumentURI());
        }
};
c.setCrawlerCallback(crawlerCallback);

```

```
// stop after 1000 pages have been crawled (default is unlimited)
c.setStopCount(1000);

// set the maximum number of threads to 10
c.setMaxThreads(10);

// the crawler should automatically use different proxies
// after every 3rd request (default is no proxy switching)
c.setSwitchProxyRequests(3);

// set a list of proxies to choose from
List<String> proxyList = new ArrayList<String>();
proxyList.add("83.244.106.73:8080");
proxyList.add("83.244.106.73:80");
proxyList.add("67.159.31.22:8080");
c.setProxyList(proxyList);

// start the crawling process from a certain page,
// true = follow links within the start domain
// true = follow outgoing links
c.startCrawl("http://www.dmoz.org/", true, true);
```

6.0.7 FAQ Extractor

The FAQ extractor can extract question-answer pairs from several structured frequently asked questions pages on websites. The usage is quite simple as shown in the following listing.

```
// create a list of question answer pairs
ArrayList<QA> qas = null;

// the URL that contains the FAQ
String url = "http://blog.pandora.com/faq/";

// start extracting question and answers from the URL
qas = QAExtractor.getInstance().extractFAQ(url);

// print the extracted questions and answers
CollectionHelper.print(qas);
```

6.0.8 Fact Extraction

The Fact extractor can be used to detect facts in tables on web pages given a URL and optionally a small set of seed attribute names that help the extractor.

```
// the URL of the facts
String url = "http://en.wikipedia.org/wiki/Nokia_N95";

// the concept of the attributes
Concept c = new Concept("Mobile Phone");

// a small list of seed attributes
HashSet<Attribute> seedAttributes = new HashSet<Attribute>();
int attributeType = Attribute.VALUE_STRING;
seedAttributes.add(new Attribute("Second camera", attributeType, c));
seedAttributes.add(new Attribute("Memory card", attributeType, c));
seedAttributes.add(new Attribute("Form factor", attributeType, c));

// detect the facts using the seeds from the URL
ArrayList<Fact> detectedFacts = null;
detectedFacts = FactExtractor.extractFacts(url, seedAttributes);

// print the extracted facts
CollectionHelper.print(detectedFacts);
```

6.0.9 Web Page Classification

The Web Page Classification module can be used to classify web pages by their URL or their full content.

Basic Features

The Web Page Classification module has the following basic features:

- Classify web pages by its URL only.
- Classify web pages by its full content.
- Use a combination of URL and full content for classification.
- Learn, test and reuse models.
- Simple one-category classification.
- Hierarchical classification.
- Multi-category classification (tagging).
- All algorithms are language independent.

How To

This section describes how to prepare training and testing data to learn a model and test the classifier.

Preparing the Training/Testing Data The data can be specified in a simple text file. There are three classification options, namely, one-category classification, hierarchical classification and multi-category classification. They all require a similar structure of the data.

One-Category Classification We write one URL and one category separated with a single space on each line. For example:

```
http://www.google.com search
http://www.fifa.com sport
http://www.oscars.com entertainment
```

Hierarchical Classification We write one URL and multiple categories separated with a single space on each line. The categories must be in the correct order, so the first category is the main one, all following are subcategories of each other. For example:

```
http://www.google.com search search_engine
http://www.fifa.com sport team_sports soccer
http://www.oscars.com entertainment movies awards usa
```

Multi-Category Classification We write one URL and multiple categories separated with a single space on each line. The order of the categories (tags) does not matter. For example:

```
http://www.google.com search image_search video_search
http://www.fifa.com soccer sport free_time fun ball_game results to_read
http://www.oscars.com entertainment movies films awards watch video stars
```

Building the Model The model is an internal representation of the learned data. After learning a model, a classifier can applied to unseen data. We now have prepared the training and testing data so we can now learn the models. The results of the test will be printed to the console and written to a log file under data/logs. The classifier is saved as a lucene index or a database under the name "dictionary_Xclassifier_Y" where X is "url", "fullpage" or "combined" and Y is 1 (one-category), 2 (hierarchical) or 3 (multi-category). The model will be written to data/models. How the file is saved can be configured in the config/classification.conf file. See 3.1.2 for more information.

```java
// create a classifier mananger object
ClassifierManager classifierManager = new ClassifierManager();

// use 80% of the data in the training/testing file as training data
// the rest is used for testing
classifierManager.setTrainingDataPercentage(80);

// specify location of training/testing file
String ttFile = "training_testing_file.txt";

// build and test the model
// the second parameter specifies that we want to use URL features only
// the third parameter specifies that we want to use
// multi-category classification (tagging)
// the last parameter is set to true in order to train not just test it
classifierManager.trainAndTestClassifier(ttFile,
                                         WebPageClassifier.URL,
                                         WebPageClassifier.TAG,
                                         true);
```

Using the Model After we trained a model for a classifier we can apply it to unseen data. Let's use the model we trained for a URL classifier with tagging:

```java
// create the URL classifier
WebPageClassifier classifier = new URLClassifier();

// create a classification document
ClassificationDocument classifiedDocument = null;

// the web page to be classified
String url = "http://en.wikipedia.org/wiki/Computer";

// use the classifier to classify a web page to multiple categories
// using URL features only
classifiedDocument = classifier.classify(url, WebPageClassifier.TAG);

// print out classification results
System.out.println(classifiedDocument);
```

6.0.10 Helpers

The toolkit contains many helper functionalities for reoccurring tasks in the tud.iir.helper package. The following code snippet shows several sample usages of some of the functions.

```java
// sort a map by its value in ascending order (2nd parameter = true)
Map m = CollectionHelper.sortByValue(map, true);

// reverse a list
List l = CollectionHelper.reverse(list);

// print the contents of a collection
CollectionHelper.print(collection);

// get the runtime of an algorithm and print it (2nd parameter = true)
long startTime = System.currentTimeMillis();
for (int i = 0; i < 10000; i++) {
        int c = i * 2;
}
DateHelper.getRuntime(t1, true);

// (de) serialization of objects
FileHelper.serialize(obj, "obj.ser");
Object obj = FileHelper.deserialize("obj.ser");

// rename, copy, move and delete files
FileHelper.rename(new File("a.txt"), "b.txt");
FileHelper.copyFile("src.txt", "dest.txt");
FileHelper.move(new File("src.txt"), "dest.txt");
FileHelper.delete("src.txt");

// get files from a folder
File[] files = FileHelper.getFiles("folder");

// zip and unzip a text
FileHelper.zip("text", "zipFile.zip");
String t = FileHelper.unzipFileToString("zipFile.zip");

// perform some action on every line of an ASCII file
final Object[] obj = new Object[1];
obj[0] = 1;

LineAction la = new LineAction(obj) {

    @Override
    public void performAction(String line, int lineNumber) {
        System.out.println(lineNumber + ": " + line + " " + obj[0]);
    }
}
FileHelper.performActionOnEveryLine(filePath, la);

// round a number with a number of digits
double r = MathHelper.round(2.3333, 2);

// calculate n-grams of a string
Set<String> nGrams = StringHelper.calculateNGrams("abcde", 3);

```

```java
// English singular word to plural
String p = StringHelper.wordToPlural("city");

// remove HTML tags
String r = StringHelper.removeHTMLTags("<a>abc</a>",
                                        true, true
                                        true, true);

// trim a string
String t = StringHelper.trim("␣␣to␣trim++++");

// get singular or plural of an English word
String plural = StringHelper.wordToPlural("city");
String singular = StringHelper.wordToSingular("cities");

// reverse a string
String r = StringHelper.reverse("abc");

// encode and decode base64
String e = StringHelper.encodeBase64("abc");
String d = StringHelper.decodeBase64(e);
```

6.1 Reference Libraries

The TUDIIR Toolkit makes excessive use of third party libraries. We do not intend to re-implement code but rather to built on it and create something superior. Here an incomplete list of libraries the toolkit uses:

- Apache Commons [?] for many standard tasks in string and number manipulation and more.
- Fathom [?] to measure readability of English text.
- iText [?] for creating PDF documents.
- Jena [?] for reading and writing ontology files.
- jYaml [?] to read and write YAML files.
- Log4j [?] for logging.
- Lucene [?] for indexing and making learned models persistent.
- NekoHTML [?] to clean up the HTML of web pages in order to process them correctly.
- SimMetrics [?] to calculate similarities of strings.
- Twitter4j [?] to query the Twitter API.
- Weka [?] for machine learning.

6.2 History

The foundation of the toolkit code came out of the WebKnox project[?] that was started in 2008. The code is in development by students of the Dresden University of Technology. Contributors are:

- Christopher Friedrich
- Martin Gregor

- Philipp Katz
- David Urbansky
- Robert Willner
- Martin Werner

Chapter 7

References